ESP32-P4

ESP-IDF Programming Guide

Release v5.3.2
乐鑫信息科技
2024 年 12 月 06 日

Table of contents

Table of contents i

1 快速入门 **3**
1.1 概述 . . . 3
1.2 准备工作 . . . 3
1.2.1 硬件: . . . 3
1.2.2 软件: . . . 4
1.3 安装 . . . 4
1.3.1 IDE . . . 4
1.3.2 手动安装 . . . 4
1.4 编译第一个工程 . . . 29
1.5 卸载 ESP-IDF . . . 30

2 API 参考 **31**
2.1 API 约定 . . . 31
2.1.1 错误处理 . . . 31
2.1.2 配置结构体 . . . 31
2.1.3 私有 API . . . 33
2.1.4 示例项目组件 . . . 33
2.1.5 API 稳定性 . . . 33
2.2 应用层协议 . . . 34
2.2.1 ASIO 端口 . . . 34
2.2.2 ESP-Modbus . . . 34
2.2.3 ESP-MQTT . . . 35
2.2.4 ESP-TLS . . . 53
2.2.5 ESP HTTP 客户端 . . . 72
2.2.6 ESP 本地控制 . . . 89
2.2.7 ESP 串行从机链路 . . . 98
2.2.8 ESP x509 证书包 . . . 112
2.2.9 HTTP 服务器 . . . 115
2.2.10 HTTPS 服务器 . . . 143
2.2.11 ICMP 回显 . . . 148
2.2.12 mDNS 服务 . . . 153
2.2.13 Mbed TLS . . . 153
2.2.14 IP 网络层协议 . . . 155
2.3 错误代码参考 . . . 155
2.4 连网 API . . . 162
2.4.1 以太网 . . . 162
2.4.2 Thread . . . 207
2.4.3 ESP-NETIF . . . 218
2.4.4 IP 网络层协议 . . . 252
2.4.5 应用层协议 . . . 255
2.5 外设 API . . . 255
2.5.1 模数转换器 (ADC) 单次转换模式驱动 . . . 255
2.5.2 模数转换器 (ADC) 连续转换模式驱动 . . . 266
2.5.3 模数转换器 (ADC) 校准驱动程序 . . . 274
2.5.4 Analog Comparator . . . 277

2.5.5 时钟树 . . . 287
2.5.6 事件任务矩阵 (ETM) . . . 306
2.5.7 GPIO & RTC GPIO . . . 315
2.5.8 通用定时器 . . . 339
2.5.9 专用 GPIO . . . 355
2.5.10 哈希消息认证码 (HMAC) . . . 359
2.5.11 数字签名 (DS) . . . 363
2.5.12 Inter-Integrated Circuit (I2C) . . . 369
2.5.13 I2S . . . 394
2.5.14 Image Signal Processor . . . 442
2.5.15 LCD . . . 455
2.5.16 低压差线性稳压器 (LDO) . . . 483
2.5.17 LED PWM 控制器 . . . 486
2.5.18 Camera Controller Driver . . . 509
2.5.19 电机控制脉宽调制器 (MCPWM) . . . 516
2.5.20 并行 IO . . . 570
2.5.21 脉冲计数器 (PCNT) . . . 585
2.5.22 Pixel-Processing Accelerator (PPA) . . . 600
2.5.23 红外遥控 (RMT) . . . 614
2.5.24 SD Pull-up Requirements . . . 642
2.5.25 SDMMC 主机驱动 . . . 643
2.5.26 SD SPI 主机驱动程序 . . . 650
2.5.27 Sigma-Delta 调制器 (SDM) . . . 656
2.5.28 SPI flash API . . . 661
2.5.29 SPI 主机驱动程序 . . . 694
2.5.30 SPI 从机驱动程序 . . . 716
2.5.31 SPI 从机半双工模式 . . . 723
2.5.32 JPEG Encoder and Decoder . . . 731
2.5.33 温度传感器 . . . 748
2.5.34 电容式触摸传感器 . . . 757
2.5.35 Two-Wire Automotive Interface (TWAI) . . . 780
2.5.36 通用异步接收器/发送器 (UART) . . . 803
2.5.37 USB 设备栈 . . . 829
2.5.38 USB 主机 . . . 833
2.6 项目配置 . . . 891
2.6.1 简介 . . . 891
2.6.2 项目配置菜单 . . . 891
2.6.3 使用 sdkconfig.defaults . . . 891
2.6.4 Kconfig 格式规定 . . . 891
2.6.5 Kconfig 选项的向后兼容性 . . . 892
2.6.6 配置选项参考 . . . 892
2.7 配网 API . . . 1240
2.7.1 协议通信 . . . 1240
2.8 存储 API . . . 1252
2.8.1 FAT 文件系统 . . . 1253
2.8.2 量产程序 . . . 1264
2.8.3 非易失性存储库 . . . 1268
2.8.4 NVS 加密 . . . 1291
2.8.5 NVS 分区生成程序 . . . 1297
2.8.6 NVS 分区解析程序 . . . 1302
2.8.7 SD/SDIO/MMC 驱动程序 . . . 1303
2.8.8 分区 API . . . 1311
2.8.9 SPIFFS 文件系统 . . . 1320
2.8.10 虚拟文件系统组件 . . . 1325
2.8.11 磨损均衡 API . . . 1341
2.9 系统 API . . . 1344
2.9.1 应用程序镜像格式 . . . 1344
2.9.2 引导加载程序镜像的格式 . . . 1350

2.9.3 应用级追踪 . . . 1353
2.9.4 使用外部堆栈调用函数 . . . 1358
2.9.5 芯片版本 . . . 1360
2.9.6 控制台终端 . . . 1364
2.9.7 eFuse Manager . . . 1374
2.9.8 错误代码和辅助函数 . . . 1405
2.9.9 ESP HTTPS OTA 升级 . . . 1408
2.9.10 事件循环库 . . . 1415
2.9.11 FreeRTOS 概述 . . . 1428
2.9.12 FreeRTOS (IDF) . . . 1430
2.9.13 FreeRTOS（附加功能） . . . 1545
2.9.14 堆内存分配 . . . 1572
2.9.15 基于 MMU 的存储管理 . . . 1587
2.9.16 Memory Synchronization . . . 1593
2.9.17 堆内存调试 . . . 1597
2.9.18 ESP 定时器 . . . 1609
2.9.19 内部 API 和不稳定的 API . . . 1614
2.9.20 处理器间调用 (IPC) . . . 1616
2.9.21 中断分配 . . . 1621
2.9.22 日志库 . . . 1628
2.9.23 杂项系统 API . . . 1636
2.9.24 空中升级 (OTA) . . . 1654
2.9.25 电源管理 . . . 1666
2.9.26 POSIX Thread . . . 1673
2.9.27 随机数发生器 . . . 1678
2.9.28 睡眠模式 . . . 1681
2.9.29 SoC 功能 . . . 1696
2.9.30 系统时间 . . . 1718
2.9.31 异步内存复制 . . . 1724
2.9.32 ULP LP-Core 协处理器编程 . . . 1729
2.9.33 看门狗 . . . 1752

3 H/W 硬件参考 1759

4 API 指南 1761
4.1 应用层跟踪库 . . . 1761
4.1.1 概述 . . . 1761
4.1.2 运行模式 . . . 1761
4.1.3 配置选项与依赖项 . . . 1762
4.1.4 如何使用此库 . . . 1763
4.2 应用程序的启动流程 . . . 1771
4.2.1 一级引导程序 . . . 1771
4.2.2 二级引导程序 . . . 1771
4.2.3 应用程序启动阶段 . . . 1772
4.3 引导加载程序 (Bootloader) . . . 1773
4.3.1 引导加载程序兼容性 . . . 1773
4.3.2 日志级别 . . . 1774
4.3.3 恢复出厂设置 . . . 1774
4.3.4 从测试固件启动 . . . 1775
4.3.5 回滚 . . . 1775
4.3.6 看门狗 . . . 1775
4.3.7 引导加载程序大小 . . . 1775
4.3.8 从深度睡眠中快速启动 . . . 1776
4.3.9 自定义引导加载程序 . . . 1776
4.4 构建系统 . . . 1776
4.4.1 概述 . . . 1776
4.4.2 使用构建系统 . . . 1777
4.4.3 示例项目 . . . 1778

4.4.4 项目 CMakeLists 文件 . . . 1779
4.4.5 组件 CMakeLists 文件 . . . 1781
4.4.6 组件配置 . . . 1783
4.4.7 预处理器定义 . . . 1783
4.4.8 组件依赖 . . . 1783
4.4.9 覆盖项目的部分设置 . . . 1787
4.4.10 仅配置组件 . . . 1789
4.4.11 CMake 调试 . . . 1789
4.4.12 组件 CMakeLists 示例 . . . 1789
4.4.13 自定义 sdkconfig 的默认值 . . . 1793
4.4.14 flash 参数 . . . 1793
4.4.15 构建 Bootloader . . . 1794
4.4.16 编写纯 CMake 组件 . . . 1794
4.4.17 组件中使用第三方 CMake 项目 . . . 1794
4.4.18 组件中使用预建库 . . . 1795
4.4.19 在自定义 CMake 项目中使用 ESP-IDF . . . 1795
4.4.20 ESP-IDF CMake 构建系统 API . . . 1796
4.4.21 文件通配 & 增量构建 . . . 1800
4.4.22 构建系统的元数据 . . . 1800
4.4.23 构建系统内部 . . . 1801
4.4.24 从 ESP-IDF GNU Make 构建系统迁移到 CMake 构建系统 . . . 1802
4.5 C Support . . . 1803
4.5.1 C Version . . . 1804
4.5.2 Unsupported C Features . . . 1804
4.6 C++ 支持 . . . 1804
4.6.1 esp-idf-cxx 组件 . . . 1804
4.6.2 C++ 语言标准 . . . 1804
4.6.3 多线程 . . . 1805
4.6.4 异常处理 . . . 1805
4.6.5 运行时类型信息 (RTTI) . . . 1805
4.6.6 在 C++ 中进行开发 . . . 1805
4.6.7 限制 . . . 1807
4.6.8 注意事项 . . . 1807
4.7 核心转储 . . . 1807
4.7.1 概述 . . . 1807
4.7.2 配置 . . . 1807
4.7.3 将核心转储保存到 flash . . . 1808
4.7.4 将核心转储保存到 UART . . . 1809
4.7.5 核心转储命令 . . . 1810
4.7.6 回溯中的 ROM 函数 . . . 1811
4.7.7 按需转储变量 . . . 1811
4.7.8 运行 idf.py coredump-info 和 idf.py coredump-debug . . . 1812
4.8 Current Consumption Measurement of Modules . . . 1814
4.8.1 Notes to Measurement . . . 1814
4.8.2 Hardware Connection . . . 1814
4.8.3 Measurement Steps . . . 1816
4.9 错误处理 . . . 1817
4.9.1 概述 . . . 1817
4.9.2 错误码 . . . 1818
4.9.3 错误码到错误消息 . . . 1818
4.9.4 ESP_ERROR_CHECK 宏 . . . 1818
4.9.5 ESP_ERROR_CHECK_WITHOUT_ABORT 宏 . . . 1819
4.9.6 ESP_RETURN_ON_ERROR 宏 . . . 1819
4.9.7 ESP_GOTO_ON_ERROR 宏 . . . 1819
4.9.8 ESP_RETURN_ON_FALSE 宏 . . . 1819
4.9.9 ESP_GOTO_ON_FALSE 宏 . . . 1819
4.9.10 CHECK 宏使用示例 . . . 1819
4.9.11 错误处理模式 . . . 1820

4.9.12 C++ 异常 . . . 1821
4.10 片外 RAM . . . 1821
4.10.1 简介 . . . 1821
4.10.2 硬件 . . . 1821
4.10.3 配置片外 RAM . . . 1821
4.10.4 片外 RAM 使用限制 . . . 1823
4.10.5 初始化失败 . . . 1823
4.10.6 加密 . . . 1823
4.11 严重错误 . . . 1824
4.11.1 概述 . . . 1824
4.11.2 紧急处理程序 . . . 1824
4.11.3 寄存器转储与回溯 . . . 1826
4.11.4 GDB Stub . . . 1827
4.11.5 RTC 看门狗超时 . . . 1828
4.11.6 Guru Meditation 错误 . . . 1828
4.11.7 其他严重错误 . . . 1829
4.12 硬件抽象 . . . 1832
4.12.1 架构 . . . 1832
4.12.2 LL 层（低级层） . . . 1833
4.12.3 HAL（硬件抽象层） . . . 1834
4.13 JTAG 调试 . . . 1835
4.13.1 引言 . . . 1835
4.13.2 工作原理 . . . 1836
4.13.3 选择 JTAG 适配器 . . . 1836
4.13.4 安装 OpenOCD . . . 1837
4.13.5 配置 ESP32-P4 目标板 . . . 1837
4.13.6 启动调试器 . . . 1840
4.13.7 调试范例 . . . 1840
4.13.8 从源码构建 OpenOCD . . . 1840
4.13.9 注意事项和补充内容 . . . 1845
4.13.10 相关文档 . . . 1849
4.14 链接器脚本生成机制 . . . 1874
4.14.1 概述 . . . 1874
4.14.2 快速上手 . . . 1874
4.14.3 链接器脚本生成机制内核 . . . 1877
4.15 lwIP . . . 1882
4.15.1 支持的 API . . . 1883
4.15.2 BSD 套接字 API . . . 1883
4.15.3 Netconn API . . . 1887
4.15.4 lwIP FreeRTOS 任务 . . . 1887
4.15.5 IPv6 支持 . . . 1888
4.15.6 ESP-lwIP 自定义修改 . . . 1888
4.15.7 性能优化 . . . 1890
4.16 存储器类型 . . . 1891
4.16.1 DRAM（数据 RAM） . . . 1891
4.16.2 IRAM（指令 RAM） . . . 1892
4.16.3 IROM（代码从 flash 中运行） . . . 1893
4.16.4 DROM（数据存储在 flash 中） . . . 1893
4.16.5 RTC FAST memory（RTC 快速存储器） . . . 1893
4.16.6 紧密耦合内存 (TCM) . . . 1893
4.16.7 具备 DMA 功能 . . . 1893
4.16.8 在堆栈中放置 DMA 缓冲区 . . . 1894
4.17 OpenThread . . . 1894
4.17.1 OpenThread 协议栈运行模式 . . . 1894
4.17.2 编写 OpenThread 应用程序 . . . 1895
4.17.3 OpenThread 边界路由器 . . . 1896
4.18 分区表 . . . 1896
4.18.1 概述 . . . 1896

4.18.2 内置分区表 . . . 1897
4.18.3 创建自定义分区表 . . . 1897
4.18.4 生成二进制分区表 . . . 1900
4.18.5 分区大小检查 . . . 1900
4.18.6 烧写分区表 . . . 1901
4.18.7 分区工具 (parttool.py) . . . 1901
4.19 性能 . . . 1902
4.19.1 如何优化性能 . . . 1902
4.19.2 指南 . . . 1902
4.20 Reproducible Builds . . . 1918
4.20.1 Introduction . . . 1918
4.20.2 Reasons for Non-Reproducible Builds . . . 1919
4.20.3 Enabling Reproducible Builds in ESP-IDF . . . 1919
4.20.4 How Reproducible Builds Are Achieved . . . 1919
4.20.5 Reproducible Builds and Debugging . . . 1919
4.20.6 Factors Which Still Affect Reproducible Builds . . . 1920
4.21 线程局部存储 . . . 1920
4.21.1 概述 . . . 1920
4.21.2 FreeRTOS 原生 API . . . 1920
4.21.3 Pthread API . . . 1920
4.21.4 C11 标准 . . . 1920
4.22 工具 . . . 1921
4.22.1 IDF 前端工具 - idf.py . . . 1921
4.22.2 IDF 监视器 . . . 1925
4.22.3 IDF Docker 镜像 . . . 1933
4.22.4 IDF Windows 安装程序 . . . 1936
4.22.5 IDF 组件管理器 . . . 1937
4.22.6 IDF clang-tidy . . . 1938
4.22.7 可下载的 ESP-IDF 工具 . . . 1939
4.23 ESP32-P4 中的单元测试 . . . 1952
4.23.1 添加常规测试用例 . . . 1952
4.23.2 添加多设备测试用例 . . . 1953
4.23.3 添加多阶段测试用例 . . . 1954
4.23.4 应用于不同芯片的单元测试 . . . 1954
4.23.5 编译单元测试程序 . . . 1955
4.23.6 运行单元测试 . . . 1955
4.23.7 带缓存补偿定时器的定时代码 . . . 1956
4.23.8 Mocks . . . 1957
4.24 在主机上运行 ESP-IDF 应用程序 . . . 1959
4.24.1 介绍 . . . 1959
4.24.2 使用模拟器的前提 . . . 1960
4.24.3 构建和运行 . . . 1960
4.24.4 组件 Linux/Mock 支持情况概述 . . . 1960
4.25 USB 串行/JTAG 控制器控制台 . . . 1961
4.25.1 硬件要求 . . . 1961
4.25.2 软件配置 . . . 1962
4.25.3 上传应用程序 . . . 1962
4.25.4 限制 . . . 1962
4.26 低功耗模式使用指南 . . . 1963
4.26.1 系统低功耗模式介绍 . . . 1963
4.27 Wi-Fi Expansion . . . 1969

5 迁移指南 **1971**
5.1 迁移到 ESP-IDF 5.x . . . 1971
5.1.1 从 4.4 迁移到 5.0 . . . 1971
5.1.2 从 5.0 迁移到 5.1 . . . 1999
5.1.3 从 5.1 迁移到 5.2 . . . 2002
5.1.4 从 5.2 迁移到 5.3 . . . 2004

6 安全指南 **2009**
6.1 概述 . . . 2009
6.1.1 安全 . . . 2009
6.2 功能 . . . 2012
6.2.1 flash 加密 . . . 2012
6.2.2 Secure Boot V2 . . . 2025
6.3 流程 . . . 2034
6.3.1 Host-Based Security Workflows . . . 2034

7 库与框架 **2045**
7.1 云框架 . . . 2045
7.1.1 ESP RainMaker . . . 2045
7.1.2 AWS IoT . . . 2045
7.1.3 Azure IoT . . . 2045
7.1.4 Google IoT Core . . . 2045
7.1.5 阿里云 IoT . . . 2045
7.1.6 Joylink IoT . . . 2045
7.1.7 腾讯 IoT . . . 2046
7.1.8 腾讯云 IoT . . . 2046
7.1.9 百度 IoT . . . 2046
7.2 其他库和开发框架 . . . 2046
7.2.1 ESP-ADF . . . 2046
7.2.2 ESP-CSI . . . 2046
7.2.3 ESP-DSP . . . 2046
7.2.4 ESP-WIFI-MESH . . . 2047
7.2.5 ESP-WHO . . . 2047
7.2.6 ESP RainMaker . . . 2047
7.2.7 ESP-IoT-Solution . . . 2047
7.2.8 ESP-Protocols . . . 2047
7.2.9 ESP-BSP . . . 2048
7.2.10 ESP-IDF-CXX . . . 2048

8 贡献指南 **2049**
8.1 如何贡献 . . . 2049
8.2 准备工作 . . . 2049
8.3 Pull Request 提交流程 . . . 2049
8.4 法律规范 . . . 2049
8.5 相关文档 . . . 2050
8.5.1 Espressif IoT Development Framework Style Guide . . . 2050
8.5.2 为 ESP-IDF 安装 pre-commit 钩子 . . . 2058
8.5.3 编写文档 . . . 2059
8.5.4 创建示例项目 . . . 2063
8.5.5 API 文档模板 . . . 2064
8.5.6 贡献者协议 . . . 2066
8.5.7 版权标头指南 . . . 2068
8.5.8 ESP-IDF pytest 指南 . . . 2069

9 ESP-IDF 版本简介 **2081**
9.1 发布版本 . . . 2081
9.2 我该选择哪个版本? . . . 2081
9.3 版本管理 . . . 2081
9.4 支持期限 . . . 2082
9.5 查看当前版本 . . . 2083
9.6 Git 工作流 . . . 2084
9.7 更新 ESP-IDF . . . 2084
9.7.1 更新至一个稳定发布版本 . . . 2084
9.7.2 更新至一个预发布版本 . . . 2085
9.7.3 更新至 master 分支 . . . 2085
9.7.4 更新至一个发布分支 . . . 2085

10 资源 **2087**
10.1 PlatformIO . . . 2087
10.1.1 什么是 PlatformIO? . . . 2087
10.1.2 安装 . . . 2087
10.1.3 配置 . . . 2088
10.1.4 教程 . . . 2088
10.1.5 项目示例 . . . 2088
10.1.6 更多内容 . . . 2088
10.2 CLion . . . 2088
10.2.1 CLion 是什么? . . . 2088
10.2.2 安装 . . . 2088
10.2.3 配置 . . . 2088
10.2.4 资源 . . . 2088
10.3 VisualGDB . . . 2088
10.3.1 VisualGDB 是什么? . . . 2089
10.3.2 安装 . . . 2089
10.3.3 配置 . . . 2089
10.3.4 资源 . . . 2089
10.4 有用的链接 . . . 2089

11 Copyrights and Licenses **2091**
11.1 Software Copyrights . . . 2091
11.1.1 Firmware Components . . . 2091
11.1.2 Documentation . . . 2092
11.2 ROM Source Code Copyrights . . . 2092
11.3 Xtensa libhal MIT License . . . 2093
11.4 TinyBasic Plus MIT License . . . 2093
11.5 TJpgDec License . . . 2093

12 关于本指南 **2095**

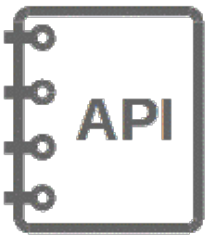
13 切换语言 **2097**

索引 **2099**

索引 **2099**

这里是乐鑫 IoT 开发框架 (esp-idf) 的文档中心。ESP-IDF 是 ESP32、ESP32-S、ESP32-C、ESP32-H 和 ESP32-P 系列芯片的官方开发框架。

本文档仅包含针对 ESP32-P4 芯片的 ESP-IDF 使用。

		
快速入门	API 参考	API 指南

Chapter 1

快速入门

本文档旨在指导用户搭建 ESP32-P4 硬件开发的软件环境，通过一个简单的示例展示如何使用 ESP-IDF (Espressif IoT Development Framework) 配置菜单，并编译、下载固件至 ESP32-P4 开发板等步骤。

备注： 这是 ESP-IDF 稳定版本 v5.3.2 的文档，还有其他版本的文档*ESP-IDF 版本简介* 供参考。

1.1 概述

ESP32-P4 SoC 芯片支持以下功能：

- 搭载 RISC-V 32 位双核处理器的高性能 MCU
- 强大的图像与语音处理能力
- 支持单精度 FPU 和 AI 扩展
- 外设丰富包括 MIPI、USB、SDIO、以太网等
- 内置安全硬件

ESP32-P4 采用 40 nm 工艺制成，具有最佳的功耗性能、射频性能、稳定性、通用性和可靠性，适用于各种应用场景和不同功耗需求。

乐鑫为用户提供完整的软、硬件资源，进行 ESP32-P4 硬件设备的开发。其中，乐鑫的软件开发环境 ESP-IDF 旨在协助用户快速开发物联网 (IoT) 应用，可满足用户对 Wi-Fi、蓝牙、低功耗等方面的要求。

1.2 准备工作

1.2.1 硬件：

- 一款 **ESP32-P4** 开发板
- **USB 数据线**（A 转 Micro-B）
- 电脑（Windows、Linux 或 macOS）

备注： 目前一些开发板使用的是 USB Type C 接口。请确保使用合适的数据线来连接开发板！

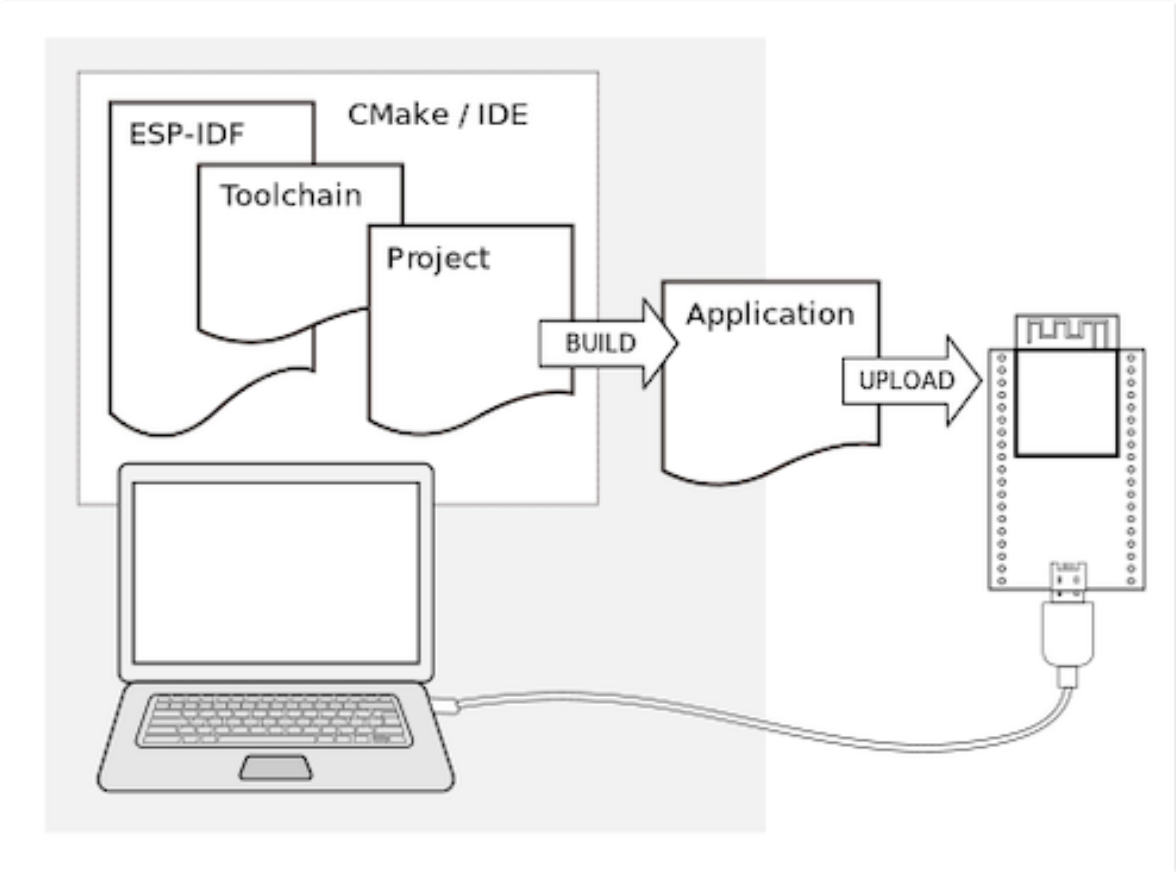
以下是 ESP32-P4 官方开发板，点击链接可了解更多硬件信息。

即将发布

1.2.2 软件：

如需在 **ESP32-P4** 上使用 ESP-IDF，请安装以下软件：

- 设置 **工具链**，用于编译 ESP32-P4 代码；
- **编译构建工具**——CMake 和 Ninja 编译构建工具，用于编译 ESP32-P4 **应用程序**；
- 获取 **ESP-IDF** 软件开发框架。该框架已经基本包含 ESP32-P4 使用的 API（软件库和源代码）和运行 **工具链**的脚本；



1.3 安装

为安装所需软件，乐鑫提供了以下方法，可根据需要选择其中之一。

1.3.1 IDE

备注： 建议通过自己喜欢的集成开发环境 (IDE) 安装 ESP-IDF。

- Eclipse Plugin
- VSCode Extension

1.3.2 手动安装

请根据操作系统，选择对应的手动安装流程。

Windows 平台工具链的标准设置

概述 ESP-IDF 需要安装一些必备工具，才能围绕 ESP32-P4 构建固件，包括 Python、Git、交叉编译器、CMake 和 Ninja 编译工具等。

本入门指南介绍了如何通过 **命令提示符**进行有关操作。不过，安装 ESP-IDF 后，还可以使用 Eclipse Plugin 或其他支持 CMake 的图形化工具 IDE。

备注： 限定条件：

- 请注意 ESP-IDF 和 ESP-IDF 工具的安装路径不能超过 90 个字符，安装路径过长可能会导致构建失败。
- Python 或 ESP-IDF 的安装路径中一定不能包含空格或括号。
- 除非操作系统配置为支持 Unicode UTF-8，否则 Python 或 ESP-IDF 的安装路径中也不能包括特殊字符（非 ASCII 码字符）

系统管理员可以通过如下方式将操作系统配置为支持 Unicode UTF-8：`Control Panel` > 更改 `date、time`、或 `number` 格式 > `Administrative tab` > 更改 `system locale` > 勾选选项 `Beta: Use Unicode UTF-8 for worldwide language support` > `Ok` > 重启电脑。

ESP-IDF 工具安装器 安装 ESP-IDF 必备工具最简易的方式是下载一个 ESP-IDF 工具安装器。

Windows Installer Download

在线安装与离线安装的区别 在线安装程序非常小，可以安装 ESP-IDF 的所有版本。在安装过程中，安装程序只下载必要的依赖文件，包括 Git For Windows 安装器。在线安装程序会将下载的文件存储在缓存目录 `%userprofile%/espressif` 中。

离线安装程序不需要任何网络连接。安装程序中包含了所有需要的依赖文件，包括 Git For Windows 安装器。

安装内容 安装程序会安装以下组件：

- 内置的 Python
- 交叉编译器
- OpenOCD
- CMake 和 Ninja 编译工具
- ESP-IDF

安装程序允许将程序下载到现有的 ESP-IDF 目录。推荐将 ESP-IDF 下载到 `%userprofile%\Desktop\esp-idf` 目录下，其中 `%userprofile%` 代表家目录。

启动 ESP-IDF 环境 安装结束时，如果勾选了 `Run ESP-IDF PowerShell Environment` 或 `Run ESP-IDF Command Prompt (cmd.exe)`，安装程序会在选定的提示符窗口启动 ESP-IDF。

`Run ESP-IDF PowerShell Environment`:

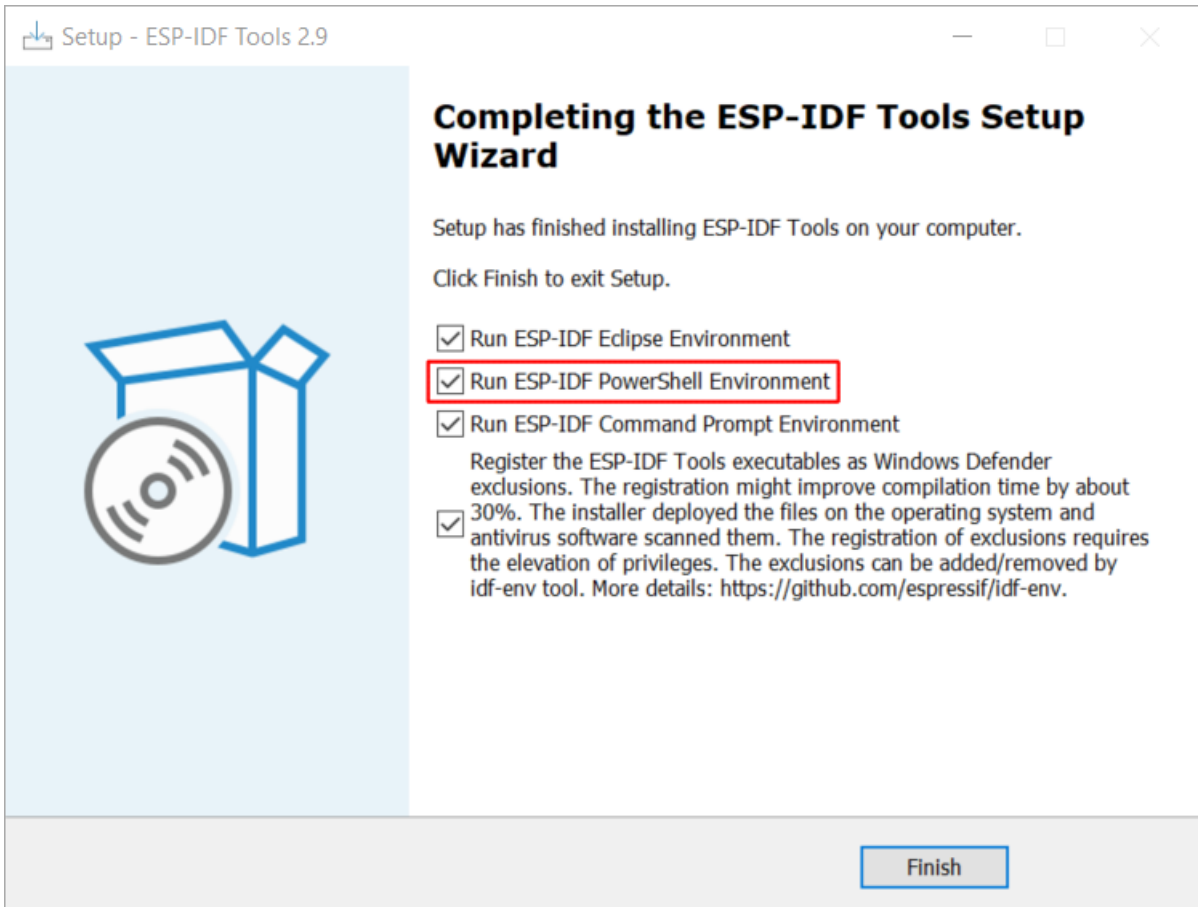


图 1: 完成 ESP-IDF 工具安装向导时运行 Run ESP-IDF PowerShell Environment

`Run ESP-IDF Command Prompt (cmd.exe)`:

使用命令提示符 在后续步骤中，将介绍如何使用 Windows 的命令提示符进行操作。

ESP-IDF 工具安装器可在“开始”菜单中，创建一个打开 ESP-IDF 命令提示符窗口的快捷方式。本快捷方式可以打开 Windows 命令提示符（即 cmd.exe），并运行 `export.bat` 脚本以设置各环境变量（比如 `PATH`，`IDF_PATH` 等）。此外，还可以通过 Windows 命令提示符使用各种已经安装的工具。

注意，本快捷方式仅适用 ESP-IDF 工具安装器中指定的 ESP-IDF 路径。如果电脑上存在多个 ESP-IDF 路径（比如需要不同版本的 ESP-IDF），有以下两种解决方法：

1. 为 ESP-IDF 工具安装器创建的快捷方式创建一个副本，并将新快捷方式的 ESP-IDF 工作路径指定为希望使用的 ESP-IDF 路径。
2. 或者，可以运行 `cmd.exe`，并切换至希望使用的 ESP-IDF 目录，然后运行 `export.bat`。注意，这种方法要求 `PATH` 中存在 Python 和 Git。如果在使用时遇到有关“找不到 Python 或 Git”的错误信息，请使用第一种方法。

开始使用 ESP-IDF 现在你已经具备了使用 ESP-IDF 的所有条件，接下来将介绍如何开始第一个工程。

本指南将介绍如何初步上手 ESP-IDF，包括如何使用 ESP32-P4 创建第一个工程，并构建、烧录和监控设备输出。

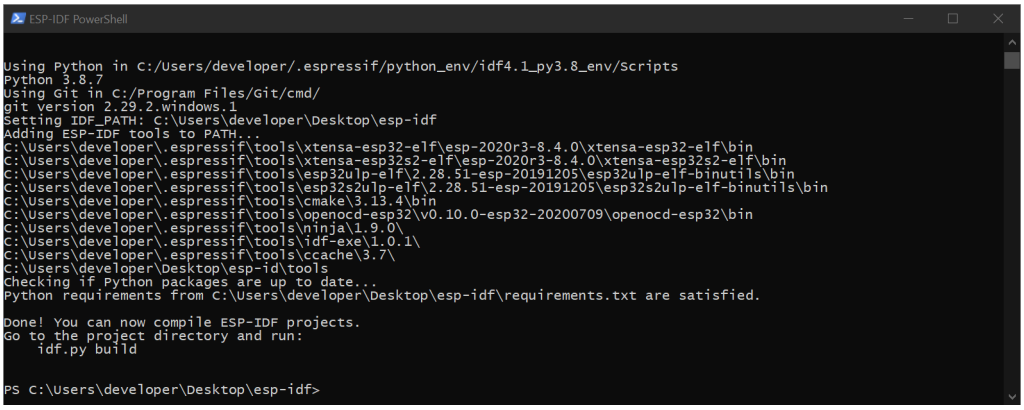


图 2: ESP-IDF PowerShell

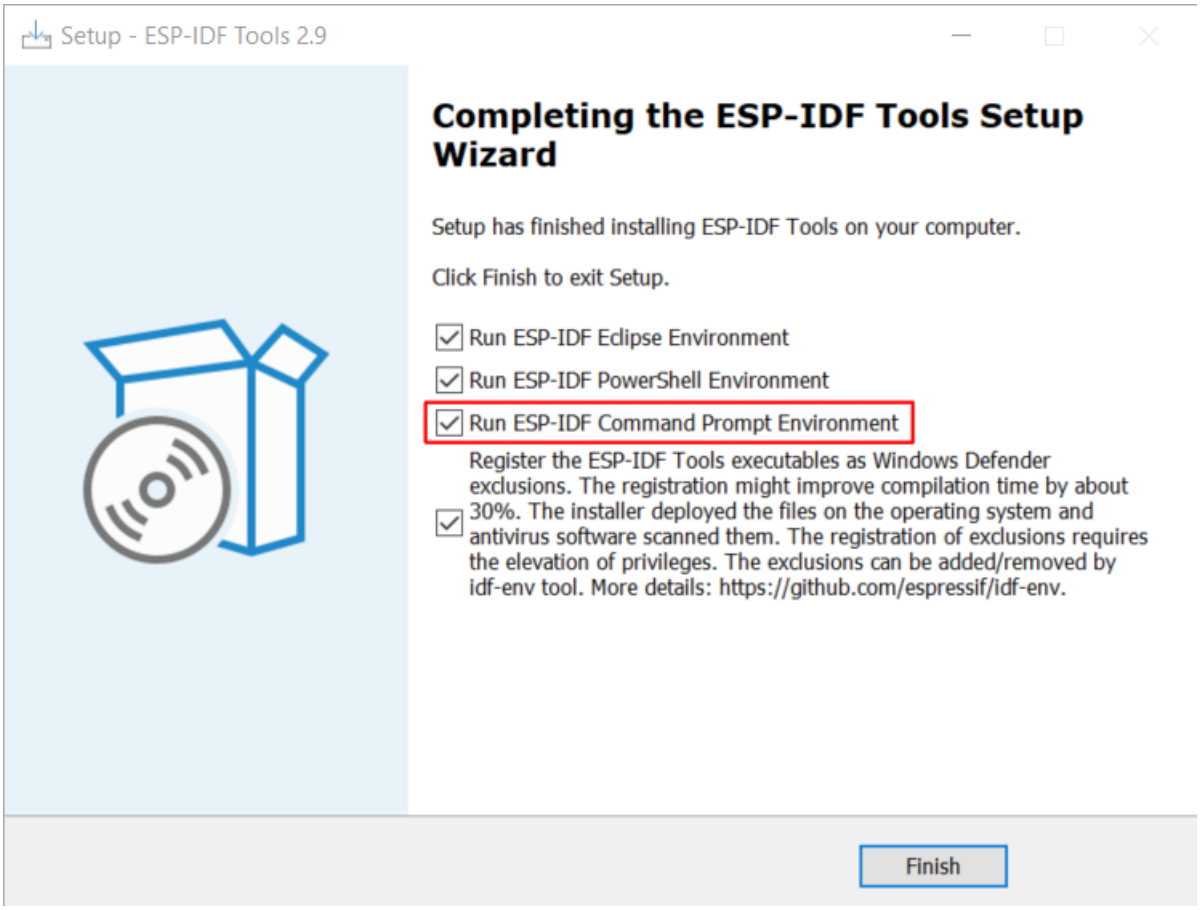


图 3: 完成 ESP-IDF 工具安装向导时运行 Run ESP-IDF Command Prompt (cmd.exe)

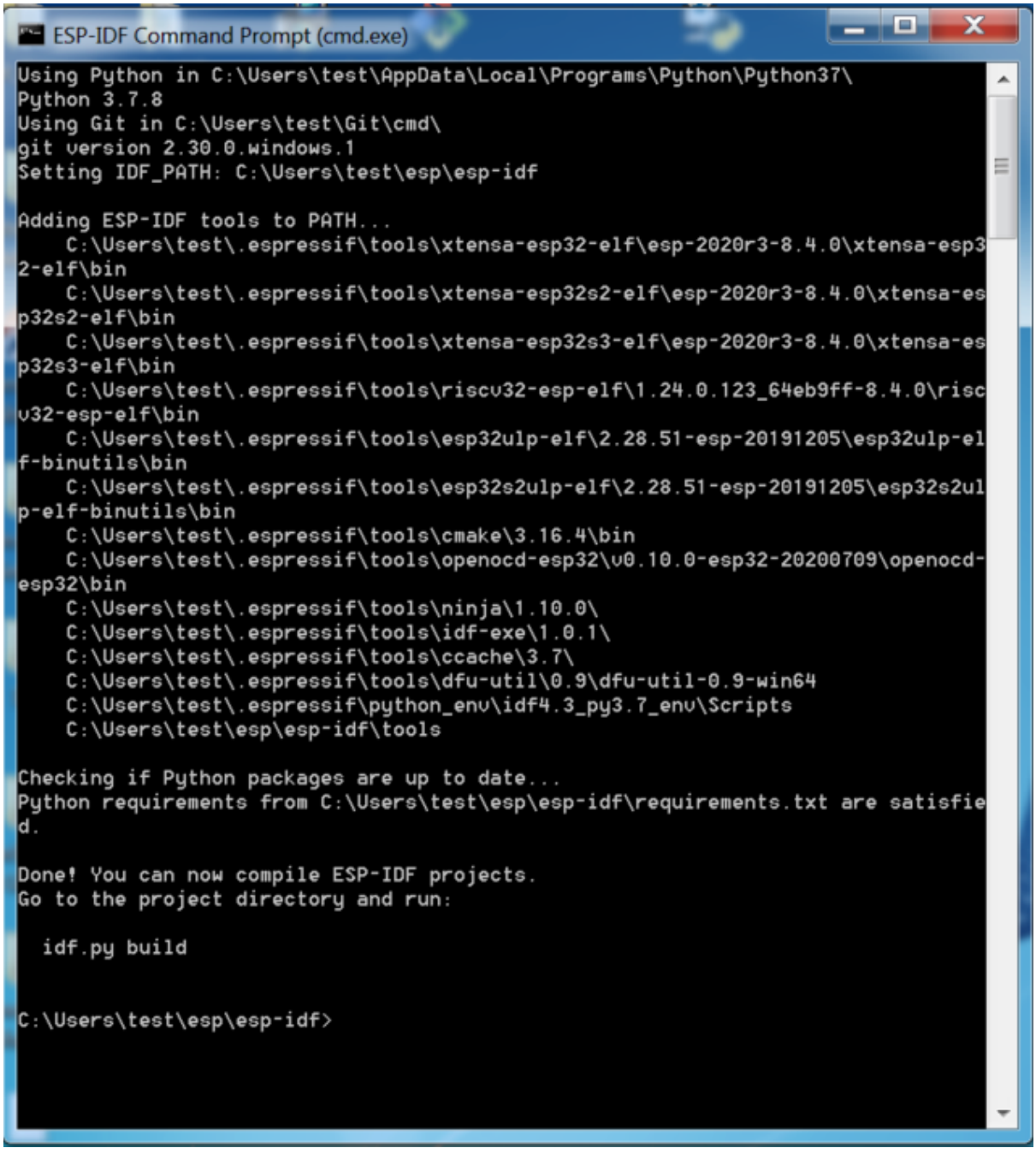


图 4: ESP-IDF 命令提示符窗口

备注： 如果还未安装 ESP-IDF，请参照安装 中的步骤，获取使用本指南所需的所有软件。

开始创建工程 现在，可以准备开发 ESP32-P4 应用程序了。可以从 ESP-IDF 中 examples 目录下的 get-started/hello_world 工程开始。

重要： ESP-IDF 编译系统不支持 ESP-IDF 路径或其工程路径中带有空格。

将 get-started/hello_world 工程复制至本地的 ~/esp 目录下：

```
cd %userprofile%\esp
xcopy /e /i %IDF_PATH%\examples\get-started\hello_world hello_world
```

备注： ESP-IDF 的 examples 目录下有一系列示例工程，可以按照上述方法复制并运行其中的任何示例，也可以直接编译示例，无需进行复制。

连接设备 现在，请将 ESP32-P4 开发板连接到 PC，并查看开发板使用的串口。

在 Windows 操作系统中，串口名称通常以 COM 开头。

有关如何查看串口名称的详细信息，请见*与 ESP32-P4 创建串口连接*。

备注： 请记住串口名，以便后续使用。

配置工程 请进入 hello_world 目录，设置 ESP32-P4 为目标芯片，然后运行工程配置工具 menuconfig。

Windows

```
cd %userprofile%\esp\hello_world
idf.py set-target esp32p4
idf.py menuconfig
```

打开一个新工程后，应首先使用 idf.py set-target esp32p4 设置“目标”芯片。注意，此操作将清除并初始化项目之前的编译和配置（如有）。也可以直接将“目标”配置为环境变量（此时可跳过该步骤）。更多信息，请见*选择目标芯片：set-target*。

正确操作上述步骤后，系统将显示以下菜单：

可以通过此菜单设置项目的具体变量，包括 Wi-Fi 网络名称、密码和处理器速度等。hello_world 示例项目会以默认配置运行，因此在这一项目中，可以跳过使用 menuconfig 进行项目配置这一步骤。

备注： 终端窗口中显示出的菜单颜色可能会与上图不同。可以通过选项 --style 来改变外观。请运行 idf.py menuconfig --help 命令，获取更多信息。

编译工程 请使用以下命令，编译烧录工程：

```
idf.py build
```

运行以上命令可以编译应用程序和所有 ESP-IDF 组件，接着生成引导加载程序、分区表和应用程序二进制文件。

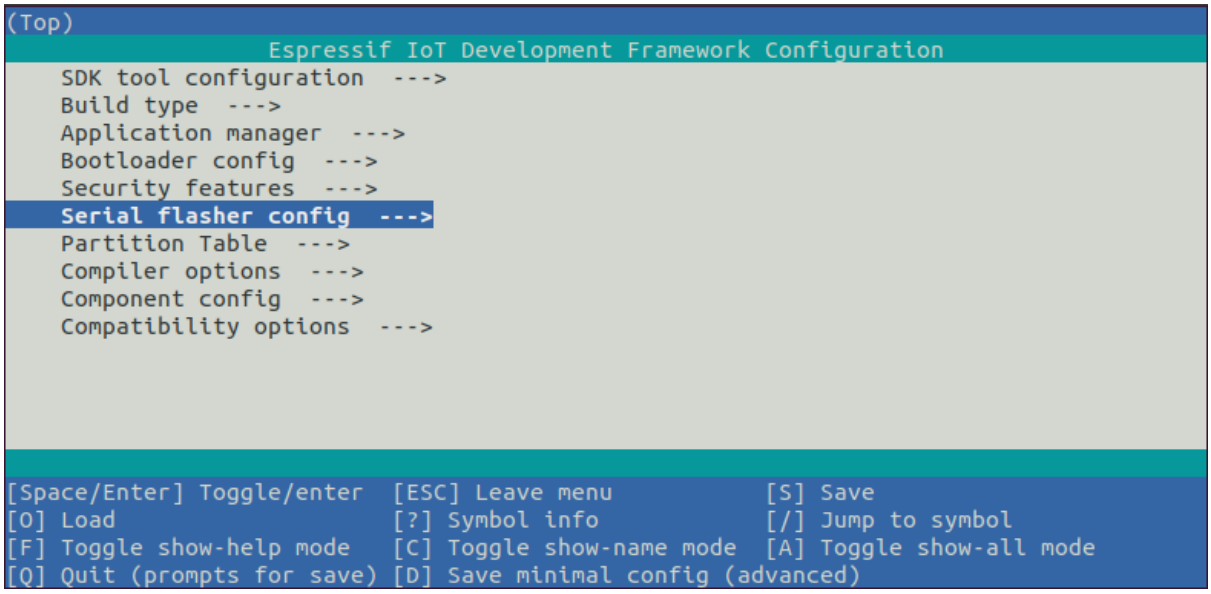


图 5: 工程配置—主窗口

```
$ idf.py build
Running cmake in directory /path/to/hello_world/build
Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
Warn about uninitialized values.
-- Found Git: /usr/bin/git (found version "2.17.0")
-- Building empty aws_iot component due to configuration
-- Component names: ...
-- Component paths: ...

... (more lines of build system output)

[527/527] Generating hello_world.bin
esptool.py v2.3.1

Project build complete. To flash, run this command:
../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600 write_flash -
↪-flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/hello_world.
↪bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/partition_table/
↪partition-table.bin
or run 'idf.py -p PORT flash'
```

如果一切正常，编译完成后将生成.bin 文件。

烧录到设备 请运行以下命令，将刚刚生成的二进制文件烧录至 ESP32-P4 开发板：

```
idf.py -p PORT flash
```

请将 PORT 替换为 ESP32-P4 开发板的串口名称。如果 `PORT` 未经定义，*idf.py* 将尝试使用可用的串口自动连接。

更多有关 idf.py 参数的详情，请见*idf.py*。

备注： 勾选 `flash` 选项将自动编译并烧录工程，因此无需再运行 `idf.py build`。

若在烧录过程中遇到问题，请参考下文中的“其他提示”。也可以前往烧录故障排除 或与 *ESP32-P4* 创建串口连接 获取更多详细信息。

常规操作 在烧录过程中，会看到类似如下的输出日志：

```
...
esptool.py v4.8.dev3
Serial port /dev/cu.SLAB_USBtoUART
Connecting....
Chip is ESP32-P4 (revision v0.0)
Features: High-Performance MCU
Crystal is 40MHz
MAC: 00:00:00:00:00:00
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00002000 to 0x00007fff...
Flash will be erased from 0x00010000 to 0x0003bfff...
Flash will be erased from 0x00008000 to 0x00008fff...
SHA digest in image updated
Compressed 20960 bytes to 12653...
Writing at 0x00002000... (100 %)
Wrote 20960 bytes (12653 compressed) at 0x00002000 in 0.6 seconds (effective 277.1␣
↪kbit/s)...
Hash of data verified.
Compressed 179584 bytes to 94528...
Writing at 0x00035efb... (100 %)
Wrote 179584 bytes (94528 compressed) at 0x00010000 in 2.6 seconds (effective 549.
↪9 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (effective 420.7␣
↪kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
Done
```

如果一切顺利，烧录完成后，开发板将会复位，应用程序"hello_world" 开始运行。

如果希望使用 Eclipse 或是 VS Code IDE，而非 `idf.py`，请参考 Eclipse Plugin，以及 VSCode Extension。

监视输出 可以使用 `idf.py -p PORT monitor` 命令，监视 "hello_world" 工程的运行情况。注意，不要忘记将 PORT 替换为自己的串口名称。

运行该命令后，*IDF 监视器* 应用程序将启动。

```
$ idf.py -p <PORT> monitor
Running idf_monitor in directory [...]/esp/hello_world/build
Executing "python [...]/esp-idf/tools/idf_monitor.py -b 115200 [...]/esp/hello_
↪world/build/hello_world.elf"...
--- idf_monitor on <PORT> 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57
...
```

此时，就可以在启动日志和诊断日志之后，看到打印的 "Hello world!" 了。

```
    ...
    Hello world!
    Restarting in 10 seconds...
    This is esp32p4 chip with 2 CPU core(s), silicon revision v0.0, 2 MB external␣
↪flash
Minimum free heap size: 618848 bytes
    Restarting in 9 seconds...
    Restarting in 8 seconds...
    Restarting in 7 seconds...
```

使用快捷键 `Ctrl+]`，可退出 ESP-IDF 监视器。

备注： 也可以运行以下命令，一次性执行构建、烧录和监视过程：

```
idf.py -p PORT flash monitor
```

此外，

- 请前往*IDF 监视器*，了解更多使用 ESP-IDF 监视器的快捷键和其他详情。
- 请前往*idf.py*，查看更多 `idf.py` 命令和选项。

恭喜完成 ESP32-P4 的入门学习！

现在，可以尝试一些其他 examples，或者直接开发自己的应用程序。

重要： 一些示例程序不支持 ESP32-P4，因为 ESP32-P4 中不包含所需的硬件。

在编译示例程序前请查看 README 文件中 `Supported Targets` 表格。如果表格中包含 ESP32-P4，或者不存在这个表格，那么即表示 ESP32-P4 支持这个示例程序。

其他提示

权限问题 使用某些 Linux 版本向 ESP32-P4 烧录固件时，可能会出现类似 `Could not open port <PORT>: Permission denied: '<PORT>'` 错误消息。此时可以在 Linux 将用户添加至*dialout 组*或*uucp 组* 来解决此类问题。

兼容的 Python 版本 ESP-IDF 支持 Python 3.8 及以上版本，建议升级操作系统到最新版本从而更新 Python。也可选择从 sources 安装最新版 Python，或使用 Python 管理系统如 pyenv 对版本进行升级管理。

擦除 flash ESP-IDF 支持擦除 flash。请运行以下命令，擦除整个 flash：

```
idf.py -p PORT erase-flash
```

若存在需要擦除的 OTA 数据，请运行以下命令：

```
idf.py -p PORT erase-otadata
```

擦除 flash 需要一段时间，在擦除过程中，请勿断开设备连接。

相关文档 想要自定义安装流程的高阶用户可参照：

- *在 Windows 环境下更新 ESP-IDF 工具*
- *与 ESP32-P4 创建串口连接*
- Eclipse Plugin
- VSCode Extension
- *IDF 监视器*

在 Windows 环境下更新 ESP-IDF 工具

使用脚本安装 ESP-IDF 工具 请从 Windows “命令提示符” 窗口，切换至 ESP-IDF 的安装目录。然后运行:

```
install.bat
```

对于 Powershell，请切换至 ESP-IDF 的安装目录。然后运行:

```
install.ps1
```

该命令可下载并安装 ESP-IDF 所需的工具。如已经安装了某个版本的工具，则该命令将无效。该工具的下载安装位置由 ESP-IDF 工具安装器的设置决定，默认情况下为：`C:\Users\username\.espressif`。

使用 “导出脚本” 将 ESP-IDF 工具添加至 PATH 环境变量 ESP-IDF 工具安装器将在 “开始菜单” 为 “ESP-IDF 命令提示符” 创建快捷方式。点击该快捷方式可打开 Windows 命令提示符窗口，可在该窗口使用所有已安装的工具。

有些情况下，正在使用的命令提示符窗口并不是通过快捷方式打开的，此时如果想要在该窗口使用 ESP-IDF，可以根据下方步骤将 ESP-IDF 工具添加至 PATH 环境变量。

首先，请打开需要使用 ESP-IDF 的命令提示符窗口，切换至安装 ESP-IDF 的目录，然后执行 `export.bat`，具体命令如下:

```
cd %userprofile%\esp\esp-idf
export.bat
```

对于 Powershell 用户，请同样切换至安装 ESP-IDF 的目录，然后执行 `export.ps1`，具体命令如下:

```
cd ~/esp/esp-idf
export.ps1
```

运行完成后，就可以通过命令提示符使用 ESP-IDF 工具了。

与 ESP32-P4 创建串口连接

可以使用 USB 至 UART 桥或 ESP32-P4 支持的 USB 外设，与 ESP32-P4 创建串口连接。

部分开发板中已经安装有 USB 至 UART 桥。如未安装，可使用外部桥。

支持的 USB 外设 ESP32-P4 支持 USB 外设。无需 USB 至 UART 桥，便可直接烧录设备。

部分支持 USB 外设的开发板同时也包含 USB 至 UART 桥。

安装有 USB 至 UART 桥的开发板 在安装有 USB 至 UART 桥的开发板中，PC 和桥之间通过 USB 连接，桥和 ESP32-P4 之间通过 UART 连接。

外部 USB 至 UART 桥 部分开发板使用外部 USB 至 UART 桥。这种情况通常出现在需要控制空间和成本的产品中，例如一些小型开发板或成品。

使用 USB 进行烧录 ESP32-P4 支持 USB 外设，无需外部 USB 至 UART 桥，即可烧录二进制文件。

ESP32-P4 上的 USB 使用 **GPIO24/26** 作为 **D+**，**GPIO24/26** 作为 **D-**。

使用 UART 进行烧录 本节描述如何使用 USB 至 UART 桥在 ESP32-P4 和 PC 之间建立串行连接。板上桥与外部桥均适用。

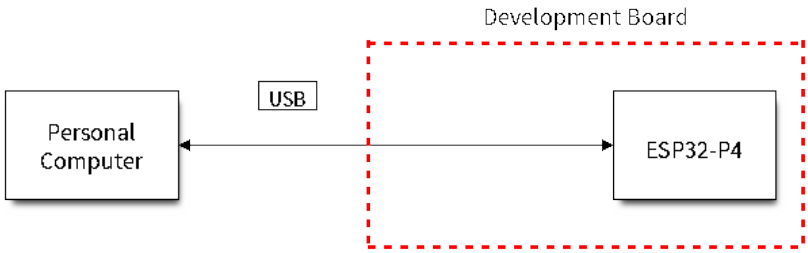


图 6: 支持 USB 的 SoC

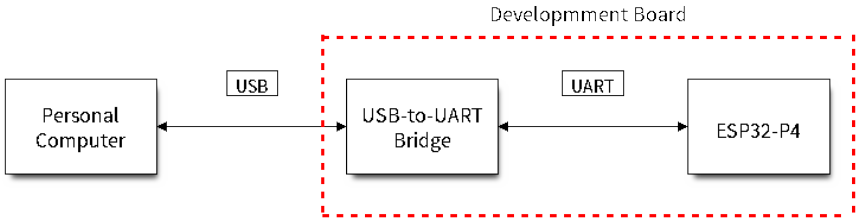


图 7: 安装有 USB 至 UART 桥的开发板

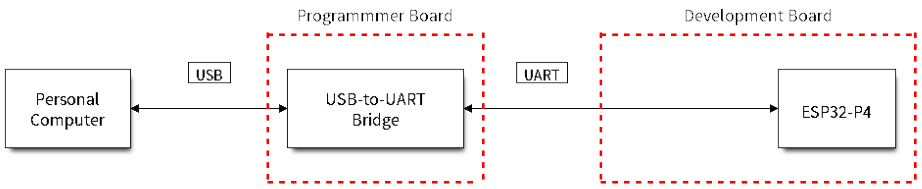


图 8: 外部 USB 至 UART 桥

连接 ESP32-P4 和 PC 用 USB 线将 ESP32-P4 开发板连接到 PC。如果设备驱动程序没有自动安装，请先确认 ESP32-P4 开发板上的 USB 至 UART 桥（或外部转 UART 适配器）型号，然后在网上搜索驱动程序，并进行手动安装。

以下是乐鑫 ESP32-P4 开发板驱动程序的链接：

- CP210x: CP210x USB 至 UART 桥 VCP 驱动程序
- FTDI: FTDI 虚拟 COM 端口驱动程序

以上驱动仅供参考，请查看开发板用户指南，了解开发板具体使用的 USB 至 UART 桥芯片。一般情况下，当 ESP32-P4 开发板与 PC 连接时，对应驱动程序应该已经被打包在操作系统中，并已经自动安装。

对于使用 USB 至 UART 桥下载的设备，可以运行以下命令，包括定义波特率的可选参数。

```
idf.py -p PORT [-b BAUD] flash
```

将 `PORT` 替换为 ESP32-P4 开发板的串口名称。`-b` 为可选参数，默认的波特率为 `460800`。如需改变烧录器的波特率，请用需要的波特率代替 `BAUD`。

要查看串口名称，Windows 用户请参考*check-port-on-windows*，Linux 与 macOS 用户请参考*check-port-on-linux-and-macos*。

例如，Windows 平台上串口名称为 `COM3`，所需波特率为 `115200`，可使用如下命令烧录开发板：

```
idf.py -p COM3 -b 115200 flash
```

Linux 平台上串口名称为 `/dev/ttyUSB0`，所需波特率为 `115200`，可使用如下命令烧录开发板：

```
idf.py -p /dev/ttyUSB0 -b 115200 flash
```

macOS 平台上串口名称为 `/dev/cu.usbserial-1401`，所需波特率为 `115200`，可使用如下命令烧录开发板：

```
idf.py -p /dev/cu.usbserial-1401 -b 115200 flash
```

备注： 如果设备不支持自动下载模式，则需要手动进入下载模式。请按住 `BOOT` 按钮，同时按一下 `RESET` 按钮。之后，松开 `BOOT` 按钮。

在 Windows 上查看端口 检查 Windows 设备管理器中的 COM 端口列表。断开 ESP32-P4 与 PC 的连接，然后重新连接，查看哪个端口从列表中消失后又再次出现。

以下为 ESP32 DevKitC 和 ESP32 WROVER KIT 串口：

在 Linux 和 macOS 上查看端口 查看 ESP32-P4 开发板（或外部转串口适配器）的串口设备名称，请将以下命令运行两次。首先，断开开发板或适配器，首次运行以下命令；然后，连接开发板或适配器，再次运行以下命令。其中，第二次运行命令后出现的端口即是 ESP32-P4 对应的串口：

Linux:

```
ls /dev/tty*
```

macOS:

```
ls /dev/cu.*
```

备注： 对于 macOS 用户：若没有看到串口，请检查是否安装 USB/串口驱动程序。具体应使用的驱动程序，见章节*连接 ESP32-P4 和 PC*。对于 macOS High Sierra (10.13) 的用户，你可能还需要手动允许驱动程序的加载，具体可打开 *系统偏好设置* -> *安全和隐私* -> *通用*，检查是否有信息显示：“来自开发人员的系统软件...”，其中开发人员的名称为 Silicon Labs 或 FTDI。

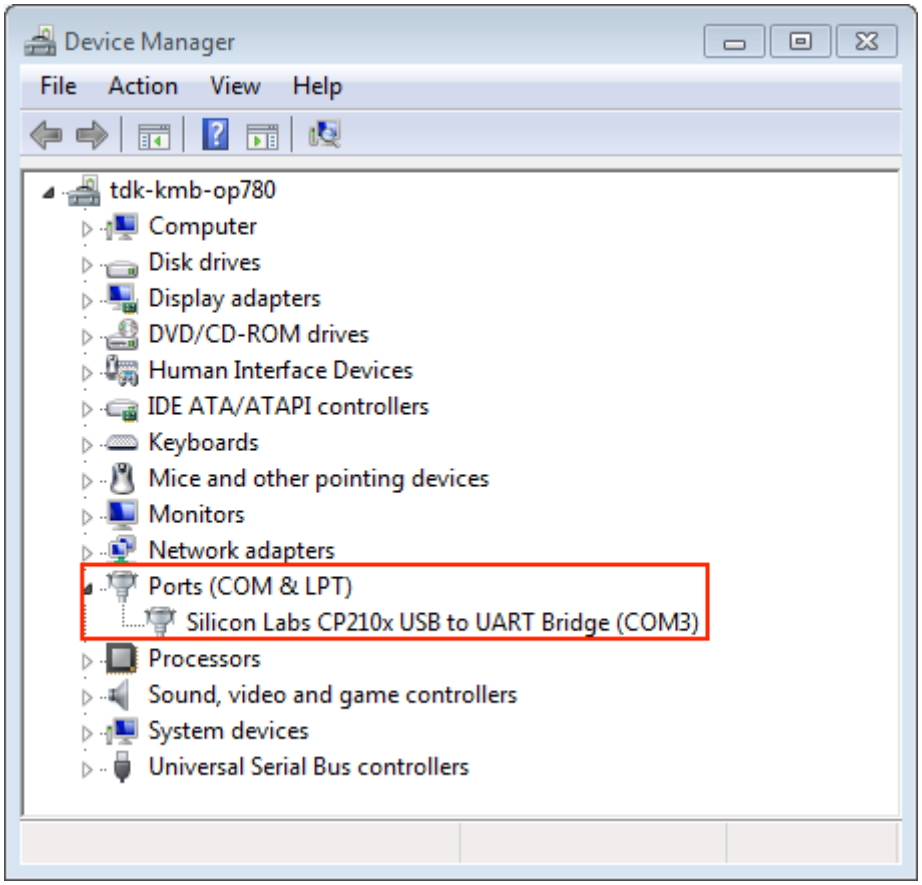


图 9: 设备管理器中 ESP32-DevKitC 的 USB 至 UART 桥

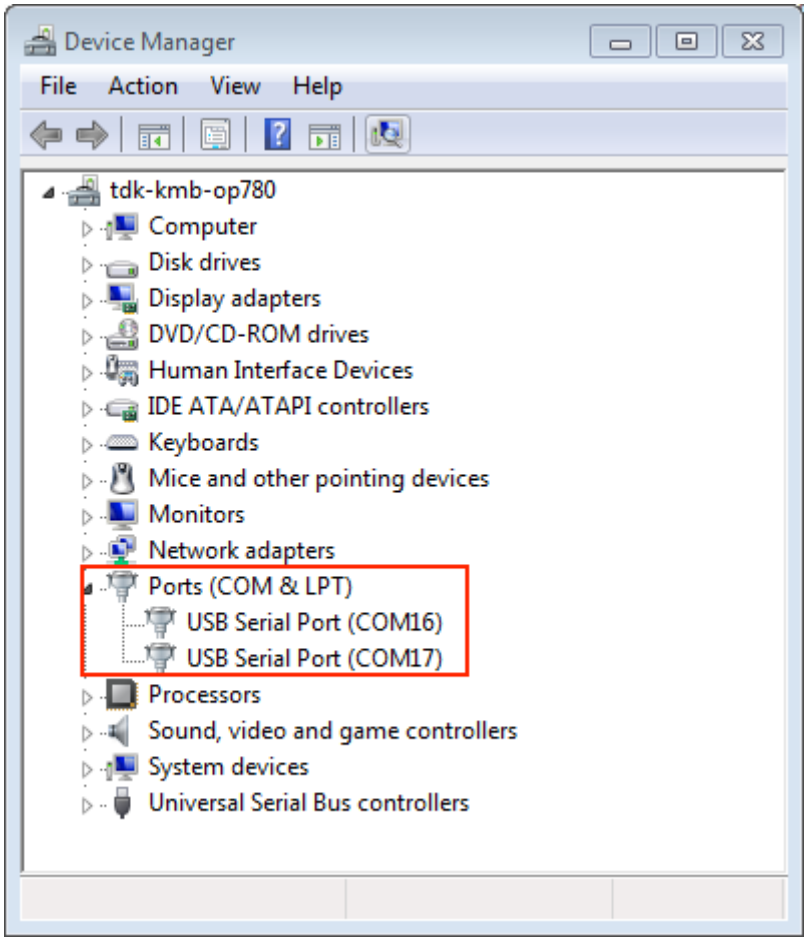


图 10: Windows 设备管理器中 ESP-WROVER-KIT 的两个 USB 串行端口

在 Linux 中添加用户到 dialout 或 uucp 组 当前登录用户应当可以通过 USB 对串口进行读写操作。在多数 Linux 版本中，都可以通过以下命令，将用户添加到 dialout 组，从而获许读写权限:

```
sudo usermod -a -G dialout $USER
```

在 Arch Linux 中，需要通过以下命令将用户添加到 uucp 组中:

```
sudo usermod -a -G uucp $USER
```

请重新登录，确保串口读写权限生效。

确认串口连接 现在，请使用串口终端程序，查看重置 ESP32-P4 后终端上是否有输出，从而验证串口连接是否可用。

ESP32-P4 的控制台波特率默认为 115200。

Windows 和 Linux 操作系统 在本示例中，我们将使用 PuTTY SSH Client，PuTTY SSH Client 既可用于 Windows 也可用于 Linux。也可以使用其他串口程序并设置如下的通信参数。

运行终端，配置在上述步骤中确认的串口：波特率 = 115200（如有需要，请更改为使用芯片的默认波特率），数据位 = 8，停止位 = 1，奇偶校验 = N。以下截屏分别展示了如何在 Windows 和 Linux 中配置串口和上述通信参数（如 115200-8-1-N）。注意，这里一定要选择在上述步骤中确认的串口进行配置。

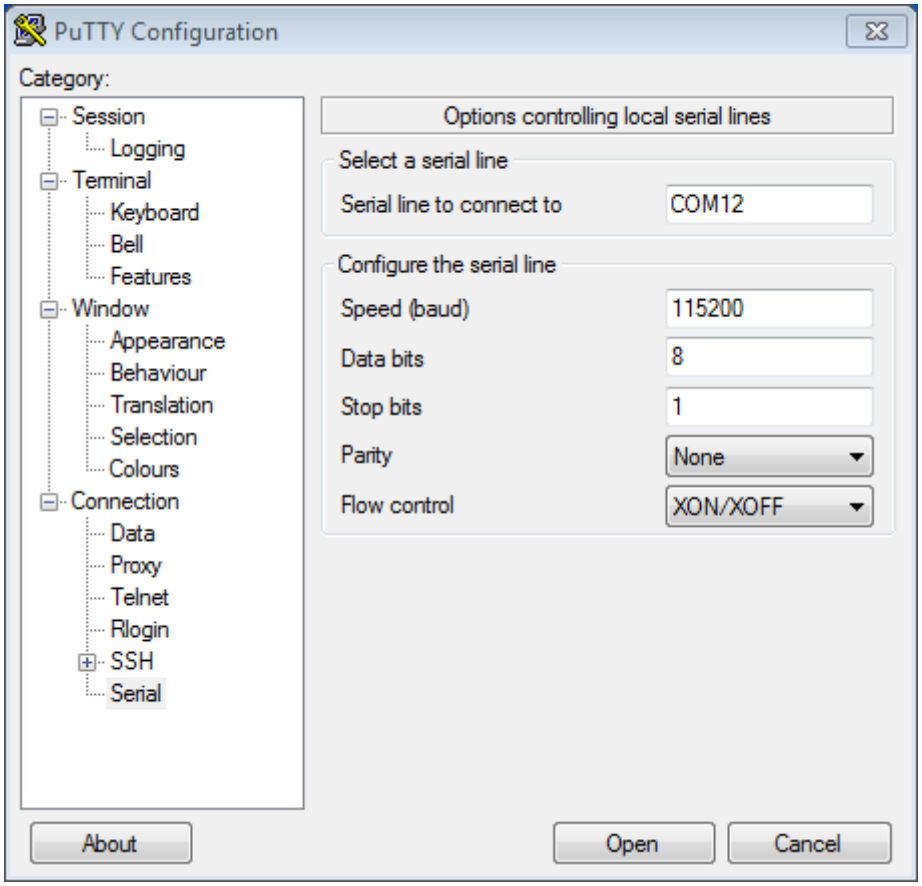


图 11: 在 Windows 操作系统中使用 PuTTY 设置串口通信参数

然后，请检查 ESP32-P4 是否有打印日志。如有，请在终端打开串口进行查看。这里的日志内容取决于加载到 ESP32-P4 的应用程序，请参考输出示例。如果没有看到输出日志，请尝试重启开发板。

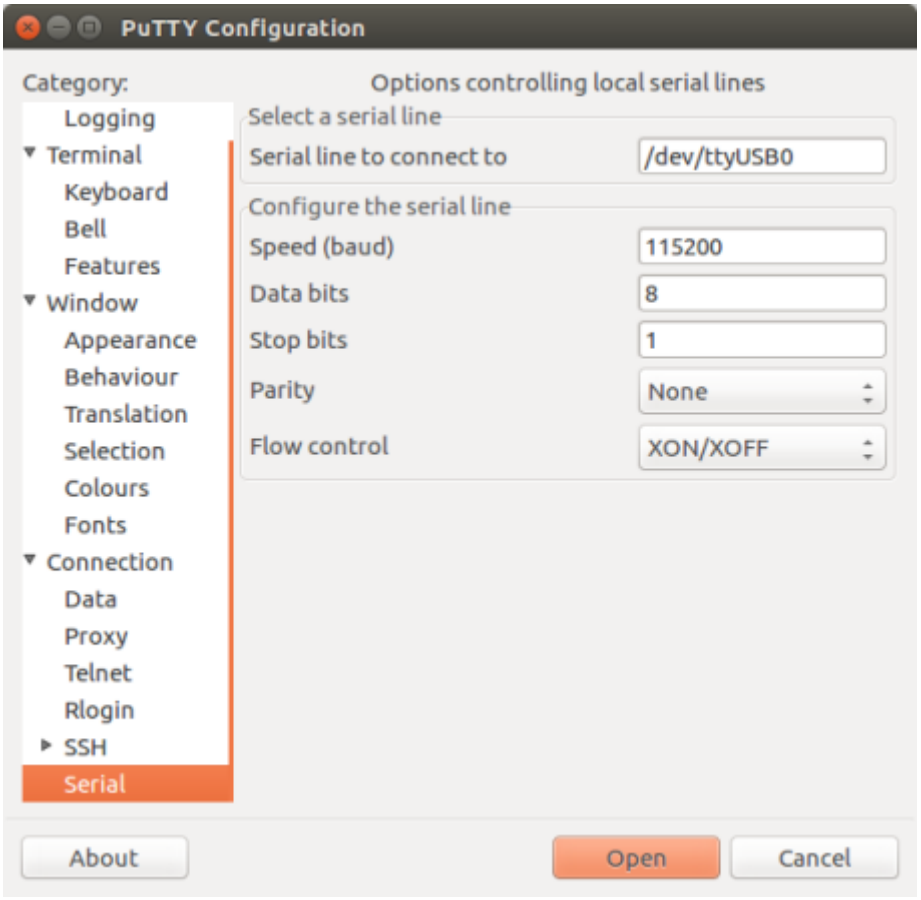


图 12: 在 Linux 操作系统中使用 PuTTY 设置串口通信参数

备注： 请在验证完串口通信正常后，关闭串口终端。如果终端一直保持打开的状态，之后上传固件时将无法访问串口。

备注： 如果没有日志输出，请检查以下原因：

- ESP32-P4 的供电是否正常
- 启动终端程序后，是否重置开发板
- 使用*在 Windows 上查看端口* 与*在 Linux 和 macOS 上查看端口* 中描述的方法，检查所选串口是否正确
- 其他程序是否正在使用该串口
- 对于*Windows 和 Linux 操作系统* 中描述的串口终端程序，其选择的端口是否正确
- 串口终端程序中的串口设置是否适用于该应用程序
- 开发板上选择的 USB 连接器（UART）是否正确
- 应用程序是否会输出日志
- 是否禁用了日志输出（使用 hello world 示例 进行测试）

macOS 操作系统 macOS 提供了 **屏幕**命令，因此无需安装串口终端程序。

- 参考*在 Linux 和 macOS 上查看端口*，运行以下命令：

```
ls /dev/cu.*
```

- 会看到类似如下输出：

```
/dev/cu.Bluetooth-Incoming-Port /dev/cu.SLAB_USBtoUART       /dev/cu.SLAB_
↪USBtoUART7
```

- 根据连接到电脑上的开发板类型和数量，输出结果会有所不同。请选择开发板的设备名称，并运行以下命令（如有需要，请将“115200”更改为使用芯片的默认波特率）：

```
screen /dev/cu.device_name 115200
```

将 `device_name` 替换为运行 `ls /dev/cu.*` 后出现的设备串口号。

- **屏幕**显示的日志即为所需内容。日志内容取决于加载到 ESP32-P4 的应用程序，请参考输出示例。请使用 `Ctrl-A + K` 键退出当前 **屏幕**会话。

备注： 请在验证完串口通信正常后，关闭 **当前屏幕会话**。如果直接关闭终端窗口而没有关闭 **屏幕**，之后上传固件时将无法访问串口。

输出示例 以下是一个日志示例。如果没看到任何输出，请尝试重置开发板。

```
ets Jun  8 2016 00:22:57

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57

rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0008,len:8
load:0x3fff0010,len:3464
load:0x40078000,len:7828
load:0x40080000,len:252
entry 0x40080034
```

(下页继续)

(续上页)

```
I (44) boot: ESP-IDF v2.0-rc1-401-gf9fba35 2nd stage bootloader
I (45) boot: compile time 18:48:10
...
```

如果打印出的日志是可读的（而不是乱码），则表示串口连接正常。此时，可以继续进行安装，并最终将应用程序上载到 ESP32-P4。

备注： 在某些串口接线方式下，在 ESP32-P4 启动并开始打印串口日志前，需要在终端程序中禁用串口 RTS & DTR 管脚。该问题仅存在于将 RTS & DTR 管脚直接连接到 EN & GPIO35 管脚上的情况，绝大多数开发板（包括乐鑫所有的开发板）都没有这个问题。更多详细信息，请参考 esptool 文档。

如在安装 ESP32-P4 硬件开发的软件环境时，从*第五步：开始使用 ESP-IDF 吧* 跳转到了这里，请从*第五步：开始使用 ESP-IDF 吧* 继续阅读。

烧录故障排除

连接失败 如果在运行给定命令时出现如“连接失败”这样的错误，造成该错误的原因之一可能是运行 `esptool.py` 时出现错误。`esptool.py` 是构建系统调用的程序，用于重置芯片、与 ROM 引导加载器交互以及烧录固件的工具。可以按照以下步骤进行手动复位，轻松解决该问题。如果问题仍未解决，请参考 esptool 故障排除 获取更多信息。

`esptool.py` 通过使 USB 至 UART 桥（如 FTDI 或 CP210x）的 DTR 和 RTS 控制线生效来自动复位 ESP32-P4（请参考*与 ESP32-P4 创建串口连接* 获取更多详细信息）。DTR 和 RTS 控制线又连接到 ESP32-P4 的 `GPIO35` 和 `CHIP_PU` (EN) 管脚上，因此 DTR 和 RTS 的电压电平变化会使 ESP32-P4 进入固件下载模式。相关示例可查看 ESP32 DevKitC 开发板的 原理图。

一般来说，使用官方的 ESP-IDF 开发板不会出现问题。但是，`esptool.py` 在以下情况下不能自动重置硬件：

- 硬件未连接到 `GPIO35` 和 `CIHP_PU` 的 DTR 和 RTS 控制线。
- DTR 和 RTS 控制线的配置方式不同。
- 不存在这样的串行控制线路。

根据硬件的种类，也可以将 ESP32-P4 开发板手动设置为固件下载模式（复位）。

- 对于乐鑫开发板，可以参考对应开发板的入门指南或用户指南。例如，可以通过按住 `Boot` 按钮 (`GPIO35`) 再按住 `EN` 按钮 (`CHIP_PU`) 来手动复位 ESP-IDF 开发板。
- 对于其他类型的硬件，可以尝试将 `GPIO35` 拉低。

Linux 和 macOS 平台工具链的标准设置

详细安装步骤 请根据下方详细步骤，完成安装过程。

设置开发环境 以下是为 ESP32-P4 设置 ESP-IDF 的具体步骤。

- *第一步：安装准备*
- *第二步：获取 ESP-IDF*
- *第三步：设置工具*
- *第四步：设置环境变量*
- *第五步：开始使用 ESP-IDF 吧*

第一步：安装准备 为了在 ESP32-P4 中使用 ESP-IDF，需要根据操作系统安装一些软件包。可以参考以下安装指南，安装 Linux 和 macOS 的系统上所有需要的软件包。

Linux 用户 编译 ESP-IDF 需要以下软件包。请根据使用的 Linux 发行版本，选择合适的安装命令。

- Ubuntu 和 Debian:

```
sudo apt-get install git wget flex bison gperf python3 python3-pip python3-
↪venv cmake ninja-build ccache libffi-dev libssl-dev dfu-util libusb-1.0-0
```

- CentOS 7 & 8:

```
sudo yum -y update && sudo yum install git wget flex bison gperf python3␣
↪python3-setuptools cmake ninja-build ccache dfu-util libusbx
```

目前仍然支持 CentOS 7，但为了更好的用户体验，建议使用 CentOS 8。

- Arch:

```
sudo pacman -S --needed gcc git make flex bison gperf python cmake ninja␣
↪ccache dfu-util libusb
```

备注:

- 使用 ESP-IDF 需要 CMake 3.16 或以上版本。较早的 Linux 发行版可能需要升级自身的软件源仓库，或开启 backports 套件库，或安装"cmake3" 软件包（不是安装"cmake"）。
- 如果上述列表中没有当前所用系统，请参考所用系统的相关文档，查看安装软件包所用的命令。

---

macOS 用户 ESP-IDF 将使用 macOS 上默认安装的 Python 版本。

- 安装 CMake 和 Ninja 编译工具：
  - 若有 HomeBrew，可以运行:

```
brew install cmake ninja dfu-util
```

  - 若有 MacPorts，可以运行:

```
sudo port install cmake ninja dfu-util
```

  - 若以上均不适用，请访问 CMake 和 Ninja 主页，查询有关 macOS 平台的下载安装问题。
- 强烈建议同时安装 ccache 以获得更快的编译速度。如有 HomeBrew，可通过 MacPorts 上的 `brew install ccache` 或 `sudo port install ccache` 完成安装。

备注: 如在上述任何步骤中遇到以下错误:

```
xcrun: error: invalid active developer path (/Library/Developer/CommandLineTools),␣
↪missing xcrun at: /Library/Developer/CommandLineTools/usr/bin/xcrun
```

则必须安装 XCode 命令行工具，可运行 `xcode-select --install` 命令进行安装。

Apple M1 用户 如果使用的是 Apple M1 系列且看到如下错误提示:

```
WARNING: directory for tool xtensa-esp32-elf version esp-2021r2-patch3-8.4.0 is␣
↪present, but tool was not found
ERROR: tool xtensa-esp32-elf has no installed versions. Please run 'install.sh' to␣
↪install it.
```

或者:

```
zsh: bad CPU type in executable: ~/.espressif/tools/xtensa-esp32-elf/esp-2021r2-
↪patch3-8.4.0/xtensa-esp32-elf/bin/xtensa-esp32-elf-gcc
```

运行如下命令，安装 Apple Rosetta 2：

```
/usr/sbin/softwareupdate --install-rosetta --agree-to-license
```

安装 Python 3 Catalina 10.15 发布说明 中表示不推荐使用 Python 2.7 版本，在未来的 macOS 版本中也不会默认包含 Python 2.7。执行以下命令来检查当前使用的 Python 版本:

```
python --version
```

如果输出结果是 `Python 2.7.17`，则代表默认解析器是 Python 2.7。这时需要运行以下命令检查电脑上是否已经安装过 Python 3:

```
python3 --version
```

如果运行上述命令出现错误，则代表电脑上没有安装 Python 3。

请根据以下步骤安装 Python 3：

- 使用 HomeBrew 进行安装的方法如下:

  ```
  brew install python3
  ```

- 使用 MacPorts 进行安装的方法如下:

  ```
  sudo port install python38
  ```

第二步：获取 ESP-IDF 在围绕 ESP32-P4 构建应用程序之前，请先获取乐鑫提供的软件库文件 ESP-IDF 仓库。

获取 ESP-IDF 的本地副本：打开终端，切换到要保存 ESP-IDF 的工作目录，使用 `git clone` 命令克隆远程仓库。针对不同操作系统的详细步骤，请见下文。

打开终端，运行以下命令：

```
mkdir -p ~/esp
cd ~/esp
git clone -b v5.3.2 --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF 将下载至 `~/esp/esp-idf`。

请前往*ESP-IDF 版本简介*，查看 ESP-IDF 不同版本的具体适用场景。

第三步：设置工具 除了 ESP-IDF 本身，还需要为支持 ESP32-P4 的项目安装 ESP-IDF 使用的各种工具，比如编译器、调试器、Python 包等。

```
cd ~/esp/esp-idf
./install.sh esp32p4
```

或使用 Fish shell：

```
cd ~/esp/esp-idf
./install.fish esp32p4
```

上述命令仅仅为 ESP32-P4 安装所需工具。如果需要为多个目标芯片开发项目，则可以一次性指定多个目标，如下所示：

```
cd ~/esp/esp-idf
./install.sh esp32,esp32s2
```

或使用 Fish shell：

```
cd ~/esp/esp-idf
./install.fish esp32,esp32s2
```

如果需要一次性为所有支持的目标芯片安装工具，可以运行如下命令：

```
cd ~/esp/esp-idf
./install.sh all
```

或使用 Fish shell：

```
cd ~/esp/esp-idf
./install.fish all
```

备注： 对于 macOS 用户，如在上述任何步骤中遇到以下错误:

```
<urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable␣
↪to get local issuer certificate (_ssl.c:xxx)
```

可运行电脑 Python 文件夹中的 `Install Certificates.command` 安装证书。了解更多信息，请参考安装 ESP-IDF 工具时出现的下载错误。

下载工具备选方案 ESP-IDF 工具安装器会下载 Github 发布版本中附带的一些工具，如果访问 Github 较为缓慢，可以设置一个环境变量，从而优先选择 Espressif 的下载服务器进行 Github 资源下载。

备注： 该设置只影响从 Github 发布版本中下载的单个工具，它并不会改变访问任何 Git 仓库的 URL。

要在安装工具时优先选择 Espressif 下载服务器，请在运行 `install.sh` 时使用以下命令：

```
cd ~/esp/esp-idf
export IDF_GITHUB_ASSETS="dl.espressif.com/github_assets"
./install.sh
```

备注： 推荐国内用户使用国内的下载服务器，以加快下载速度。

```
cd ~/esp/esp-idf
export IDF_GITHUB_ASSETS="dl.espressif.cn/github_assets"
./install.sh
```

自定义工具安装路径 本步骤中介绍的脚本将 ESP-IDF 所需的编译工具默认安装在用户的根目录中，即 Linux 系统中的 `$HOME/.espressif` 目录。可以选择将工具安装到其他目录中，**但请在运行安装脚本前，导出环境变量 IDF_TOOLS_PATH**。注意，请确保用户账号已经具备了读写该路径的权限。

```
export IDF_TOOLS_PATH="$HOME/required_idf_tools_path"
./install.sh

. ./export.sh
```

如果修改了 `IDF_TOOLS_PATH` 变量，请在运行任意 ESP-IDF 工具或脚本前，将该变量导出到环境变量中。

备注： 如未导出环境变量，大多数 shell 将不支持在变量赋值中使用 `IDF_TOOLS_PATH`，例如 `IDF_TOOLS_PATH="$HOME/required_idf_tools_path" ./install.sh`。因为即便在源脚本中

导出或修改了该变量，当前的执行环境也不受变量赋值影响。

第四步：设置环境变量 此时，刚刚安装的工具尚未添加至 PATH 环境变量，无法通过“命令窗口”使用这些工具。因此，必须设置一些环境变量。这可以通过 ESP-IDF 提供的另一个脚本进行设置。

请在需要运行 ESP-IDF 的终端窗口运行以下命令：

```
. $HOME/esp/esp-idf/export.sh
```

对于 fish shell（仅支持 fish 3.0.0 及以上版本），请运行以下命令：

```
. $HOME/esp/esp-idf/export.fish
```

注意，命令开始的”.” 与路径之间应有一个空格！

如果需要经常运行 ESP-IDF，可以为执行 `export.sh` 创建一个别名，具体步骤如下：

1. 复制并粘贴以下命令到 shell 配置文件中（`.profile`、`.bashrc`、`.zprofile` 等）

   ```
   alias get_idf='. $HOME/esp/esp-idf/export.sh'
   ```

2. 通过重启终端窗口或运行 `source [path to profile]`，如 `source ~/.bashrc` 来刷新配置文件。

现在可以在任何终端窗口中运行 `get_idf` 来设置或刷新 ESP-IDF 环境。

不建议直接将 `export.sh` 添加到 shell 的配置文件。这样做会导致在每个终端会话中都激活 IDF 虚拟环境（包括无需使用 ESP-IDF 的会话）。这违背了使用虚拟环境的目的，还可能影响其他软件的使用。

第五步：开始使用 ESP-IDF 吧 现在你已经具备了使用 ESP-IDF 的所有条件，接下来将介绍如何开始第一个工程。

本指南将介绍如何初步上手 ESP-IDF，包括如何使用 ESP32-P4 创建第一个工程，并构建、烧录和监控设备输出。

备注： 如果还未安装 ESP-IDF，请参照安装 中的步骤，获取使用本指南所需的所有软件。

开始创建工程 现在，可以准备开发 ESP32-P4 应用程序了。可以从 ESP-IDF 中 examples 目录下的 get-started/hello_world 工程开始。

重要： ESP-IDF 编译系统不支持 ESP-IDF 路径或其工程路径中带有空格。

将 get-started/hello_world 工程复制至本地的 `~/esp` 目录下：

```
cd ~/esp
cp -r $IDF_PATH/examples/get-started/hello_world .
```

备注： ESP-IDF 的 examples 目录下有一系列示例工程，可以按照上述方法复制并运行其中的任何示例，也可以直接编译示例，无需进行复制。

连接设备 现在，请将 ESP32-P4 开发板连接到 PC，并查看开发板使用的串口。

通常，串口在不同操作系统下显示的名称有所不同：

- **Linux 操作系统：** 以 `/dev/tty` 开头

- **macOS 操作系统：** 以 `/dev/cu.` 开头

有关如何查看串口名称的详细信息，请见*与 ESP32-P4 创建串口连接*。

备注： 请记住串口名，以便后续使用。

配置工程 请进入 `hello_world` 目录，设置 ESP32-P4 为目标芯片，然后运行工程配置工具 `menuconfig`。

```
cd ~/esp/hello_world
idf.py set-target esp32p4
idf.py menuconfig
```

打开一个新工程后，应首先使用 `idf.py set-target esp32p4` 设置“目标”芯片。注意，此操作将清除并初始化项目之前的编译和配置（如有）。也可以直接将“目标”配置为环境变量（此时可跳过该步骤）。更多信息，请见*选择目标芯片：set-target*。

正确操作上述步骤后，系统将显示以下菜单：

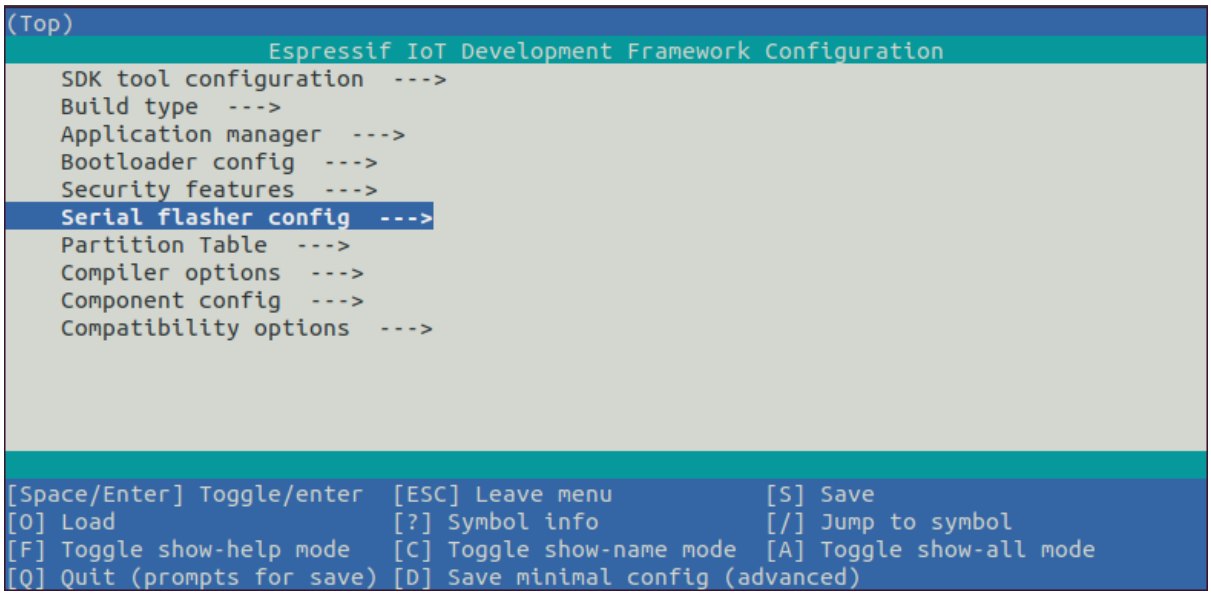


图 13: 工程配置—主窗口

可以通过此菜单设置项目的具体变量，包括 Wi-Fi 网络名称、密码和处理器速度等。`hello_world` 示例项目会以默认配置运行，因此在这一项目中，可以跳过使用 `menuconfig` 进行项目配置这一步骤。

备注： 终端窗口中显示出的菜单颜色可能会与上图不同。可以通过选项 `--style` 来改变外观。请运行 `idf.py menuconfig --help` 命令，获取更多信息。

编译工程 请使用以下命令，编译烧录工程：

```
idf.py build
```

运行以上命令可以编译应用程序和所有 ESP-IDF 组件，接着生成引导加载程序、分区表和应用程序二进制文件。

```
$ idf.py build
Running cmake in directory /path/to/hello_world/build
Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
Warn about uninitialized values.
-- Found Git: /usr/bin/git (found version "2.17.0")
-- Building empty aws_iot component due to configuration
-- Component names: ...
-- Component paths: ...

... (more lines of build system output)

[527/527] Generating hello_world.bin
esptool.py v2.3.1

Project build complete. To flash, run this command:
../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600 write_flash -
↪-flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/hello_world.
↪bin  build 0x1000 build/bootloader/bootloader.bin 0x8000 build/partition_table/
↪partition-table.bin
or run 'idf.py -p PORT flash'
```

如果一切正常，编译完成后将生成.bin 文件。

烧录到设备 请运行以下命令，将刚刚生成的二进制文件烧录至 ESP32-P4 开发板：

```
idf.py -p PORT flash
```

请将 PORT 替换为 ESP32-P4 开发板的串口名称。如果 `PORT` 未经定义，*idf.py* 将尝试使用可用的串口自动连接。

更多有关 idf.py 参数的详情，请见*idf.py*。

备注： 勾选 `flash` 选项将自动编译并烧录工程，因此无需再运行 `idf.py build`。

若在烧录过程中遇到问题，请参考下文中的 “其他提示”。也可以前往烧录故障排除 或与 *ESP32-P4* 创建串口连接 获取更多详细信息。

常规操作 在烧录过程中，会看到类似如下的输出日志：

```
...
esptool.py v4.8.dev3
Serial port /dev/cu.SLAB_USBtoUART
Connecting....
Chip is ESP32-P4 (revision v0.0)
Features: High-Performance MCU
Crystal is 40MHz
MAC: 00:00:00:00:00:00
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00002000 to 0x00007fff...
Flash will be erased from 0x00010000 to 0x0003bfff...
Flash will be erased from 0x00008000 to 0x00008fff...
SHA digest in image updated
Compressed 20960 bytes to 12653...
Writing at 0x00002000... (100 %)
```

(下页继续)

(续上页)

```
Wrote 20960 bytes (12653 compressed) at 0x00002000 in 0.6 seconds (effective 277.1␣
↪kbit/s)...
Hash of data verified.
Compressed 179584 bytes to 94528...
Writing at 0x00035efb... (100 %)
Wrote 179584 bytes (94528 compressed) at 0x00010000 in 2.6 seconds (effective 549.
↪9 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (effective 420.7␣
↪kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
Done
```

如果一切顺利，烧录完成后，开发板将会复位，应用程序"hello_world" 开始运行。

如果希望使用 Eclipse 或是 VS Code IDE，而非 `idf.py`，请参考 Eclipse Plugin，以及 VSCode Extension。

监视输出 可以使用 `idf.py -p PORT monitor` 命令，监视 "hello_world" 工程的运行情况。注意，不要忘记将 PORT 替换为自己的串口名称。

运行该命令后，*IDF 监视器* 应用程序将启动。

```
$ idf.py -p <PORT> monitor
Running idf_monitor in directory [...]/esp/hello_world/build
Executing "python [...]/esp-idf/tools/idf_monitor.py -b 115200 [...]/esp/hello_
↪world/build/hello_world.elf"...
--- idf_monitor on <PORT> 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57
...
```

此时，就可以在启动日志和诊断日志之后，看到打印的 "Hello world!" 了。

```
    ...
    Hello world!
    Restarting in 10 seconds...
    This is esp32p4 chip with 2 CPU core(s), silicon revision v0.0, 2 MB external␣
↪flash
Minimum free heap size: 618848 bytes
    Restarting in 9 seconds...
    Restarting in 8 seconds...
    Restarting in 7 seconds...
```

使用快捷键 `Ctrl+]`，可退出 ESP-IDF 监视器。

备注： 也可以运行以下命令，一次性执行构建、烧录和监视过程：

```
idf.py -p PORT flash monitor
```

此外，

- 请前往*IDF 监视器*，了解更多使用 ESP-IDF 监视器的快捷键和其他详情。
- 请前往*idf.py*，查看更多 `idf.py` 命令和选项。

恭喜完成 ESP32-P4 的入门学习！

现在，可以尝试一些其他 examples，或者直接开发自己的应用程序。

重要： 一些示例程序不支持 ESP32-P4，因为 ESP32-P4 中不包含所需的硬件。

在编译示例程序前请查看 README 文件中 `Supported Targets` 表格。如果表格中包含 ESP32-P4，或者不存在这个表格，那么即表示 ESP32-P4 支持这个示例程序。

其他提示

权限问题 使用某些 Linux 版本向 ESP32-P4 烧录固件时，可能会出现类似 `Could not open port <PORT>: Permission denied: '<PORT>'` 错误消息。此时可以在 Linux 将用户添加至*dialout 组*或 *uucp 组* 来解决此类问题。

兼容的 Python 版本 ESP-IDF 支持 Python 3.8 及以上版本，建议升级操作系统到最新版本从而更新 Python。也可选择从 sources 安装最新版 Python，或使用 Python 管理系统如 pyenv 对版本进行升级管理。

擦除 flash ESP-IDF 支持擦除 flash。请运行以下命令，擦除整个 flash：

```
idf.py -p PORT erase-flash
```

若存在需要擦除的 OTA 数据，请运行以下命令：

```
idf.py -p PORT erase-otadata
```

擦除 flash 需要一段时间，在擦除过程中，请勿断开设备连接。

建议：更新 ESP-IDF 乐鑫会不时推出新版本的 ESP-IDF，修复 bug 或提供新的功能。请注意，ESP-IDF 的每个主要版本和次要版本都有相应的支持期限。支持期限满后，版本停止更新维护，用户可将项目升级到最新的 ESP-IDF 版本。更多关于支持期限的信息，请参考*ESP-IDF 版本*。

因此，在使用时，也应注意更新本地版本。最简单的方法是：直接删除本地的 `esp-idf` 文件夹，然后按照*第二步：获取 ESP-IDF* 中的指示，重新完成克隆。

另一种方法是仅更新变更的部分，具体方式请前往*更新 ESP-IDF* 章节查看。具体更新步骤会根据使用的 ESP-IDF 版本有所不同。

注意，更新完成后，请再次运行安装脚本，以防新版 ESP-IDF 所需的工具也有所更新。具体请参考*第三步：设置工具*。

一旦重新安装好工具，请使用导出脚本更新环境，具体请参考*第四步：设置环境变量*。

相关文档

- *与 ESP32-P4 创建串口连接*
- Eclipse Plugin
- VSCode Extension
- *IDF 监视器*

1.4 编译第一个工程

如果已经安装好 ESP-IDF，且没有使用集成开发环境 (IDE)，请在命令提示行中，按照*在 Windows 中开始创建工程* 或*在 Linux 和 macOS 中开始创建工程* 编译第一个工程。

1.5 卸载 ESP-IDF

如需卸载 ESP-IDF，请参考*卸载 ESP-IDF*。

Chapter 2

API 参考

2.1 API 约定

本文介绍了 ESP-IDF 应用程序编程接口 (API) 中常见的约定和假设。

ESP-IDF 提供了几种编程接口：

- 在 ESP-IDF 组件的公共头文件中声明的 C 函数、结构体、枚举、类型定义和预处理器宏。编程指南的 API 参考部分描述了这些函数、结构体和类型。
- 编译系统函数、预定义变量和选项，详情请参阅*ESP-IDF CMake 构建系统 API*。
- *Kconfig* 选项，可用于代码及编译系统文件 (`CMakeLists.txt`)。
- *主机工具* 及其命令行参数。

ESP-IDF 由多个组件组成，组件中包含专门为 ESP 芯片编写的代码或第三方库（即第三方组件）。对于某些第三方库，ESP-IDF 提供专用的包装器和接口，以简化对第三方库的使用，或提高其与 ESP-IDF 其他功能的兼容性。某些情况下，第三方组件将直接呈现底层库的原始 API。

以下各节介绍了部分 ESP-IDF API 及其使用的相关内容。

2.1.1 错误处理

多数 ESP-IDF API 会返回由*esp_err_t* 类型定义的错误代码。有关出错处理的更多信息，请参阅*错误处理* 部分。有关 ESP-IDF 组件返回的错误代码列表，请参阅*错误代码参考*。

2.1.2 配置结构体

重要： 为确保应用程序与未来 ESP-IDF 版本的兼容性，请正确初始化配置结构体。

多数 ESP-IDF 中的初始化、配置和安装函数（通常以 `..._init()`、`..._config()` 和 `..._install()` 命名）都需要一个指向配置结构体的指针作为参数。例如：

```
const esp_timer_create_args_t my_timer_args = {
    .callback = &my_timer_callback,
    .arg = callback_arg,
    .name = "my_timer"
};
```

（下页继续）

(续上页)

```
esp_timer_handle_t my_timer;
esp_err_t err = esp_timer_create(&my_timer_args, &my_timer);
```

初始化函数不会存储指向配置结构体的指针，因此在栈上分配结构体是安全的。

应用程序必须初始化结构体的所有字段，以下为错误示例:

```
esp_timer_create_args_t my_timer_args;
my_timer_args.callback = &my_timer_callback;
/* 错误！字段 .arg 和 .name 未初始化 */
esp_timer_create(&my_timer_args, &my_timer);
```

大多数 ESP-IDF 示例使用 C99 的 指定初始化器 来完成结构体初始化，从而以简洁的方式设置子集字段，并将剩余字段初始化为零:

```
const esp_timer_create_args_t my_timer_args = {
    .callback = &my_timer_callback,
    /* 正确，字段 .arg 和 .name 已初始化为零 */
};
```

C++ 语言同样支持指定初始化器语法，但初始化器必须遵循声明顺序。在 C++ 代码中使用 ESP-IDF API 时，可以考虑使用以下模式:

```
/* 正确：.dispatch_method、.name 以及 .skip_unhandled_events 初始化为零 */
const esp_timer_create_args_t my_timer_args = {
    .callback = &my_timer_callback,
    .arg = &my_arg,
};

///* 错误：esp_timer_create_args_t 中，.arg 在 .callback 之后声明 */
//const esp_timer_create_args_t my_timer_args = {
//    .arg = &my_arg,
//    .callback = &my_timer_callback,
//};
```

了解指定初始化器的更多信息，请参见指定初始化器。注意，C++20 之前的 C++ 语言不是当前 ESP-IDF 的默认版本，不支持指定初始化器。如需使用 C++20 之前的 C++ 标准编译代码，可以借助 GCC 扩展生成以下模式:

```
esp_timer_create_args_t my_timer_args = {};
/* 所有字段初始化为零 */
my_timer_args.callback = &my_timer_callback;
```

默认初始化器

ESP-IDF 为某些配置结构体提供了用于设置字段默认值的宏:

```
httpd_config_t config = HTTPD_DEFAULT_CONFIG();
/* HTTPD_DEFAULT_CONFIG␣
↪扩展到一个指定的初始化器。此时，所有字段均已设置为默认值，且支持编辑： */
config.server_port = 8081;
httpd_handle_t server;
esp_err_t err = httpd_start(&server, &config);
```

当特定配置结构体提供了默认初始化器宏时，推荐使用该默认初始化器宏。

2.1.3 私有 API

在 ESP-IDF 中，某些头文件包含的 API 仅限于在 ESP-IDF 源代码中使用，不支持在应用程序中使用。此类头文件的名称或路径通常带有 `private` 或 `esp_private`。某些组件（如*hal*）则仅包含私有 API。

私有 API 可能在次要或补丁版本之间以不兼容的方式被删除或更改。

2.1.4 示例项目组件

ESP-IDF 示例中提供了一系列演示 ESP-IDF API 使用方式的工程。为避免在各个示例中重复引用相同的代码片段，示例的常用组件中定义了一些通用辅助工具。这些常用组件包括 common_components 目录下的组件和示例本身的部分组件，它们不属于 ESP-IDF API 的范畴。

不建议在自定义项目中通过 `EXTRA_COMPONENT_DIRS` 编译系统变量直接引用这些组件，因为在不同的 ESP-IDF 版本中，组件可能存在显著变化。基于 ESP-IDF 示例开始新项目时，需将项目及其所依赖的公共组件从 ESP-IDF 中复制出来，并将公共组件视为项目的一部分。请注意，公共组件是针对示例编写的，可能不包括生产应用程序所需的所有出错处理。在使用前，需阅读代码并判断它是否适用于所需用例。

2.1.5 API 稳定性

ESP-IDF 使用 语义版本管理办法，详情请参阅版本管理。

ESP-IDF 的次要版本和错误修复版本会保证与过往版本的兼容性。以下各节解释了兼容性的不同方面和限制。

源代码级别兼容性

ESP-IDF 确保在 ESP-IDF 组件的公共头文件中声明的 C 函数、结构体、枚举、类型定义和预处理宏的源代码级别兼容性。源代码级别兼容性意味着应用程序无需修改即可在新版本的 ESP-IDF 上重新编译。

以下在次要版本之间的更改不会破坏源代码级别兼容性：

- 使用 `deprecated` 属性废弃函数、使用预处理器 `#warning` 废弃头文件。废弃功能已在 ESP-IDF 发布说明中列出。建议更新源代码以使用替换被废弃的函数或文件的新函数或文件。ESP-IDF 的主要版本将移除废弃的函数和文件。
- 重命名组件，在组件间移动源代码和头文件，但需确保编译系统仍可以找到正确的文件。
- 重命名 Kconfig 选项。Kconfig 系统的向后兼容性 确保应用程序在 `sdkconfig` 文件、CMake 文件和源代码中仍然可以使用原始的 Kconfig 选项名称。

缺少二进制兼容性

ESP-IDF 无法确保版本间的二进制兼容性。这意味着，如果使用某个 ESP-IDF 版本构建了一个预编译库，在下一个次要或错误修复版本中，无法确保该库将以相同方式运行。以下更改可以保持源代码级别兼容性，但不保证二进制兼容性：

- 更改 C 枚举成员的数值。
- 添加新的结构体成员或更改成员顺序。关于有助于确保兼容性的提示，请参阅配置结构体。
- 用具有相同签名的 `static inline` 函数替换 `extern` 函数，反之亦然。
- 用兼容的 C 函数替换类似于函数的宏。

其他不兼容情况

尽管我们致力于优化 ESP-IDF 版本升级，但是在次要版本之间，ESP-IDF 的某些部分可能会不兼容。如有不属于下列情况的意外重大更新，欢迎向我们发送报告：

- *私有 API*。
- 示例项目组件。

- 明确标记为"beta"、"preview" 或"experimental" 的功能。
- 为缓解安全问题做出的更改，或以更安全的行为取代不安全的默认行为的更改。
- 从未运行成功的功能。例如，如果某个函数或枚举值从未成功使用，则可能会以修复的形式将其重命名或删除。这包括依赖于非功能芯片硬件功能的软件功能。
- 未明确记录的意外或未定义行为可能会被修复或更改，如缺少参数范围验证。
- 在菜单配置中*Kconfig* 选项的位置。
- 示例项目的位置和名称。

2.2 应用层协议

2.2.1 ASIO 端口

ASIO 是一个跨平台的 C++ 库，参见 https://think-async.com/Asio/。它采用现代 C++ 方法提供了一个一致的异步模型。

`ASIO` 组件自 ESP-IDF 版本 v5.0 起移到了单独的仓库：

- GitHub ASIO 组件

运行 `idf.py add-dependency espressif/asio` 将 ASIO 组件添加到你的项目中。

文档

访问以下链接查看相关文档：

- ASIO 文档 (English)

2.2.2 ESP-Modbus

乐鑫的 ESP-Modbus 库 (esp-modbus) 支持基于 RS485、Wi-Fi 和以太网接口的 Modbus 通信。自 ESP-IDF v5.0 版本以来，组件 `freemodbus` 已被移动到单独的代码仓库中：

- GitHub 上的 ESP-Modbus 组件

托管文档

相应文档请参阅：

- ESP-Modbus 文档

应用示例

以下示例分别介绍了 ESP-Modbus 库的串行端口、TCP 端口的从机和主机实现。

- protocols/modbus/serial/mb_slave
- protocols/modbus/serial/mb_master
- protocols/modbus/tcp/mb_tcp_slave
- protocols/modbus/tcp/mb_tcp_master

详情请参阅具体示例的 `README.md` 。

协议参考

- Modbus 组织与规范协议请参阅 The Modbus Organization。

2.2.3 ESP-MQTT

概述

ESP-MQTT 是 MQTT 协议客户端的实现，MQTT 是一种基于发布/订阅模式的轻量级消息传输协议。ESP-MQTT 当前支持 MQTT v5.0。

特性

- 支持基于 TCP 的 MQTT、基于 Mbed TLS 的 SSL、基于 WebSocket 的 MQTT 以及基于 WebSocket Secure 的 MQTT
- 通过 URI 简化配置流程
- 多个实例（一个应用程序中有多个客户端）
- 支持订阅、发布、认证、遗嘱消息、保持连接心跳机制以及 3 个服务质量 (QoS) 级别（组成全功能客户端）

应用示例

- protocols/mqtt/tcp：基于 TCP 的 MQTT，默认端口 1883
- protocols/mqtt/ssl：基于 TLS 的 MQTT，默认端口 8883
- protocols/mqtt/ssl_ds：基于 TLS 的 MQTT，使用数字签名外设进行身份验证，默认端口 8883
- protocols/mqtt/ssl_mutual_auth：基于 TLS 的 MQTT，使用证书进行身份验证，默认端口 8883
- protocols/mqtt/ssl_psk：基于 TLS 的 MQTT，使用预共享密钥进行身份验证，默认端口 8883
- protocols/mqtt/ws：基于 WebSocket 的 MQTT，默认端口 80
- protocols/mqtt/wss：基于 WebSocket Secure 的 MQTT，默认端口 443
- protocols/mqtt5: 使用 ESP-MQTT 库连接 MQTT v5.0 的服务器

MQTT 消息重传

调用`esp_mqtt_client_publish`或其非阻塞形式`esp_mqtt_client_enqueue`，可以创建新的 MQTT 消息。

QoS 0 的消息将只发送一次，QoS 1 和 2 具有不同行为，因为协议需要执行额外步骤来完成该过程。

ESP-MQTT 库将始终重新传输未确认的 QoS 1 和 2 发布消息，以避免连接错误导致信息丢失，虽然 MQTT 规范要求仅在重新连接且 Clean Session 标志设置为 0 时重新传输（针对此行为，将`disable_clean_session`设置为 true）。

可能需要重传的 QoS 1 和 2 消息总是处于排队状态，但若使用`esp_mqtt_client_publish`则会立即进行第一次传输尝试。未确认消息的重传将在`message_retransmit_timeout`之后进行。在*CONFIG_MQTT_OUTBOX_EXPIRED_TIMEOUT_MS*之后，消息会过期并被删除。如已设置*CONFIG_MQTT_REPORT_DELETED_MESSAGES*，则会发送事件来通知用户。

配置

通过设置`esp_mqtt_client_config_t`结构体中的字段来进行配置。配置结构体包含以下子结构体，用于配置客户端的多种操作。

- `esp_mqtt_client_config_t::broker_t` - 允许设置地址和安全验证。

- *esp_mqtt_client_config_t::credentials_t* - 用于身份验证的客户端凭据。
- *esp_mqtt_client_config_t::session_t* - MQTT 会话相关配置。
- *esp_mqtt_client_config_t::network_t* - 网络相关配置。
- *esp_mqtt_client_config_t::task_t* - 允许配置 FreeRTOS 任务。
- *esp_mqtt_client_config_t::buffer_t* - 输入输出的缓冲区大小。

下文将详细介绍不同配置。

服务器

地址 通过*address* 结构体的*uri* 字段或者*hostname*、*transport* 以及*port* 的组合，可以设置服务器地址。也可以选择设置*path*，该字段对 WebSocket 连接而言非常有用。

使用*uri* 字段的格式为 `scheme://hostname:port/path`。

- 当前支持 `mqtt`、`mqtts`、`ws` 和 `wss` 协议
- 基于 TCP 的 MQTT 示例：
  - `mqtt://mqtt.eclipseprojects.io`：基于 TCP 的 MQTT，默认端口 1883
  - `mqtt://mqtt.eclipseprojects.io:1884`：基于 TCP 的 MQTT，端口 1884
  - `mqtt://username:password@mqtt.eclipseprojects.io:1884`：基于 TCP 的 MQTT，端口 1884，带有用户名和密码
- 基于 SSL 的 MQTT 示例：
  - `mqtts://mqtt.eclipseprojects.io`：基于 SSL 的 MQTT，端口 8883
  - `mqtts://mqtt.eclipseprojects.io:8884`：基于 SSL 的 MQTT，端口 8884
- 基于 WebSocket 的 MQTT 示例：
  - `ws://mqtt.eclipseprojects.io:80/mqtt`
- 基于 WebSocket Secure 的 MQTT 示例：
  - `wss://mqtt.eclipseprojects.io:443/mqtt`
- 最简配置：

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .broker.address.uri = "mqtt://mqtt.eclipseprojects.io",
};
esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler,␣
↪client);
esp_mqtt_client_start(client);
```

备注： 默认情况下，MQTT 客户端使用事件循环库来发布相关 MQTT 事件（已连接、已订阅、已发布等）。

验证 为验证服务器身份，对于使用 TLS 的安全链接，必须设置*verification* 结构体。服务器证书可设置为 PEM 或 DER 格式。如要选择 DER 格式，必须设置等效*certificate_len* 字段，否则应在*certificate* 字段传入以空字符结尾的 PEM 格式字符串。

- **从服务器获取证书，例如：`mqtt.eclipseprojects.io`**

```
openssl s_client -showcerts -connect mqtt.eclipseprojects.io:8883 < /dev/
↪null \
2> /dev/null | openssl x509 -outform PEM > mqtt_eclipse_org.pem
```

- 检查示例应用程序：protocols/mqtt/ssl
- 配置：

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .broker = {
      .address.uri = "mqtts://mqtt.eclipseprojects.io:8883",
```

(下页继续)

(续上页)

```
        .verification.certificate = (const char *)mqtt_eclipse_org_pem_start,
    },
};
```

了解其他字段的详细信息，请查看*API 参考* 以及*TLS 服务器验证*。

客户端凭据 *credentials* 字段下包含所有客户端相关凭据。

- *username*：指向用于连接服务器用户名的指针，也可通过 URI 设置
- *client_id*：指向客户端 ID 的指针，默认为 ESP32_%CHIPID%，其中 %CHIPID% 是十六进制 MAC 地址的最后 3 个字节

认证 可以通过*authentication* 字段设置认证参数。客户端支持以下认证方式：

- *password*：使用密码
- – *certificate* 和*key*：进行双向 TLS 身份验证，PEM 或 DER 格式均可
- *use_secure_element*：使用 ESP32 中的安全元素 (ATECC608A)
- *ds_data*：使用某些乐鑫设备的数字签名外设

会话 使用*session* 字段进行 MQTT 会话相关配置。

遗嘱消息 (LWT) 通过设置*last_will* 结构体的以下字段，MQTT 会在一个客户端意外断开连接时通过遗嘱消息通知其他客户端。

- *topic*：指向 LWT 消息主题的指针
- *msg*：指向 LWT 消息的指针
- *msg_len*：LWT 消息的长度，*msg* 不以空字符结尾时需要该字段
- *qos*：LWT 消息的服务质量
- *retain*：指定 LWT 消息的保留标志

在项目配置菜单中设置 MQTT 通过 idf.py menuconfig，可以在 Component config > ESP-MQTT Configuration 中找到 MQTT 设置。

相关设置如下：

- *CONFIG_MQTT_PROTOCOL_311*：启用 MQTT 协议 3.1.1 版本
- *CONFIG_MQTT_TRANSPORT_SSL* 和*CONFIG_MQTT_TRANSPORT_WEBSOCKET*：启用特定 MQTT 传输层，例如 SSL、WEBSOCKET 和 WEBSOCKET_SECURE
- *CONFIG_MQTT_CUSTOM_OUTBOX*：禁用 mqtt_outbox 默认实现，因此可以提供特定实现

事件

MQTT 客户端可能会发布以下事件：

- MQTT_EVENT_BEFORE_CONNECT：客户端已初始化并即将开始连接至服务器。
- MQTT_EVENT_CONNECTED：客户端已成功连接至服务器。客户端已准备好收发数据。
- MQTT_EVENT_DISCONNECTED：由于无法读取或写入数据，例如因为服务器无法使用，客户端已终止连接。
- MQTT_EVENT_SUBSCRIBED：服务器已确认客户端的订阅请求。事件数据将包含订阅消息的消息 ID。
- MQTT_EVENT_UNSUBSCRIBED：服务器已确认客户端的退订请求。事件数据将包含退订消息的消息 ID。
- MQTT_EVENT_PUBLISHED：服务器已确认客户端的发布消息。消息将仅针对 QoS 级别 1 和 2 发布，因为级别 0 不会进行确认。事件数据将包含发布消息的消息 ID。

- MQTT_EVENT_DATA：客户端已收到发布消息。事件数据包含：消息 ID、发布消息所属主题名称、收到的数据及其长度。对于超出内部缓冲区的数据，将发布多个 MQTT_EVENT_DATA，并更新事件数据的*current_data_offset* 和*total_data_len* 以跟踪碎片化消息。
- MQTT_EVENT_ERROR：客户端遇到错误。使用事件数据*error_handle* 字段中的*error_type*，可以发现错误。错误类型决定*error_handle* 结构体的哪些部分会被填充。

API 参考

Header File

- components/mqtt/esp-mqtt/include/mqtt_client.h
- This header file can be included with:

```
#include "mqtt_client.h"
```

- This header file is a part of the API provided by the mqtt component. To declare that your component depends on mqtt, add the following to your CMakeLists.txt:

```
REQUIRES mqtt
```

or

```
PRIV_REQUIRES mqtt
```

Functions

esp_mqtt_client_handle_t **esp_mqtt_client_init** (const *esp_mqtt_client_config_t* *config)

Creates *MQTT* client handle based on the configuration.

参数 **config** -- *MQTT* configuration structure
返回 mqtt_client_handle if successfully created, NULL on error

esp_err_t **esp_mqtt_client_set_uri** (*esp_mqtt_client_handle_t* client, const char *uri)

Sets *MQTT* connection URI. This API is usually used to overrides the URI configured in esp_mqtt_client_init.

参数
- **client** -- *MQTT* client handle
- **uri** --

返回 ESP_FAIL if URI parse error, ESP_OK on success

esp_err_t **esp_mqtt_client_start** (*esp_mqtt_client_handle_t* client)

Starts *MQTT* client with already created client handle.

参数 **client** -- *MQTT* client handle
返回 ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization ESP_FAIL on other error

esp_err_t **esp_mqtt_client_reconnect** (*esp_mqtt_client_handle_t* client)

This api is typically used to force reconnection upon a specific event.

参数 **client** -- *MQTT* client handle
返回 ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization ESP_FAIL if client is in invalid state

esp_err_t **esp_mqtt_client_disconnect** (*esp_mqtt_client_handle_t* client)

This api is typically used to force disconnection from the broker.

参数 **client** -- *MQTT* client handle
返回 ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization

esp_err_t **esp_mqtt_client_stop** (*esp_mqtt_client_handle_t* client)

Stops *MQTT* client tasks.

- Notes:
- Cannot be called from the *MQTT* event handler

参数 **client** -- *MQTT* client handle
返回 ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization ESP_FAIL if client is in invalid state

int **esp_mqtt_client_subscribe_single**(*esp_mqtt_client_handle_t* client, const char *topic, int qos)

Subscribe the client to defined topic with defined qos.

Notes:

- Client must be connected to send subscribe message
- This API is could be executed from a user task or from a *MQTT* event callback i.e. internal *MQTT* task (API is protected by internal mutex, so it might block if a longer data receive operation is in progress.
- `esp_mqtt_client_subscribe` could be used to call this function.

参数
- **client** -- *MQTT* client handle
- **topic** -- topic filter to subscribe
- **qos** -- Max qos level of the subscription

返回 message_id of the subscribe message on success -1 on failure -2 in case of full outbox.

int **esp_mqtt_client_subscribe_multiple**(*esp_mqtt_client_handle_t* client, const *esp_mqtt_topic_t* *topic_list, int size)

Subscribe the client to a list of defined topics with defined qos.

Notes:

- Client must be connected to send subscribe message
- This API is could be executed from a user task or from a *MQTT* event callback i.e. internal *MQTT* task (API is protected by internal mutex, so it might block if a longer data receive operation is in progress.
- `esp_mqtt_client_subscribe` could be used to call this function.

参数
- **client** -- *MQTT* client handle
- **topic_list** -- List of topics to subscribe
- **size** -- size of topic_list

返回 message_id of the subscribe message on success -1 on failure -2 in case of full outbox.

int **esp_mqtt_client_unsubscribe**(*esp_mqtt_client_handle_t* client, const char *topic)

Unsubscribe the client from defined topic.

Notes:

- Client must be connected to send unsubscribe message
- It is thread safe, please refer to `esp_mqtt_client_subscribe_single` for details

参数
- **client** -- *MQTT* client handle
- **topic** --

返回 message_id of the subscribe message on success -1 on failure

int **esp_mqtt_client_publish**(*esp_mqtt_client_handle_t* client, const char *topic, const char *data, int len, int qos, int retain)

Client to send a publish message to the broker.

Notes:

- This API might block for several seconds, either due to network timeout (10s) or if publishing payloads longer than internal buffer (due to message fragmentation)

- Client doesn't have to be connected for this API to work, enqueueing the messages with qos>1 (returning -1 for all the qos=0 messages if disconnected). If MQTT_SKIP_PUBLISH_IF_DISCONNECTED is enabled, this API will not attempt to publish when the client is not connected and will always return -1.
- It is thread safe, please refer to `esp_mqtt_client_subscribe` for details

参数

- **client** -- *MQTT* client handle
- **topic** -- topic string
- **data** -- payload string (set to NULL, sending empty payload message)
- **len** -- data length, if set to 0, length is calculated from payload string
- **qos** -- QoS of publish message
- **retain** -- retain flag

返回 message_id of the publish message (for QoS 0 message_id will always be zero) on success. -1 on failure, -2 in case of full outbox.

int **esp_mqtt_client_enqueue**(*esp_mqtt_client_handle_t* client, const char *topic, const char *data, int len, int qos, int retain, bool store)

Enqueue a message to the outbox, to be sent later. Typically used for messages with qos>0, but could be also used for qos=0 messages if store=true.

This API generates and stores the publish message into the internal outbox and the actual sending to the network is performed in the mqtt-task context (in contrast to the esp_mqtt_client_publish() which sends the publish message immediately in the user task's context). Thus, it could be used as a non blocking version of esp_mqtt_client_publish().

参数

- **client** -- *MQTT* client handle
- **topic** -- topic string
- **data** -- payload string (set to NULL, sending empty payload message)
- **len** -- data length, if set to 0, length is calculated from payload string
- **qos** -- QoS of publish message
- **retain** -- retain flag
- **store** -- if true, all messages are enqueued; otherwise only QoS 1 and QoS 2 are enqueued

返回 message_id if queued successfully, -1 on failure, -2 in case of full outbox.

esp_err_t **esp_mqtt_client_destroy**(*esp_mqtt_client_handle_t* client)

Destroys the client handle.

Notes:

- Cannot be called from the *MQTT* event handler

参数 **client** -- *MQTT* client handle
返回 ESP_OK ESP_ERR_INVALID_ARG on wrong initialization

esp_err_t **esp_mqtt_set_config**(*esp_mqtt_client_handle_t* client, const *esp_mqtt_client_config_t* *config)

Set configuration structure, typically used when updating the config (i.e. on "before_connect" event.

Notes:

- When calling this function make sure to have all the intendend configurations set, otherwise default values are set.

参数

- **client** -- *MQTT* client handle
- **config** -- *MQTT* configuration structure

返回 ESP_ERR_NO_MEM if failed to allocate ESP_ERR_INVALID_ARG if conflicts on transport configuration. ESP_OK on success

esp_err_t **esp_mqtt_client_register_event**(*esp_mqtt_client_handle_t* client, *esp_mqtt_event_id_t* event, *esp_event_handler_t* event_handler, void *event_handler_arg)

Registers *MQTT* event.

参数

- **client** -- *MQTT* client handle
- **event** -- event type
- **event_handler** -- handler callback
- **event_handler_arg** -- handlers context

返回 ESP_ERR_NO_MEM if failed to allocate ESP_ERR_INVALID_ARG on wrong initialization ESP_OK on success

esp_err_t **esp_mqtt_client_unregister_event**(*esp_mqtt_client_handle_t* client, *esp_mqtt_event_id_t* event, *esp_event_handler_t* event_handler)

Unregisters mqtt event.

参数

- **client** -- mqtt client handle
- **event** -- event ID
- **event_handler** -- handler to unregister

返回 ESP_ERR_NO_MEM if failed to allocate ESP_ERR_INVALID_ARG on invalid event ID ESP_OK on success

int **esp_mqtt_client_get_outbox_size**(*esp_mqtt_client_handle_t* client)

Get outbox size.

参数 **client** -- *MQTT* client handle

返回 outbox size 0 on wrong initialization

esp_err_t **esp_mqtt_dispatch_custom_event**(*esp_mqtt_client_handle_t* client, *esp_mqtt_event_t* *event)

Dispatch user event to the mqtt internal event loop.

参数

- **client** -- *MQTT* client handle
- **event** -- *MQTT* event handle structure

返回 ESP_OK on success ESP_ERR_TIMEOUT if the event couldn't be queued (ref also CONFIG_MQTT_EVENT_QUEUE_SIZE)

Structures

struct **esp_mqtt_error_codes**

MQTT error code structure to be passed as a contextual information into ERROR event

Important: This structure extends `esp_tls_last_error` error structure and is backward compatible with it (so might be down-casted and treated as `esp_tls_last_error` error, but recommended to update applications if used this way previously)

Use this structure directly checking error_type first and then appropriate error code depending on the source of the error:

| error_type | related member variables | note | | MQTT_ERROR_TYPE_TCP_TRANSPORT | esp_tls_last_esp_err, esp_tls_stack_err, esp_tls_cert_verify_flags, sock_errno | Error reported from tcp_transport/esp-tls | | MQTT_ERROR_TYPE_CONNECTION_REFUSED | connect_return_code | Internal error reported from *MQTT* broker on connection |

Public Members

esp_err_t **esp_tls_last_esp_err**

last esp_err code reported from esp-tls component

int **esp_tls_stack_err**

tls specific error code reported from underlying tls stack

int **esp_tls_cert_verify_flags**

tls flags reported from underlying tls stack during certificate verification

esp_mqtt_error_type_t **error_type**

error type referring to the source of the error

esp_mqtt_connect_return_code_t **connect_return_code**

connection refused error code reported from MQTT* broker on connection

int **esp_transport_sock_errno**

errno from the underlying socket

struct **esp_mqtt_event_t**

MQTT event configuration structure

Public Members

esp_mqtt_event_id_t **event_id**

MQTT event type

esp_mqtt_client_handle_t **client**

MQTT client handle for this event

char ***data**

Data associated with this event

int **data_len**

Length of the data for this event

int **total_data_len**

Total length of the data (longer data are supplied with multiple events)

int **current_data_offset**

Actual offset for the data associated with this event

char ***topic**

Topic associated with this event

int **topic_len**

Length of the topic for this event associated with this event

int **msg_id**

MQTT messaged id of message

int **session_present**

MQTT session_present flag for connection event

esp_mqtt_error_codes_t ***error_handle**

esp-mqtt error handle including esp-tls errors as well as internal *MQTT* errors

bool **retain**

Retained flag of the message associated with this event

int **qos**

QoS of the messages associated with this event

bool **dup**

dup flag of the message associated with this event

esp_mqtt_protocol_ver_t **protocol_ver**

MQTT protocol version used for connection, defaults to value from menuconfig

struct **esp_mqtt_client_config_t**

MQTT client configuration structure

- Default values can be set via menuconfig
- All certificates and key data could be passed in PEM or DER format. PEM format must have a terminating NULL character and the related len field set to 0. DER format requires a related len field set to the correct length.

Public Members

struct *esp_mqtt_client_config_t*::*broker_t* **broker**

Broker address and security verification

struct *esp_mqtt_client_config_t*::*credentials_t* **credentials**

User credentials for broker

struct *esp_mqtt_client_config_t*::*session_t* **session**

MQTT session configuration.

struct *esp_mqtt_client_config_t*::*network_t* **network**

Network configuration

struct *esp_mqtt_client_config_t*::*task_t* **task**

FreeRTOS task configuration.

struct *esp_mqtt_client_config_t*::*buffer_t* **buffer**

Buffer size configuration.

struct *esp_mqtt_client_config_t*::*outbox_config_t* **outbox**

Outbox configuration.

struct **broker_t**

Broker related configuration

Public Members

struct *esp_mqtt_client_config_t*::*broker_t*::*address_t* **address**

Broker address configuration

struct *esp_mqtt_client_config_t*::*broker_t*::*verification_t* **verification**

Security verification of the broker

struct **address_t**

Broker address

- uri have precedence over other fields
- If uri isn't set at least hostname, transport and port should.

Public Members

const char ***uri**

Complete *MQTT* broker URI

const char ***hostname**

Hostname, to set ipv4 pass it as string)

esp_mqtt_transport_t **transport**

Selects transport

const char ***path**

Path in the URI

uint32_t **port**

MQTT server port

struct **verification_t**

Broker identity verification

If fields are not set broker's identity isn't verified. it's recommended to set the options in this struct for security reasons.

Public Members

bool **use_global_ca_store**

Use a global ca_store, look esp-tls documentation for details.

esp_err_t (***crt_bundle_attach**)(void *conf)

Pointer to ESP x509 Certificate Bundle attach function for the usage of certificate bundles. Client only attach the bundle, the clean up must be done by the user.

const char ***certificate**

Certificate data, default is NULL. It's not copied nor freed by the client, user needs to clean up.

size_t **certificate_len**

Length of the buffer pointed to by certificate.

const struct *psk_key_hint* ***psk_hint_key**

Pointer to PSK struct defined in esp_tls.h to enable PSK authentication (as alternative to certificate verification). PSK is enabled only if there are no other ways to verify broker. It's not copied nor freed by the client, user needs to clean up.

bool **skip_cert_common_name_check**

Skip any validation of server certificate CN field, this reduces the security of TLS and makes the *MQTT* client susceptible to MITM attacks

const char ****alpn_protos**

NULL-terminated list of supported application protocols to be used for ALPN.

const char ***common_name**

Pointer to the string containing server certificate common name. If non-NULL, server certificate CN must match this name, If NULL, server certificate CN must match hostname. This is ignored if skip_cert_common_name_check=true. It's not copied nor freed by the client, user needs to clean up.

struct **buffer_t**

Client buffer size configuration

Client have two buffers for input and output respectivelly.

Public Members

int **size**

size of *MQTT* send/receive buffer

int **out_size**

size of *MQTT* output buffer. If not defined, defaults to the size defined by `buffer_size`

struct **credentials_t**

Client related credentials for authentication.

Public Members

const char ***username**

MQTT username

const char ***client_id**

Set *MQTT* client identifier. Ignored if set_null_client_id == true If NULL set the default client id. Default client id is `ESP32_CHIPID%` where `CHIPID%` are last 3 bytes of MAC address in hex format

bool **set_null_client_id**

Selects a NULL client id

struct *esp_mqtt_client_config_t*::*credentials_t*::*authentication_t* **authentication**

Client authentication

struct **authentication_t**

Client authentication

Fields related to client authentication by broker

For mutual authentication using TLS, user could select certificate and key, secure element or digital signature peripheral if available.

Public Members

const char ***password**

MQTT password

const char ***certificate**

Certificate for ssl mutual authentication, not required if mutual authentication is not needed. Must be provided with `key`. It's not copied nor freed by the client, user needs to clean up.

size_t **certificate_len**

Length of the buffer pointed to by certificate.

const char ***key**

Private key for SSL mutual authentication, not required if mutual authentication is not needed. If it is not NULL, also `certificate` has to be provided. It's not copied nor freed by the client, user needs to clean up.

size_t **key_len**

Length of the buffer pointed to by key.

const char ***key_password**

Client key decryption password, not PEM nor DER, if provided `key_password_len` must be correctly set.

int **key_password_len**

Length of the password pointed to by `key_password`

bool **use_secure_element**

Enable secure element, available in ESP32-ROOM-32SE, for SSL connection

void ***ds_data**

Carrier of handle for digital signature parameters, digital signature peripheral is available in some Espressif devices. It's not copied nor freed by the client, user needs to clean up.

struct **network_t**

Network related configuration

Public Members

int **reconnect_timeout_ms**

Reconnect to the broker after this value in miliseconds if auto reconnect is not disabled (defaults to 10s)

int **timeout_ms**

Abort network operation if it is not completed after this value, in milliseconds (defaults to 10s).

int **refresh_connection_after_ms**

Refresh connection after this value (in milliseconds)

bool **disable_auto_reconnect**

Client will reconnect to server (when errors/disconnect). Set `disable_auto_reconnect=true` to disable

esp_transport_handle_t **transport**

Custom transport handle to use. Warning: The transport should be valid during the client lifetime and is destroyed when esp_mqtt_client_destroy is called.

struct ifreq ***if_name**

The name of interface for data to go through. Use the default interface without setting

struct **outbox_config_t**

Client outbox configuration options.

Public Members

uint64_t **limit**

Size limit for the outbox in bytes.

struct **session_t**

MQTT Session related configuration

Public Members

struct *esp_mqtt_client_config_t*::*session_t*::*last_will_t* **last_will**

Last will configuration

bool **disable_clean_session**

MQTT clean session, default clean_session is true

int **keepalive**

MQTT keepalive, default is 120 seconds When configuring this value, keep in mind that the client attempts to communicate with the broker at half the interval that is actually set. This conservative approach allows for more attempts before the broker's timeout occurs

bool **disable_keepalive**

Set `disable_keepalive=true` to turn off keep-alive mechanism, keepalive is active by default. Note: setting the config value `keepalive` to `0` doesn't disable keepalive feature, but uses a default keepalive period

esp_mqtt_protocol_ver_t **protocol_ver**

MQTT protocol version used for connection.

int **message_retransmit_timeout**

timeout for retransmitting of failed packet

struct **last_will_t**

Last Will and Testament message configuration.

Public Members

const char ***topic**

LWT (Last Will and Testament) message topic

const char ***msg**

LWT message, may be NULL terminated

int **msg_len**

LWT message length, if msg isn't NULL terminated must have the correct length

int **qos**

LWT message QoS

int **retain**

LWT retained message flag

struct **task_t**

Client task configuration

Public Members

int **priority**

MQTT task priority

int **stack_size**

MQTT task stack size

struct **topic_t**

Topic definition struct

Public Members

const char ***filter**

Topic filter to subscribe

int **qos**

Max QoS level of the subscription

Macros

MQTT_ERROR_TYPE_ESP_TLS

MQTT_ERROR_TYPE_TCP_TRANSPORT error type hold all sorts of transport layer errors, including ESP-TLS error, but in the past only the errors from MQTT_ERROR_TYPE_ESP_TLS layer were reported, so the ESP-TLS error type is re-defined here for backward compatibility

esp_mqtt_client_subscribe(client_handle, topic_type, qos_or_size)

Convenience macro to select subscribe function to use.

Notes:

- Usage of `esp_mqtt_client_subscribe_single` is the same as previous esp_mqtt_client_subscribe, refer to it for details.

参数

- **client_handle** -- *MQTT* client handle
- **topic_type** -- Needs to be char* for single subscription or `esp_mqtt_topic_t` for multiple topics
- **qos_or_size** -- It's either a qos when subscribing to a single topic or the size of the subscription array when subscribing to multiple topics.

返回 message_id of the subscribe message on success -1 on failure -2 in case of full outbox.

Type Definitions

typedef struct esp_mqtt_client ***esp_mqtt_client_handle_t**

typedef enum *esp_mqtt_event_id_t* **esp_mqtt_event_id_t**

MQTT event types.

User event handler receives context data in *esp_mqtt_event_t* structure with

- client - *MQTT* client handle
- various other data depending on event type

typedef enum *esp_mqtt_connect_return_code_t* **esp_mqtt_connect_return_code_t**

MQTT connection error codes propagated via ERROR event

typedef enum *esp_mqtt_error_type_t* **esp_mqtt_error_type_t**

MQTT connection error codes propagated via ERROR event

typedef enum *esp_mqtt_transport_t* **esp_mqtt_transport_t**

typedef enum *esp_mqtt_protocol_ver_t* **esp_mqtt_protocol_ver_t**

MQTT protocol version used for connection

typedef struct *esp_mqtt_error_codes* **esp_mqtt_error_codes_t**

MQTT error code structure to be passed as a contextual information into ERROR event

Important: This structure extends `esp_tls_last_error` error structure and is backward compatible with it (so might be down-casted and treated as `esp_tls_last_error` error, but recommended to update applications if used this way previously)

Use this structure directly checking error_type first and then appropriate error code depending on the source of the error:

| error_type | related member variables | note | | MQTT_ERROR_TYPE_TCP_TRANSPORT | esp_tls_last_esp_err, esp_tls_stack_err, esp_tls_cert_verify_flags, sock_errno | Error reported from tcp_transport/esp-tls | | MQTT_ERROR_TYPE_CONNECTION_REFUSED | connect_return_code | Internal error reported from *MQTT* broker on connection |

typedef struct *esp_mqtt_event_t* **esp_mqtt_event_t**

MQTT event configuration structure

typedef *esp_mqtt_event_t* ***esp_mqtt_event_handle_t**

typedef struct *esp_mqtt_client_config_t* **esp_mqtt_client_config_t**

MQTT client configuration structure

- Default values can be set via menuconfig
- All certificates and key data could be passed in PEM or DER format. PEM format must have a terminating NULL character and the related len field set to 0. DER format requires a related len field set to the correct length.

typedef struct *topic_t* **esp_mqtt_topic_t**

Topic definition struct

Enumerations

enum **esp_mqtt_event_id_t**

MQTT event types.

User event handler receives context data in `esp_mqtt_event_t` structure with

- client - *MQTT* client handle
- various other data depending on event type

Values:

enumerator **MQTT_EVENT_ANY**

enumerator **MQTT_EVENT_ERROR**

on error event, additional context: connection return code, error handle from esp_tls (if supported)

enumerator **MQTT_EVENT_CONNECTED**

connected event, additional context: session_present flag

enumerator **MQTT_EVENT_DISCONNECTED**

disconnected event

enumerator **MQTT_EVENT_SUBSCRIBED**

subscribed event, additional context:

- msg_id message id
- error_handle `error_type` in case subscribing failed
- data pointer to broker response, check for errors.
- data_len length of the data for this event

enumerator **MQTT_EVENT_UNSUBSCRIBED**

unsubscribed event, additional context: msg_id

enumerator **MQTT_EVENT_PUBLISHED**

published event, additional context: msg_id

enumerator **MQTT_EVENT_DATA**

data event, additional context:

- msg_id message id
- topic pointer to the received topic
- topic_len length of the topic
- data pointer to the received data
- data_len length of the data for this event
- current_data_offset offset of the current data for this event
- total_data_len total length of the data received
- retain retain flag of the message
- qos QoS level of the message
- dup dup flag of the message Note: Multiple MQTT_EVENT_DATA could be fired for one message, if it is longer than internal buffer. In that case only first event contains topic pointer and length, other contain data only with current data length and current data offset updating.

enumerator **MQTT_EVENT_BEFORE_CONNECT**

The event occurs before connecting

enumerator **MQTT_EVENT_DELETED**

Notification on delete of one message from the internal outbox, if the message couldn't have been sent and acknowledged before expiring defined in OUTBOX_EXPIRED_TIMEOUT_MS. (events are not posted upon deletion of successfully acknowledged messages)

- This event id is posted only if MQTT_REPORT_DELETED_MESSAGES==1
- Additional context: msg_id (id of the deleted message).

enumerator **MQTT_USER_EVENT**

Custom event used to queue tasks into mqtt event handler All fields from the *esp_mqtt_event_t* type could be used to pass an additional context data to the handler.

enum **esp_mqtt_connect_return_code_t**

MQTT connection error codes propagated via ERROR event

Values:

enumerator **MQTT_CONNECTION_ACCEPTED**

Connection accepted

enumerator **MQTT_CONNECTION_REFUSE_PROTOCOL**

MQTT connection refused reason: Wrong protocol

enumerator **MQTT_CONNECTION_REFUSE_ID_REJECTED**

MQTT connection refused reason: ID rejected

enumerator **MQTT_CONNECTION_REFUSE_SERVER_UNAVAILABLE**

MQTT connection refused reason: Server unavailable

enumerator **MQTT_CONNECTION_REFUSE_BAD_USERNAME**

MQTT connection refused reason: Wrong user

enumerator **MQTT_CONNECTION_REFUSE_NOT_AUTHORIZED**

MQTT connection refused reason: Wrong username or password

enum **esp_mqtt_error_type_t**

MQTT connection error codes propagated via ERROR event

Values:

enumerator **MQTT_ERROR_TYPE_NONE**

enumerator **MQTT_ERROR_TYPE_TCP_TRANSPORT**

enumerator **MQTT_ERROR_TYPE_CONNECTION_REFUSED**

enumerator **MQTT_ERROR_TYPE_SUBSCRIBE_FAILED**

enum **esp_mqtt_transport_t**

Values:

enumerator **MQTT_TRANSPORT_UNKNOWN**

enumerator **MQTT_TRANSPORT_OVER_TCP**

MQTT over TCP, using scheme: `MQTT`

enumerator **MQTT_TRANSPORT_OVER_SSL**

MQTT over SSL, using scheme: `MQTTS`

enumerator **MQTT_TRANSPORT_OVER_WS**

MQTT over Websocket, using scheme:: `ws`

enumerator **MQTT_TRANSPORT_OVER_WSS**

MQTT over Websocket Secure, using scheme: `wss`

enum **esp_mqtt_protocol_ver_t**

MQTT protocol version used for connection

Values:

enumerator **MQTT_PROTOCOL_UNDEFINED**

enumerator **MQTT_PROTOCOL_V_3_1**

enumerator **MQTT_PROTOCOL_V_3_1_1**

enumerator **MQTT_PROTOCOL_V_5**

2.2.4 ESP-TLS

概述

ESP-TLS 组件提供简化 API 接口，用于访问常用 TLS 功能，支持如 CA 认证验证、SNI、ALPN 协商和非阻塞连接等常见场景，相关配置可在数据结构体 `esp_tls_cfg_t` 中指定。配置完成后，使用以下 API 进行 TLS 通信：

- *`esp_tls_init()`*：初始化 TLS 连接句柄。
- *`esp_tls_conn_new_sync()`*：开启新的阻塞式 TLS 连接。
- *`esp_tls_conn_new_async()`*：开启新的非阻塞式 TLS 连接。
- *`esp_tls_conn_read()`*：读取 TLS 层之上的应用数据。
- *`esp_tls_conn_write()`*：将应用数据写入 TLS 连接。
- *`esp_tls_conn_destroy()`*：释放连接。

任何应用层协议，如 HTTP1、HTTP2 等，均可调用 ESP-TLS 组件接口实现。

应用示例

使用 ESP-TLS 建立安全套接字连接的 HTTPS 简单示例，请参阅 protocols/https_request。

ESP-TLS 组件的树形结构

```
├── esp_tls.c
├── esp_tls.h
├── esp_tls_mbedtls.c
├── esp_tls_wolfssl.c
└── private_include
    ├── esp_tls_mbedtls.h
    └── esp_tls_wolfssl.h
```

ESP-TLS 组件文件 esp-tls/esp_tls.h 包含该组件的公共 API 头文件。在 ESP-TLS 组件内部，为了实现安全会话功能，会使用 MbedTLS 和 WolfSSL 两个 SSL/TLS 库中的其中一个进行安全会话的建立，与 MbedTLS 相关的 API 存放在 esp-tls/private_include/esp_tls_mbedtls.h，而与 WolfSSL 相关的 API 存放在 esp-tls/private_include/esp_tls_wolfssl.h。

TLS 服务器验证

ESP-TLS 在客户端提供了多种验证 TLS 服务器的选项，如验证对端服务器的服务器证书、或使用预共享密钥验证服务器。用户应在*esp_tls_cfg_t* 结构体中选择以下任一选项完成 TLS 服务器验证，若未做选择，则客户端默认在 TLS 连接创建时，会返回错误。

- **cacert_buf** 和 **cacert_bytes**：以缓冲区的形式向*esp_tls_cfg_t* 结构体提供 CA 证书，ESP-TLS 将使用缓冲区中的 CA 证书验证服务器。注意，须在*esp_tls_cfg_t* 结构体中设置以下变量：
  - `cacert_buf` - 指针，指向包含 CA 证书的缓冲区。
  - `cacert_bytes` - CA 证书大小（以字节为单位）。
- **use_global_ca_store**：`global_ca_store` 可一次性完成初始化及设置，并用于验证 ESP-TLS 连接的服务器，注意需要在这些服务器各自的*esp_tls_cfg_t* 结构体中设置 `use_global_ca_store = true`。有关初始化和设置 `global_ca_store` 的不同 API，请参阅文末的 API 参考。
- **crt_bundle_attach**：ESP x509 证书包 API 提供了便捷的服务器验证方法，即打包一组自定义的 x509 根证书，用于 TLS 服务器验证，详情请参阅*ESP x509 证书包*。
- **psk_hint_key**：要使用预共享密钥验证服务器，必须在 ESP-TLS menuconfig 中启用*CONFIG_ESP_TLS_PSK_VERIFICATION*，然后向结构体*esp_tls_cfg_t* 提供指向 PSK 提示和密钥的指针。若未选择有关服务器验证的其他选项，ESP-TLS 将仅用 PSK 验证服务器。
- **跳过服务器验证**：该选项并不安全，仅供测试使用。在 ESP-TLS menuconfig 中启用*CONFIG_ESP_TLS_INSECURE* 和*CONFIG_ESP_TLS_SKIP_SERVER_CERT_VERIFY* 可启用该选项，此时，若未在*esp_tls_cfg_t* 结构体选择其他服务器验证选项，ESP-TLS 将默认跳过服务器验证。

> **警告：** 启用 **跳过服务器验证**选项存在潜在风险，若未通过 API 或 `ca_store` 等其他机制提供服务器证书，可能导致设备与伪造身份的服务器建立 TLS 连接。

ESP-TLS 服务器证书选择回调

使用 MbedTLS 协议栈时，ESP-TLS 组件支持设置服务器证书选择回调函数。此时，在服务器握手期间可选择使用哪个服务器证书，该回调可获取客户端发送的"Client Hello" 消息中提供的 TLS 扩展（ALPN、SPI 等），并基于此选择传输哪个服务器证书给客户端。要启用此功能，请在 ESP-TLS menuconfig 中启用*CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK*。

证书选择回调可在结构体*esp_tls_cfg_t* 中配置，具体如下：

```
int cert_selection_callback(mbedtls_ssl_context *ssl)
{
    /* 回调应执行的代码 */
    return 0;
}

esp_tls_cfg_t cfg = {
    cert_select_cb = cert_section_callback,
};
```

底层 SSL/TLS 库选择

ESP-TLS 组件支持以 MbedTLS 或 WolfSSL 作为其底层 SSL/TLS 库，默认仅使用 MbedTLS，WolfSSL 的 SSL/TLS 库可在 https://github.com/espressif/esp-wolfssl 上公开获取，该仓库提供二进制格式的 WolfSSL 组件，并提供了一些示例帮助用户了解相关 API。有关许可证和其他选项，请参阅仓库的 `README.md` 文件。下文介绍了在工程中使用 WolfSSL 的具体流程。

备注： 库选项位于 ESP-TLS 内部，因此切换库不会更改工程的 ESP-TLS 特定代码。

在 ESP-IDF 使用 WolfSSL

要在工程中使用 WolfSSL，可采取以下两种方式：

1) 使用以下三行命令，将 WolfSSL 作为组件直接添加到工程中：

```
（首先用 cd 命令进入工程目录）
mkdir components
cd components
git clone https://github.com/espressif/esp-wolfssl.git
```

2) 将 WolfSSL 作为额外组件添加到工程中。

- 使用以下命令下载 WolfSSL：

```
git clone https://github.com/espressif/esp-wolfssl.git
```

- 参照 wolfssl/examples 示例，在工程的 `CMakeLists.txt` 文件中设置 `EXTRA_COMPONENT_DIRS`，从而在 ESP-IDF 中包含 ESP-WolfSSL，详情请参阅*构建系统* 中的*可选的项目变量* 小节。

完成上述步骤后，可以在工程配置菜单中将 WolfSSL 作为底层 SSL/TLS 库，具体步骤如下：

```
idf.py menuconfig > ESP-TLS > SSL/TLS Library > Mbedtls/Wolfssl
```

MbedTLS 与 WolfSSL 对比

下表是在使用 WolfSSL 和 MbedTLS 两种 SSL/TLS 库，并将所有相关配置设置为默认值时，运行具有服务器身份验证的 protocols/https_request 示例的比较结果。对于 MbedTLS，IN_CONTENT 长度和 OUT_CONTENT 长度分别设置为 16384 字节和 4096 字节。

属性	WolfSSL	MbedTLS
总消耗堆空间	~ 19 KB	~ 37 KB
任务栈使用	~ 2.2 KB	~ 3.6 KB
二进制文件大小	~ 858 KB	~ 736 KB

备注： 若配置选项不同或相应库的版本不同，得到的值可能与上表不同。

ESP-TLS 的数字签名

ESP-TLS 支持在 ESP32-P4 中使用数字签名 (DS)，但只有当 ESP-TLS 以 MbedTLS（默认协议栈）为底层 SSL/TLS 协议栈时，才支持使用 TLS 的数字签名。有关数字签名的详细信息，请参阅*数字签名 (DS)*。有关数字签名的技术细节（例如私钥参数计算），请参阅 **ESP32-P4 技术参考手册 > 数字签名 (DS)** [PDF]。在使用数字签名前，应预先配置数字签名外设，请参阅*TLS 连接所需的 DS 外设配置*。

数字签名外设必须用所需的加密私钥参数初始化，相应参数在配置数字签名外设时获取。具备所需的数字签名上下文，即数字签名参数时，ESP-TLS 会在内部初始化数字签名外设。要将数字签名上下文传递给 ESP-TLS 上下文，请参阅以下代码段。注意，在删除 TLS 连接之前，不应释放传递给 ESP-TLS 上下文的数字签名上下文。

```
#include "esp_tls.h"
esp_ds_data_ctx_t *ds_ctx;
/* 使用加密的私钥参数初始化 ds_ctx，这类参数可以从 nvs␣
↪中读取，或由应用程序代码提供 */
```

（下页继续）

(续上页)

```
esp_tls_cfg_t cfg = {
    .clientcert_buf = /* 客户端证书 */,
    .clientcert_bytes = /* 客户端证书长度 */,
    /* 其他配置选项 */
    .ds_data = (void *)ds_ctx,
};
```

备注： 当使用数字签名进行 TLS 连接时，除其他必要参数外，仅需提供客户端证书 (clientcert_buf) 和数字签名参数 (ds_data)，此时可将客户端密钥 (clientkey_buf) 设置为 NULL。

- 使用数字签名外设进行双向认证的示例请参阅 SSL 双向认证，该示例使用 ESP-TLS 实现 TLS 连接。

TLS 加密套件

ESP-TLS 支持在客户端模式下设置加密套件列表，TLS 密码套件列表用于向服务器传递所支持的密码套件信息，用户可以根据自己需求增减加密套件，且适用于任何 TLS 协议栈配置。如果服务器支持列表中的任一密码套件，则 TLS 连接成功，反之连接失败。

连接客户端时，在*esp_tls_cfg_t* 结构体中设置 ciphersuites_list 的步骤如下：

```
/* 加密套件列表必须以 0 结尾，并且在整个 TLS 连接期间，加密套件的内存地址空间有效
↪*/
static const int ciphersuites_list[] = {MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_
↪SHA384, MBEDTLS_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, 0};
esp_tls_cfg_t cfg = {
    .ciphersuites_list = ciphersuites_list,
};
```

ESP-TLS 不会检查 ciphersuites_list 的有效性，因此需调用*esp_tls_get_ciphersuites_list()* 获取 TLS 协议栈中支持的加密套件列表，并检查设置的加密套件是否在支持的加密套件列表中。

备注： 此功能仅在 MbedTLS 协议栈中有效。

TLS 协议版本

ESP-TLS 能够为 TLS 连接设置相应的 TLS 协议版本，指定版本将用于建立专用 TLS 连接。也就是说，在运行时不同的 TLS 连接可以配置到 TLS 1.2、TLS 1.3 等不同协议版本。

备注： 目前，仅在 MbedTLS 作为 ESP-TLS 的底层 SSL/TLS 协议栈时支持此功能。

要在 ESP-TLS 中设置 TLS 协议版本，请设置 esp_tls_cfg_t::tls_version，从*esp_tls_proto_ver_t* 中选择所需版本。如未指定协议版本字段，将默认根据服务器要求建立 TLS 连接。

ESP-TLS 连接的协议版本可按如下方式配置：

```
#include "esp_tls.h"
esp_tls_cfg_t cfg = {
    .tls_version = ESP_TLS_VER_TLS_1_2,
};
```

API 参考

Header File

- components/esp-tls/esp_tls.h
- This header file can be included with:

```
#include "esp_tls.h"
```

- This header file is a part of the API provided by the `esp-tls` component. To declare that your component depends on `esp-tls`, add the following to your CMakeLists.txt:

```
REQUIRES esp-tls
```

or

```
PRIV_REQUIRES esp-tls
```

Functions

esp_err_t **esp_tls_cfg_server_session_tickets_init** (*esp_tls_cfg_server_t* *cfg)

Initialize the server side TLS session ticket context.

This function initializes the server side tls session ticket context which holds all necessary data structures to enable tls session tickets according to RFC5077. Use esp_tls_cfg_server_session_tickets_free to free the data.

参数 **cfg** -- **[in]** server configuration as esp_tls_cfg_server_t

返回 ESP_OK if setup succeeded ESP_ERR_INVALID_ARG if context is already initialized ESP_ERR_NO_MEM if memory allocation failed ESP_ERR_NOT_SUPPORTED if session tickets are not available due to build configuration ESP_FAIL if setup failed

void **esp_tls_cfg_server_session_tickets_free** (*esp_tls_cfg_server_t* *cfg)

Free the server side TLS session ticket context.

参数 **cfg** -- server configuration as esp_tls_cfg_server_t

esp_tls_t ***esp_tls_init** (void)

Create TLS connection.

This function allocates and initializes esp-tls structure handle.

返回 tls Pointer to esp-tls as esp-tls handle if successfully initialized, NULL if allocation error

esp_tls_t ***esp_tls_conn_http_new** (const char *url, const *esp_tls_cfg_t* *cfg)

Create a new blocking TLS/SSL connection with a given "HTTP" url.

Note: This API is present for backward compatibility reasons. Alternative function with the same functionality is `esp_tls_conn_http_new_sync` (and its asynchronous version `esp_tls_conn_http_new_async`)

参数

- **url** -- **[in]** url of host.
- **cfg** -- **[in]** TLS configuration as esp_tls_cfg_t. If you wish to open non-TLS connection, keep this NULL. For TLS connection, a pass pointer to 'esp_tls_cfg_t'. At a minimum, this structure should be zero-initialized.

返回 pointer to esp_tls_t, or NULL if connection couldn't be opened.

int **esp_tls_conn_new_sync** (const char *hostname, int hostlen, int port, const *esp_tls_cfg_t* *cfg, *esp_tls_t* *tls)

Create a new blocking TLS/SSL connection.

This function establishes a TLS/SSL connection with the specified host in blocking manner.

参数

- **hostname** -- **[in]** Hostname of the host.
- **hostlen** -- **[in]** Length of hostname.

- **port** -- **[in]** Port number of the host.
- **cfg** -- **[in]** TLS configuration as esp_tls_cfg_t. If you wish to open non-TLS connection, keep this NULL. For TLS connection, a pass pointer to esp_tls_cfg_t. At a minimum, this structure should be zero-initialized.
- **tls** -- **[in]** Pointer to esp-tls as esp-tls handle.

返回

- -1 If connection establishment fails.
- 1 If connection establishment is successful.
- 0 If connection state is in progress.

int **esp_tls_conn_http_new_sync**(const char *url, const *esp_tls_cfg_t* *cfg, *esp_tls_t* *tls)

Create a new blocking TLS/SSL connection with a given "HTTP" url.

The behaviour is same as esp_tls_conn_new_sync() API. However this API accepts host's url.

参数

- **url** -- **[in]** url of host.
- **cfg** -- **[in]** TLS configuration as esp_tls_cfg_t. If you wish to open non-TLS connection, keep this NULL. For TLS connection, a pass pointer to 'esp_tls_cfg_t'. At a minimum, this structure should be zero-initialized.
- **tls** -- **[in]** Pointer to esp-tls as esp-tls handle.

返回

- -1 If connection establishment fails.
- 1 If connection establishment is successful.
- 0 If connection state is in progress.

int **esp_tls_conn_new_async**(const char *hostname, int hostlen, int port, const *esp_tls_cfg_t* *cfg, *esp_tls_t* *tls)

Create a new non-blocking TLS/SSL connection.

This function initiates a non-blocking TLS/SSL connection with the specified host, but due to its non-blocking nature, it doesn't wait for the connection to get established.

参数

- **hostname** -- **[in]** Hostname of the host.
- **hostlen** -- **[in]** Length of hostname.
- **port** -- **[in]** Port number of the host.
- **cfg** -- **[in]** TLS configuration as esp_tls_cfg_t. `non_block` member of this structure should be set to be true.
- **tls** -- **[in]** pointer to esp-tls as esp-tls handle.

返回

- -1 If connection establishment fails.
- 0 If connection establishment is in progress.
- 1 If connection establishment is successful.

int **esp_tls_conn_http_new_async**(const char *url, const *esp_tls_cfg_t* *cfg, *esp_tls_t* *tls)

Create a new non-blocking TLS/SSL connection with a given "HTTP" url.

The behaviour is same as esp_tls_conn_new_async() API. However this API accepts host's url.

参数

- **url** -- **[in]** url of host.
- **cfg** -- **[in]** TLS configuration as esp_tls_cfg_t.
- **tls** -- **[in]** pointer to esp-tls as esp-tls handle.

返回

- -1 If connection establishment fails.
- 0 If connection establishment is in progress.
- 1 If connection establishment is successful.

ssize_t **esp_tls_conn_write**(*esp_tls_t* *tls, const void *data, size_t datalen)

Write from buffer 'data' into specified tls connection.

参数

- **tls** -- **[in]** pointer to esp-tls as esp-tls handle.
- **data** -- **[in]** Buffer from which data will be written.
- **datalen** -- **[in]** Length of data buffer.

返回

- >=0 if write operation was successful, the return value is the number of bytes actually written to the TLS/SSL connection.
- <0 if write operation was not successful, because either an error occured or an action must be taken by the calling process.
- ESP_TLS_ERR_SSL_WANT_READ/ ESP_TLS_ERR_SSL_WANT_WRITE. if the handshake is incomplete and waiting for data to be available for reading. In this case this functions needs to be called again when the underlying transport is ready for operation.

ssize_t **esp_tls_conn_read**(*esp_tls_t* *tls, void *data, size_t datalen)

Read from specified tls connection into the buffer 'data'.

参数

- **tls** -- **[in]** pointer to esp-tls as esp-tls handle.
- **data** -- **[in]** Buffer to hold read data.
- **datalen** -- **[in]** Length of data buffer.

返回

- >0 if read operation was successful, the return value is the number of bytes actually read from the TLS/SSL connection.
- 0 if read operation was not successful. The underlying connection was closed.
- <0 if read operation was not successful, because either an error occured or an action must be taken by the calling process.

int **esp_tls_conn_destroy**(*esp_tls_t* *tls)

Close the TLS/SSL connection and free any allocated resources.

This function should be called to close each tls connection opened with esp_tls_conn_new_sync() (or esp_tls_conn_http_new_sync()) and esp_tls_conn_new_async() (or esp_tls_conn_http_new_async()) APIs.

参数 **tls** -- **[in]** pointer to esp-tls as esp-tls handle.

返回 - 0 on success

- -1 if socket error or an invalid argument

ssize_t **esp_tls_get_bytes_avail**(*esp_tls_t* *tls)

Return the number of application data bytes remaining to be read from the current record.

This API is a wrapper over mbedtls's mbedtls_ssl_get_bytes_avail() API.

参数 **tls** -- **[in]** pointer to esp-tls as esp-tls handle.

返回

- -1 in case of invalid arg
- bytes available in the application data record read buffer

esp_err_t **esp_tls_get_conn_sockfd**(*esp_tls_t* *tls, int *sockfd)

Returns the connection socket file descriptor from esp_tls session.

参数

- **tls** -- **[in]** handle to esp_tls context
- **sockfd** -- **[out]** int pointer to sockfd value.

返回 - ESP_OK on success and value of sockfd will be updated with socket file descriptor for connection

- ESP_ERR_INVALID_ARG if (tls == NULL || sockfd == NULL)

esp_err_t **esp_tls_set_conn_sockfd**(*esp_tls_t* *tls, int sockfd)

Sets the connection socket file descriptor for the esp_tls session.

参数

- **tls** -- **[in]** handle to esp_tls context
- **sockfd** -- **[in]** sockfd value to set.

返回 - ESP_OK on success and value of sockfd for the tls connection shall updated withthe provided value
- ESP_ERR_INVALID_ARG if (tls == NULL || sockfd < 0)

esp_err_t **esp_tls_get_conn_state**(*esp_tls_t* *tls, *esp_tls_conn_state_t* *conn_state)

Gets the connection state for the esp_tls session.

参数
- **tls** -- **[in]** handle to esp_tls context
- **conn_state** -- **[out]** pointer to the connection state value.

返回 - ESP_OK on success and value of sockfd for the tls connection shall updated withthe provided value
- ESP_ERR_INVALID_ARG (Invalid arguments)

esp_err_t **esp_tls_set_conn_state**(*esp_tls_t* *tls, *esp_tls_conn_state_t* conn_state)

Sets the connection state for the esp_tls session.

参数
- **tls** -- **[in]** handle to esp_tls context
- **conn_state** -- **[in]** connection state value to set.

返回 - ESP_OK on success and value of sockfd for the tls connection shall updated withthe provided value
- ESP_ERR_INVALID_ARG (Invalid arguments)

void ***esp_tls_get_ssl_context**(*esp_tls_t* *tls)

Returns the ssl context.

参数 **tls** -- **[in]** handle to esp_tls context

返回 - ssl_ctx pointer to ssl context of underlying TLS layer on success
- NULL in case of error

esp_err_t **esp_tls_init_global_ca_store**(void)

Create a global CA store, initially empty.

This function should be called if the application wants to use the same CA store for multiple connections. This function initialises the global CA store which can be then set by calling esp_tls_set_global_ca_store(). To be effective, this function must be called before any call to esp_tls_set_global_ca_store().

返回
- ESP_OK if creating global CA store was successful.
- ESP_ERR_NO_MEM if an error occured when allocating the mbedTLS resources.

esp_err_t **esp_tls_set_global_ca_store**(const unsigned char *cacert_pem_buf, const unsigned int cacert_pem_bytes)

Set the global CA store with the buffer provided in pem format.

This function should be called if the application wants to set the global CA store for multiple connections i.e. to add the certificates in the provided buffer to the certificate chain. This function implicitly calls esp_tls_init_global_ca_store() if it has not already been called. The application must call this function before calling esp_tls_conn_new().

参数
- **cacert_pem_buf** -- **[in]** Buffer which has certificates in pem format. This buffer is used for creating a global CA store, which can be used by other tls connections.
- **cacert_pem_bytes** -- **[in]** Length of the buffer.

返回
- ESP_OK if adding certificates was successful.
- Other if an error occured or an action must be taken by the calling process.

void **esp_tls_free_global_ca_store**(void)

Free the global CA store currently being used.

The memory being used by the global CA store to store all the parsed certificates is freed up. The application can call this API if it no longer needs the global CA store.

esp_err_t **esp_tls_get_and_clear_last_error**(*esp_tls_error_handle_t* h, int *esp_tls_code, int *esp_tls_flags)

Returns last error in esp_tls with detailed mbedtls related error codes. The error information is cleared internally upon return.

参数

- **h** -- **[in]** esp-tls error handle.
- **esp_tls_code** -- **[out]** last error code returned from mbedtls api (set to zero if none) This pointer could be NULL if caller does not care about esp_tls_code
- **esp_tls_flags** -- **[out]** last certification verification flags (set to zero if none) This pointer could be NULL if caller does not care about esp_tls_code

返回

- ESP_ERR_INVALID_STATE if invalid parameters
- ESP_OK (0) if no error occurred
- specific error code (based on ESP_ERR_ESP_TLS_BASE) otherwise

esp_err_t **esp_tls_get_and_clear_error_type**(*esp_tls_error_handle_t* h, *esp_tls_error_type_t* err_type, int *error_code)

Returns the last error captured in esp_tls of a specific type The error information is cleared internally upon return.

参数

- **h** -- **[in]** esp-tls error handle.
- **err_type** -- **[in]** specific error type
- **error_code** -- **[out]** last error code returned from mbedtls api (set to zero if none) This pointer could be NULL if caller does not care about esp_tls_code

返回

- ESP_ERR_INVALID_STATE if invalid parameters
- ESP_OK if a valid error returned and was cleared

esp_err_t **esp_tls_get_error_handle**(*esp_tls_t* *tls, *esp_tls_error_handle_t* *error_handle)

Returns the ESP-TLS error_handle.

参数

- **tls** -- **[in]** handle to esp_tls context
- **error_handle** -- **[out]** pointer to the error handle.

返回

- ESP_OK on success and error_handle will be updated with the ESP-TLS error handle.
- ESP_ERR_INVALID_ARG if (tls == NULL || error_handle == NULL)

mbedtls_x509_crt ***esp_tls_get_global_ca_store**(void)

Get the pointer to the global CA store currently being used.

The application must first call esp_tls_set_global_ca_store(). Then the same CA store could be used by the application for APIs other than esp_tls.

备注： Modifying the pointer might cause a failure in verifying the certificates.

返回

- Pointer to the global CA store currently being used if successful.
- NULL if there is no global CA store set.

const int ***esp_tls_get_ciphersuites_list**(void)

Get supported TLS ciphersuites list.

See https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4 for the list of ciphersuites

返回 Pointer to a zero-terminated array of IANA identifiers of TLS ciphersuites.

int **esp_tls_server_session_create**(*esp_tls_cfg_server_t* *cfg, int sockfd, *esp_tls_t* *tls)

Create TLS/SSL server session.

This function creates a TLS/SSL server context for already accepted client connection and performs TLS/SSL handshake with the client

参数

- **cfg** -- **[in]** Pointer to esp_tls_cfg_server_t
- **sockfd** -- **[in]** FD of accepted connection
- **tls** -- **[out]** Pointer to allocated esp_tls_t

返回

- 0 if successful
- <0 in case of error

void **esp_tls_server_session_delete**(*esp_tls_t* *tls)

Close the server side TLS/SSL connection and free any allocated resources.

This function should be called to close each tls connection opened with esp_tls_server_session_create()

参数 **tls** -- **[in]** pointer to esp_tls_t

esp_err_t **esp_tls_plain_tcp_connect**(const char *host, int hostlen, int port, const *esp_tls_cfg_t* *cfg, *esp_tls_error_handle_t* error_handle, int *sockfd)

Creates a plain TCP connection, returning a valid socket fd on success or an error handle.

参数

- **host** -- **[in]** Hostname of the host.
- **hostlen** -- **[in]** Length of hostname.
- **port** -- **[in]** Port number of the host.
- **cfg** -- **[in]** ESP-TLS configuration as esp_tls_cfg_t.
- **error_handle** -- **[out]** ESP-TLS error handle holding potential errors occurred during connection
- **sockfd** -- **[out]** Socket descriptor if successfully connected on TCP layer

返回 ESP_OK on success ESP_ERR_INVALID_ARG if invalid output parameters ESP-TLS based error codes on failure

Structures

struct **psk_key_hint**

ESP-TLS preshared key and hint structure.

Public Members

const uint8_t ***key**

key in PSK authentication mode in binary format

const size_t **key_size**

length of the key

const char ***hint**

hint in PSK authentication mode in string format

struct **tls_keep_alive_cfg**

esp-tls client session ticket ctx

Keep alive parameters structure

Public Members

bool **keep_alive_enable**

Enable keep-alive timeout

int **keep_alive_idle**

Keep-alive idle time (second)

int **keep_alive_interval**

Keep-alive interval time (second)

int **keep_alive_count**

Keep-alive packet retry send count

struct **esp_tls_cfg**

ESP-TLS configuration parameters.

备注： Note about format of certificates:

- This structure includes certificates of a Certificate Authority, of client or server as well as private keys, which may be of PEM or DER format. In case of PEM format, the buffer must be NULL terminated (with NULL character included in certificate size).
- Certificate Authority's certificate may be a chain of certificates in case of PEM format, but could be only one certificate in case of DER format
- Variables names of certificates and private key buffers and sizes are defined as unions providing backward compatibility for legacy *_pem_buf and *_pem_bytes names which suggested only PEM format was supported. It is encouraged to use generic names such as cacert_buf and cacert_bytes.

Public Members

const char **alpn_protos

Application protocols required for HTTP2. If HTTP2/ALPN support is required, a list of protocols that should be negotiated. The format is length followed by protocol name. For the most common cases the following is ok: const char **alpn_protos = { "h2", NULL };

- where 'h2' is the protocol name

const unsigned char *cacert_buf

Certificate Authority's certificate in a buffer. Format may be PEM or DER, depending on mbedtls-support This buffer should be NULL terminated in case of PEM

const unsigned char *cacert_pem_buf

CA certificate buffer legacy name

unsigned int cacert_bytes

Size of Certificate Authority certificate pointed to by cacert_buf (including NULL-terminator in case of PEM format)

unsigned int cacert_pem_bytes

Size of Certificate Authority certificate legacy name

const unsigned char ***clientcert_buf**

Client certificate in a buffer Format may be PEM or DER, depending on mbedtls-support This buffer should be NULL terminated in case of PEM

const unsigned char ***clientcert_pem_buf**

Client certificate legacy name

unsigned int **clientcert_bytes**

Size of client certificate pointed to by clientcert_pem_buf (including NULL-terminator in case of PEM format)

unsigned int **clientcert_pem_bytes**

Size of client certificate legacy name

const unsigned char ***clientkey_buf**

Client key in a buffer Format may be PEM or DER, depending on mbedtls-support This buffer should be NULL terminated in case of PEM

const unsigned char ***clientkey_pem_buf**

Client key legacy name

unsigned int **clientkey_bytes**

Size of client key pointed to by clientkey_pem_buf (including NULL-terminator in case of PEM format)

unsigned int **clientkey_pem_bytes**

Size of client key legacy name

const unsigned char ***clientkey_password**

Client key decryption password string

unsigned int **clientkey_password_len**

String length of the password pointed to by clientkey_password

bool **use_ecdsa_peripheral**

Use the ECDSA peripheral for the private key operations

uint8_t **ecdsa_key_efuse_blk**

The efuse block where the ECDSA key is stored

bool **non_block**

Configure non-blocking mode. If set to true the underneath socket will be configured in non blocking mode after tls session is established

bool **use_secure_element**

Enable this option to use secure element or atecc608a chip

int **timeout_ms**

Network timeout in milliseconds. Note: If this value is not set, by default the timeout is set to 10 seconds. If you wish that the session should wait indefinitely then please use a larger value e.g., INT32_MAX

bool **use_global_ca_store**

Use a global ca_store for all the connections in which this bool is set.

const char ***common_name**

If non-NULL, server certificate CN must match this name. If NULL, server certificate CN must match hostname.

bool **skip_common_name**

Skip any validation of server certificate CN field

tls_keep_alive_cfg_t ***keep_alive_cfg**

Enable TCP keep-alive timeout for SSL connection

const *psk_hint_key_t* ***psk_hint_key**

Pointer to PSK hint and key. if not NULL (and certificates are NULL) then PSK authentication is enabled with configured setup. Important note: the pointer must be valid for connection

esp_err_t (***crt_bundle_attach**)(void *conf)

Function pointer to esp_crt_bundle_attach. Enables the use of certification bundle for server verification, must be enabled in menuconfig

void ***ds_data**

Pointer for digital signature peripheral context

bool **is_plain_tcp**

Use non-TLS connection: When set to true, the esp-tls uses plain TCP transport rather then TLS/SSL connection. Note, that it is possible to connect using a plain tcp transport directly with esp_tls_plain_tcp_connect() API

struct ifreq ***if_name**

The name of interface for data to go through. Use the default interface without setting

esp_tls_addr_family_t **addr_family**

The address family to use when connecting to a host.

const int ***ciphersuites_list**

Pointer to a zero-terminated array of IANA identifiers of TLS ciphersuites. Please check the list validity by esp_tls_get_ciphersuites_list() API

esp_tls_proto_ver_t **tls_version**

TLS protocol version of the connection, e.g., TLS 1.2, TLS 1.3 (default - no preference)

struct **esp_tls_cfg_server**

ESP-TLS Server configuration parameters.

Public Members

const char ****alpn_protos**

Application protocols required for HTTP2. If HTTP2/ALPN support is required, a list of protocols that should be negotiated. The format is length followed by protocol name. For the most common cases the following is ok: const char **alpn_protos = { ”h2”, NULL };

- where ’h2’ is the protocol name

const unsigned char ***cacert_buf**

Client CA certificate in a buffer. This buffer should be NULL terminated

const unsigned char ***cacert_pem_buf**

Client CA certificate legacy name

unsigned int **cacert_bytes**

Size of client CA certificate pointed to by cacert_pem_buf

unsigned int **cacert_pem_bytes**

Size of client CA certificate legacy name

const unsigned char ***servercert_buf**

Server certificate in a buffer This buffer should be NULL terminated

const unsigned char ***servercert_pem_buf**

Server certificate legacy name

unsigned int **servercert_bytes**

Size of server certificate pointed to by servercert_pem_buf

unsigned int **servercert_pem_bytes**

Size of server certificate legacy name

const unsigned char ***serverkey_buf**

Server key in a buffer This buffer should be NULL terminated

const unsigned char ***serverkey_pem_buf**

Server key legacy name

unsigned int **serverkey_bytes**

Size of server key pointed to by serverkey_pem_buf

unsigned int **serverkey_pem_bytes**

Size of server key legacy name

const unsigned char ***serverkey_password**

Server key decryption password string

unsigned int **serverkey_password_len**

String length of the password pointed to by serverkey_password

bool **use_ecdsa_peripheral**

Use ECDSA peripheral to use private key

uint8_t **ecdsa_key_efuse_blk**

The efuse block where ECDSA key is stored

bool **use_secure_element**

Enable this option to use secure element or atecc608a chip

void ***userdata**

User data to be added to the ssl context. Can be retrieved by callbacks

Type Definitions

typedef enum *esp_tls_conn_state* **esp_tls_conn_state_t**

ESP-TLS Connection State.

typedef enum *esp_tls_role* **esp_tls_role_t**

typedef struct *psk_key_hint* **psk_hint_key_t**

ESP-TLS preshared key and hint structure.

typedef struct *tls_keep_alive_cfg* **tls_keep_alive_cfg_t**

esp-tls client session ticket ctx

Keep alive parameters structure

typedef enum *esp_tls_addr_family* **esp_tls_addr_family_t**

typedef struct *esp_tls_cfg* **esp_tls_cfg_t**

ESP-TLS configuration parameters.

备注： Note about format of certificates:

- This structure includes certificates of a Certificate Authority, of client or server as well as private keys, which may be of PEM or DER format. In case of PEM format, the buffer must be NULL terminated (with NULL character included in certificate size).
- Certificate Authority's certificate may be a chain of certificates in case of PEM format, but could be only one certificate in case of DER format
- Variables names of certificates and private key buffers and sizes are defined as unions providing backward compatibility for legacy *_pem_buf and *_pem_bytes names which suggested only PEM format was supported. It is encouraged to use generic names such as cacert_buf and cacert_bytes.

---

typedef void ***esp_tls_handshake_callback**

typedef struct *esp_tls_cfg_server* **esp_tls_cfg_server_t**

ESP-TLS Server configuration parameters.

typedef struct esp_tls **esp_tls_t**

Enumerations

enum **esp_tls_conn_state**

ESP-TLS Connection State.

Values:

enumerator **ESP_TLS_INIT**

enumerator **ESP_TLS_CONNECTING**

enumerator **ESP_TLS_HANDSHAKE**

enumerator **ESP_TLS_FAIL**

enumerator **ESP_TLS_DONE**

enum **esp_tls_role**

Values:

enumerator **ESP_TLS_CLIENT**

enumerator **ESP_TLS_SERVER**

enum **esp_tls_addr_family**

Values:

enumerator **ESP_TLS_AF_UNSPEC**

Unspecified address family.

enumerator **ESP_TLS_AF_INET**

IPv4 address family.

enumerator **ESP_TLS_AF_INET6**

IPv6 address family.

enum **esp_tls_proto_ver_t**

Values:

enumerator **ESP_TLS_VER_ANY**

enumerator **ESP_TLS_VER_TLS_1_2**

enumerator **ESP_TLS_VER_TLS_1_3**

enumerator **ESP_TLS_VER_TLS_MAX**

Header File

- components/esp-tls/esp_tls_errors.h
- This header file can be included with:

```
#include "esp_tls_errors.h"
```

- This header file is a part of the API provided by the `esp-tls` component. To declare that your component depends on `esp-tls`, add the following to your CMakeLists.txt:

```
REQUIRES esp-tls
```

or

```
PRIV_REQUIRES esp-tls
```

Structures

struct **esp_tls_last_error**

Error structure containing relevant errors in case tls error occurred.

Public Members

esp_err_t **last_error**

error code (based on ESP_ERR_ESP_TLS_BASE) of the last occurred error

int **esp_tls_error_code**

esp_tls error code from last esp_tls failed api

int **esp_tls_flags**

last certification verification flags

Macros

ESP_ERR_ESP_TLS_BASE

Starting number of ESP-TLS error codes

ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME

Error if hostname couldn't be resolved upon tls connection

ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET

Failed to create socket

ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY

Unsupported protocol family

ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST

Failed to connect to host

ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED

failed to set/get socket option

ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT
new connection in esp_tls_low_level_conn connection timeouted

ESP_ERR_ESP_TLS_SE_FAILED

ESP_ERR_ESP_TLS_TCP_CLOSED_FIN

ESP_ERR_MBEDTLS_CERT_PARTLY_OK
mbedtls parse certificates was partly successful

ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_OWN_CERT_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SETUP_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_WRITE_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED
mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED
mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED
mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_TICKET_SETUP_FAILED
mbedtls api returned failed

ESP_ERR_WOLFSSL_SSL_SET_HOSTNAME_FAILED
wolfSSL api returned error

ESP_ERR_WOLFSSL_SSL_CONF_ALPN_PROTOCOLS_FAILED

wolfSSL api returned error

ESP_ERR_WOLFSSL_CERT_VERIFY_SETUP_FAILED

wolfSSL api returned error

ESP_ERR_WOLFSSL_KEY_VERIFY_SETUP_FAILED

wolfSSL api returned error

ESP_ERR_WOLFSSL_SSL_HANDSHAKE_FAILED

wolfSSL api returned failed

ESP_ERR_WOLFSSL_CTX_SETUP_FAILED

wolfSSL api returned failed

ESP_ERR_WOLFSSL_SSL_SETUP_FAILED

wolfSSL api returned failed

ESP_ERR_WOLFSSL_SSL_WRITE_FAILED

wolfSSL api returned failed

ESP_TLS_ERR_SSL_WANT_READ

Definition of errors reported from IO API (potentially non-blocking) in case of error:

- esp_tls_conn_read()
- esp_tls_conn_write()

ESP_TLS_ERR_SSL_WANT_WRITE

ESP_TLS_ERR_SSL_TIMEOUT

Type Definitions

typedef struct *esp_tls_last_error* ***esp_tls_error_handle_t**

typedef struct *esp_tls_last_error* **esp_tls_last_error_t**

Error structure containing relevant errors in case tls error occurred.

Enumerations

enum **esp_tls_error_type_t**

Definition of different types/sources of error codes reported from different components

Values:

enumerator **ESP_TLS_ERR_TYPE_UNKNOWN**

enumerator **ESP_TLS_ERR_TYPE_SYSTEM**

System error –errno

enumerator **ESP_TLS_ERR_TYPE_MBEDTLS**

Error code from mbedTLS library

enumerator **ESP_TLS_ERR_TYPE_MBEDTLS_CERT_FLAGS**

Certificate flags defined in mbedTLS

enumerator **ESP_TLS_ERR_TYPE_ESP**

ESP-IDF error type –esp_err_t

enumerator **ESP_TLS_ERR_TYPE_WOLFSSL**

Error code from wolfSSL library

enumerator **ESP_TLS_ERR_TYPE_WOLFSSL_CERT_FLAGS**

Certificate flags defined in wolfSSL

enumerator **ESP_TLS_ERR_TYPE_MAX**

Last err type –invalid entry

2.2.5 ESP HTTP 客户端

概述

`esp_http_client` 提供了一组 API，用于从 ESP-IDF 应用程序中发起 HTTP/S 请求，具体的使用步骤如下：

- 首先调用*`esp_http_client_init()`*，创建一个*`esp_http_client_handle_t`* 实例，即基于给定的*`esp_http_client_config_t`* 配置创建 HTTP 客户端句柄。此函数必须第一个被调用。若用户未明确定义参数的配置值，则使用默认值。
- 其次调用*`esp_http_client_perform()`*，执行 `esp_http_client` 的所有操作，包括打开连接、交换数据、关闭连接（如需要），同时在当前任务完成前阻塞该任务。所有相关的事件（在*`esp_http_client_config_t`* 中指定）将通过事件处理程序被调用。
- 最后调用*`esp_http_client_cleanup()`* 来关闭连接（如有），并释放所有分配给 HTTP 客户端实例的内存。此函数必须在操作完成后最后一个被调用。

应用示例

使用 ESP HTTP 客户端发起 HTTP/S 请求的简单示例，可参考 protocols/esp_http_client。

HTTP 基本请求

如需了解实现细节，请参考应用示例中的 `http_rest_with_url` 和 `http_rest_with_hostname_path` 函数。

持久连接

持久连接是 HTTP 客户端在多次交换中重复使用同一连接的方法。如果服务器没有使用 `Connection: close` 头来请求关闭连接，连接就会一直保持开放，用于其他新请求。

为了使 ESP HTTP 客户端充分利用持久连接的优势，建议尽可能多地使用同一个句柄实例来发起请求，可参考应用示例中的函数 `http_rest_with_url` 和 `http_rest_with_hostname_path`。示例中，一旦创建连接，即会在连接关闭前发出多个请求（如 `GET`、`POST`、`PUT` 等）。

HTTPS 请求

ESP HTTP 客户端支持使用 **mbedTLS** 的 SSL 连接，需将 `url` 配置为以 `https` 开头，或将 `transport_type` 设置为 `HTTP_TRANSPORT_OVER_SSL`。可以通过*CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS* 来配置 HTTPS 支持（默认启用）。

备注： 在发起 HTTPS 请求时，如需服务器验证，首先需要向 `esp_http_client_config_t` 配置中的 `cert_pem` 成员提供额外的根证书（PEM 格式）。用户还可以通过 `esp_http_client_config_t` 配置中的 `crt_bundle_attach` 成员，使用 `ESP x509 Certificate Bundle` 进行服务器验证。

如需了解上文备注中的实现细节，请参考应用示例中的函数 `https_with_url` 和 `https_with_hostname_path`。

HTTP 流

有些应用程序需要主动打开连接并控制数据交换（数据流）。在这种情况下，应用流程与常规请求不同。请参考以下示例：

- *esp_http_client_init()*：创建一个 HTTP 客户端句柄。
- `esp_http_client_set_*` 或 `esp_http_client_delete_*`：修改 HTTP 连接参数（可选）。
- *esp_http_client_open()*：用 `write_len`（该参数为需要写入服务器的内容长度）打开 HTTP 连接，设置 `write_len=0` 为只读连接。
- *esp_http_client_write()*：向服务器写入数据，最大长度为*esp_http_client_open()* 函数中的 `write_len` 值；配置 `write_len=0` 无需调用此函数。
- *esp_http_client_fetch_headers()*：在发送完请求头和服务器数据（如有）后，读取 HTTP 服务器的响应头。从服务器返回 `content-length`，并可以由*esp_http_client_get_status_code()* 继承，以获取连接的 HTTP 状态。
- *esp_http_client_read()*：读取 HTTP 流。
- *esp_http_client_close()*：关闭连接。
- *esp_http_client_cleanup()*：释放分配的资源。

如需了解实现细节，请参考应用示例中的函数 `http_perform_as_stream_reader`。

HTTP 认证

ESP HTTP 客户端同时支持 基本和 摘要认证。

- 用户可以在 `url` 或 `esp_http_client_config_t` 配置中的 `username` 和 `password` 处输入用户名和密码。对于 `auth_type = HTTP_AUTH_TYPE_BASIC`，HTTP 客户端只需执行一项操作就可通过认证过程。
- 如果 `auth_type = HTTP_AUTH_TYPE_NONE`，但配置中有 `username` 和 `password` 字段，HTTP 客户端需要执行两项操作。客户端在第一次尝试连接服务器时，会收到 `401 Unauthorized` 头，而后再根据这些信息来选择认证方法，并在第二项操作中执行。
- 如需了解实现细节，请参考应用示例中的函数 `http_auth_basic`、`http_auth_basic_redirect`（用于基本认证）和 `http_auth_digest`（用于摘要认证）。
- 目前，摘要认证仅支持 MD5 和 SHA-256 算法。

认证配置示例

- 基于 URI 的认证

```
esp_http_client_config_t config = {
    .url = "http://user:passwd@httpbin.org/basic-auth/user/passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

- 基于用户名和密码的认证

```
esp_http_client_config_t config = {
    .url = "http://httpbin.org/basic-auth/user/passwd",
    .username = "user",
    .password = "passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

事件处理

ESP HTTP 客户端支持事件处理，发生相关事件时会触发相应的事件处理程序。*esp_http_client_event_id_t* 中包含了所有使用 ESP HTTP 客户端执行 HTTP 请求时可能发生的事件。

通过*esp_http_client_config_t::event_handler* 设置回调函数即可启用事件处理功能。

ESP HTTP 客户端诊断信息

诊断信息可以帮助用户深入了解出现的问题。在 ESP HTTP 客户端中，可以通过在事件循环库 中注册事件处理程序来获取诊断信息。此功能的增加基于 ESP Insights 框架，该框架可帮助收集诊断信息。然而，即使不依赖 ESP Insights 框架，也可以获取诊断信息。事件处理程序可通过*esp_event_handler_register()* 函数注册到事件循环中。

事件循环中不同 HTTP 客户端事件的预期数据类型如下所示：

- HTTP_EVENT_ERROR : `esp_http_client_handle_t`
- HTTP_EVENT_ON_CONNECTED : `esp_http_client_handle_t`
- HTTP_EVENT_HEADERS_SENT : `esp_http_client_handle_t`
- HTTP_EVENT_ON_HEADER : `esp_http_client_handle_t`
- HTTP_EVENT_ON_DATA : `esp_http_client_on_data_t`
- HTTP_EVENT_ON_FINISH : `esp_http_client_handle_t`
- HTTP_EVENT_DISCONNECTED : `esp_http_client_handle_t`
- HTTP_EVENT_REDIRECT : `esp_http_client_redirect_event_data_t`

在无法接收到*HTTP_EVENT_DISCONNECTED* 之前，与事件数据一起接收到的*esp_http_client_handle_t* 将始终有效。这个句柄主要是为了区分不同的客户端连接，无法用于其他目的，因为它可能会随着客户端连接状态的变化而改变。

TLS 协议版本

可在*esp_http_client_config_t* 中设置用于底层 TLS 连接的 TLS 协议版本。了解更多信息，请参考*ESP-TLS* 中的 **TLS 协议版本**章节。

HTTP 客户端的 TLS 协议版本可按如下方式配置：

```
#include "esp_http_client.h"
esp_http_client_config_t config = {
    .tls_version = ESP_HTTP_CLIENT_TLS_VER_TLS_1_2,
};
```

API 参考

Header File

- components/esp_http_client/include/esp_http_client.h
- This header file can be included with:

```
#include "esp_http_client.h"
```

- This header file is a part of the API provided by the `esp_http_client` component. To declare that your component depends on `esp_http_client`, add the following to your CMakeLists.txt:

```
REQUIRES esp_http_client
```

or

```
PRIV_REQUIRES esp_http_client
```

Functions

esp_http_client_handle_t **esp_http_client_init** (const *esp_http_client_config_t* *config)

Start a HTTP session This function must be the first function to call, and it returns a esp_http_client_handle_t that you must use as input to other functions in the interface. This call MUST have a corresponding call to esp_http_client_cleanup when the operation is complete.

参数 **config** -- **[in]** The configurations, see `http_client_config_t`

返回

- `esp_http_client_handle_t`
- NULL if any errors

esp_err_t **esp_http_client_perform** (*esp_http_client_handle_t* client)

Invoke this function after `esp_http_client_init` and all the options calls are made, and will perform the transfer as described in the options. It must be called with the same esp_http_client_handle_t as input as the esp_http_client_init call returned. esp_http_client_perform performs the entire request in either blocking or non-blocking manner. By default, the API performs request in a blocking manner and returns when done, or if it failed, and in non-blocking manner, it returns if EAGAIN/EWOULDBLOCK or EINPROGRESS is encountered, or if it failed. And in case of non-blocking request, the user may call this API multiple times unless request & response is complete or there is a failure. To enable non-blocking esp_http_client_perform(), `is_async` member of *esp_http_client_config_t* must be set while making a call to esp_http_client_init() API. You can do any amount of calls to esp_http_client_perform while using the same esp_http_client_handle_t. The underlying connection may be kept open if the server allows it. If you intend to transfer more than one file, you are even encouraged to do so. esp_http_client will then attempt to reuse the same connection for the following transfers, thus making the operations faster, less CPU intense and using less network resources. Just note that you will have to use `esp_http_client_set_**` between the invokes to set options for the following esp_http_client_perform.

备注: You must never call this function simultaneously from two places using the same client handle. Let the function return first before invoking it another time. If you want parallel transfers, you must use several esp_http_client_handle_t. This function include `esp_http_client_open` -> `esp_http_client_write` -> `esp_http_client_fetch_headers` -> `esp_http_client_read` (and option) `esp_http_client_close`.

参数 **client** -- The esp_http_client handle

返回

- ESP_OK on successful
- ESP_FAIL on error

esp_err_t **esp_http_client_cancel_request** (*esp_http_client_handle_t* client)

Cancel an ongoing HTTP request. This API closes the current socket and opens a new socket with the same esp_http_client context.

参数 **client** -- The esp_http_client handle

返回

- ESP_OK on successful
- ESP_FAIL on error
- ESP_ERR_INVALID_ARG
- ESP_ERR_INVALID_STATE

esp_err_t **esp_http_client_set_url**(*esp_http_client_handle_t* client, const char *url)

Set URL for client, when performing this behavior, the options in the URL will replace the old ones.

参数

- **client** -- **[in]** The esp_http_client handle
- **url** -- **[in]** The url

返回

- ESP_OK
- ESP_FAIL

esp_err_t **esp_http_client_set_post_field**(*esp_http_client_handle_t* client, const char *data, int len)

Set post data, this function must be called before `esp_http_client_perform`. Note: The data parameter passed to this function is a pointer and this function will not copy the data.

参数

- **client** -- **[in]** The esp_http_client handle
- **data** -- **[in]** post data pointer
- **len** -- **[in]** post length

返回

- ESP_OK
- ESP_FAIL

int **esp_http_client_get_post_field**(*esp_http_client_handle_t* client, char **data)

Get current post field information.

参数

- **client** -- **[in]** The client
- **data** -- **[out]** Point to post data pointer

返回 Size of post data

esp_err_t **esp_http_client_set_header**(*esp_http_client_handle_t* client, const char *key, const char *value)

Set http request header, this function must be called after esp_http_client_init and before any perform function.

参数

- **client** -- **[in]** The esp_http_client handle
- **key** -- **[in]** The header key
- **value** -- **[in]** The header value

返回

- ESP_OK
- ESP_FAIL

esp_err_t **esp_http_client_get_header**(*esp_http_client_handle_t* client, const char *key, char **value)

Get http request header. The value parameter will be set to NULL if there is no header which is same as the key specified, otherwise the address of header value will be assigned to value parameter. This function must be called after `esp_http_client_init`.

参数

- **client** -- **[in]** The esp_http_client handle
- **key** -- **[in]** The header key
- **value** -- **[out]** The header value

返回

- ESP_OK
- ESP_FAIL

esp_err_t **esp_http_client_get_username**(*esp_http_client_handle_t* client, char **value)

Get http request username. The address of username buffer will be assigned to value parameter. This function must be called after `esp_http_client_init`.

参数

- **client** -- **[in]** The esp_http_client handle
- **value** -- **[out]** The username value

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_set_username**(*esp_http_client_handle_t* client, const char *username)

Set http request username. The value of username parameter will be assigned to username buffer. If the username parameter is NULL then username buffer will be freed.

参数

- **client** -- **[in]** The esp_http_client handle
- **username** -- **[in]** The username value

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_get_password**(*esp_http_client_handle_t* client, char **value)

Get http request password. The address of password buffer will be assigned to value parameter. This function must be called after `esp_http_client_init`.

参数

- **client** -- **[in]** The esp_http_client handle
- **value** -- **[out]** The password value

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_set_password**(*esp_http_client_handle_t* client, const char *password)

Set http request password. The value of password parameter will be assigned to password buffer. If the password parameter is NULL then password buffer will be freed.

参数

- **client** -- **[in]** The esp_http_client handle
- **password** -- **[in]** The password value

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_set_authtype**(*esp_http_client_handle_t* client, *esp_http_client_auth_type_t* auth_type)

Set http request auth_type.

参数

- **client** -- **[in]** The esp_http_client handle
- **auth_type** -- **[in]** The esp_http_client auth type

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_get_user_data**(*esp_http_client_handle_t* client, void **data)

Get http request user_data. The value stored from the *esp_http_client_config_t* will be written to the address passed into data.

参数

- **client** -- **[in]** The esp_http_client handle
- **data** -- **[out]** A pointer to the pointer that will be set to user_data.

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_set_user_data**(*esp_http_client_handle_t* client, void *data)

Set http request user_data. The value passed in +data+ will be available during event callbacks. No memory management will be performed on the user's behalf.

参数

- **client** -- **[in]** The esp_http_client handle
- **data** -- **[in]** The pointer to the user data

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

int **esp_http_client_get_errno**(*esp_http_client_handle_t* client)

Get HTTP client session errno.

参数 **client** -- **[in]** The esp_http_client handle

返回

- (-1) if invalid argument
- errno

esp_err_t **esp_http_client_set_method**(*esp_http_client_handle_t* client, *esp_http_client_method_t* method)

Set http request method.

参数

- **client** -- **[in]** The esp_http_client handle
- **method** -- **[in]** The method

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_set_timeout_ms**(*esp_http_client_handle_t* client, int timeout_ms)

Set http request timeout.

参数

- **client** -- **[in]** The esp_http_client handle
- **timeout_ms** -- **[in]** The timeout value

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_delete_header**(*esp_http_client_handle_t* client, const char *key)

Delete http request header.

参数

- **client** -- **[in]** The esp_http_client handle
- **key** -- **[in]** The key

返回

- ESP_OK
- ESP_FAIL

esp_err_t **esp_http_client_open**(*esp_http_client_handle_t* client, int write_len)

This function will be open the connection, write all header strings and return.

参数

- **client** -- **[in]** The esp_http_client handle
- **write_len** -- **[in]** HTTP Content length need to write to the server

返回

- ESP_OK
- ESP_FAIL

int **esp_http_client_write**(*esp_http_client_handle_t* client, const char *buffer, int len)

This function will write data to the HTTP connection previously opened by esp_http_client_open()

参数

- **client** -- **[in]** The esp_http_client handle
- **buffer** -- The buffer
- **len** -- **[in]** This value must not be larger than the write_len parameter provided to esp_http_client_open()

返回

- (-1) if any errors
- Length of data written

int64_t **esp_http_client_fetch_headers**(*esp_http_client_handle_t* client)

This function need to call after esp_http_client_open, it will read from http stream, process all receive headers.

参数 **client** -- **[in]** The esp_http_client handle

返回

- (0) if stream doesn't contain content-length header, or chunked encoding (checked by `esp_http_client_is_chunked` response)
- (-1: ESP_FAIL) if any errors
- (-ESP_ERR_HTTP_EAGAIN = -0x7007) if call is timed-out before any data was ready
- Download data length defined by content-length header

bool **esp_http_client_is_chunked_response**(*esp_http_client_handle_t* client)

Check response data is chunked.

参数 **client** -- **[in]** The esp_http_client handle

返回 true or false

int **esp_http_client_read**(*esp_http_client_handle_t* client, char *buffer, int len)

Read data from http stream.

备注: (-ESP_ERR_HTTP_EAGAIN = -0x7007) is returned when call is timed-out before any data was ready

---

参数

- **client** -- **[in]** The esp_http_client handle
- **buffer** -- The buffer
- **len** -- **[in]** The length

返回

- (-1) if any errors
- Length of data was read

int **esp_http_client_get_status_code**(*esp_http_client_handle_t* client)

Get http response status code, the valid value if this function invoke after `esp_http_client_perform`

参数 **client** -- **[in]** The esp_http_client handle

返回 Status code

int64_t **esp_http_client_get_content_length**(*esp_http_client_handle_t* client)

Get http response content length (from header Content-Length) the valid value if this function invoke after `esp_http_client_perform`

参数 **client** -- **[in]** The esp_http_client handle

返回

- (-1) Chunked transfer
- Content-Length value as bytes

esp_err_t **esp_http_client_close**(*esp_http_client_handle_t* client)

Close http connection, still kept all http request resources.

参数 **client** -- **[in]** The esp_http_client handle

返回

- ESP_OK
- ESP_FAIL

esp_err_t **esp_http_client_cleanup**(*esp_http_client_handle_t* client)

This function must be the last function to call for an session. It is the opposite of the esp_http_client_init function and must be called with the same handle as input that a esp_http_client_init call returned. This might close all connections this handle has used and possibly has kept open until now. Don't call this function if you intend to transfer more files, re-using handles is a key to good performance with esp_http_client.

参数 **client** -- **[in]** The esp_http_client handle

返回

- ESP_OK
- ESP_FAIL

esp_http_client_transport_t **esp_http_client_get_transport_type**(*esp_http_client_handle_t* client)

Get transport type.

参数 **client** -- **[in]** The esp_http_client handle

返回

- HTTP_TRANSPORT_UNKNOWN
- HTTP_TRANSPORT_OVER_TCP
- HTTP_TRANSPORT_OVER_SSL

esp_err_t **esp_http_client_set_redirection**(*esp_http_client_handle_t* client)

Set redirection URL. When received the 30x code from the server, the client stores the redirect URL provided by the server. This function will set the current URL to redirect to enable client to execute the redirection request. When `disable_auto_redirect` is set, the client will not call this function but the event `HTTP_EVENT_REDIRECT` will be dispatched giving the user control over the redirection event.

参数 **client** -- **[in]** The esp_http_client handle

返回

- ESP_OK
- ESP_FAIL

esp_err_t **esp_http_client_reset_redirect_counter**(*esp_http_client_handle_t* client)

Reset the redirection counter. This is useful to reset redirect counter in cases where the same handle is used for multiple requests.

参数 **client** -- **[in]** The esp_http_client handle

返回

- ESP_OK
- ESP_ERR_INVALID_ARG

esp_err_t **esp_http_client_set_auth_data**(*esp_http_client_handle_t* client, const char *auth_data, int len)

On receiving a custom authentication header, this API can be invoked to set the authentication information from the header. This API can be called from the event handler.

参数

- **client** -- **[in]** The esp_http_client handle
- **auth_data** -- **[in]** The authentication data received in the header
- **len** -- **[in]** length of auth_data.

返回

- ESP_ERR_INVALID_ARG
- ESP_OK

void **esp_http_client_add_auth**(*esp_http_client_handle_t* client)

On receiving HTTP Status code 401, this API can be invoked to add authorization information.

备注: There is a possibility of receiving body message with redirection status codes, thus make sure to flush

off body data after calling this API.

参数 **client** -- **[in]** The esp_http_client handle

bool **esp_http_client_is_complete_data_received**(*esp_http_client_handle_t* client)

Checks if entire data in the response has been read without any error.

参数 **client** -- **[in]** The esp_http_client handle

返回

- true
- false

int **esp_http_client_read_response**(*esp_http_client_handle_t* client, char *buffer, int len)

Helper API to read larger data chunks This is a helper API which internally calls `esp_http_client_read` multiple times till the end of data is reached or till the buffer gets full.

参数

- **client** -- **[in]** The esp_http_client handle
- **buffer** -- The buffer
- **len** -- **[in]** The buffer length

返回

- Length of data was read

esp_err_t **esp_http_client_flush_response**(*esp_http_client_handle_t* client, int *len)

Process all remaining response data This uses an internal buffer to repeatedly receive, parse, and discard response data until complete data is processed. As no additional user-supplied buffer is required, this may be preferable to `esp_http_client_read_response` in situations where the content of the response may be ignored.

参数

- **client** -- **[in]** The esp_http_client handle
- **len** -- Length of data discarded

返回

- ESP_OK If successful, len will have discarded length
- ESP_FAIL If failed to read response
- ESP_ERR_INVALID_ARG If the client is NULL

esp_err_t **esp_http_client_get_url**(*esp_http_client_handle_t* client, char *url, const int len)

Get URL from client.

参数

- **client** -- **[in]** The esp_http_client handle
- **url** -- **[inout]** The buffer to store URL
- **len** -- **[in]** The buffer length

返回

- ESP_OK
- ESP_FAIL

esp_err_t **esp_http_client_get_chunk_length**(*esp_http_client_handle_t* client, int *len)

Get Chunk-Length from client.

参数

- **client** -- **[in]** The esp_http_client handle
- **len** -- **[out]** Variable to store length

返回

- ESP_OK If successful, len will have length of current chunk
- ESP_FAIL If the server is not a chunked server
- ESP_ERR_INVALID_ARG If the client or len are NULL

Structures

struct **esp_http_client_event**

HTTP Client events data.

Public Members

esp_http_client_event_id_t **event_id**

event_id, to know the cause of the event

esp_http_client_handle_t **client**

esp_http_client_handle_t context

void ***data**

data of the event

int **data_len**

data length of data

void ***user_data**

user_data context, from *esp_http_client_config_t* user_data

char ***header_key**

For HTTP_EVENT_ON_HEADER event_id, it's store current http header key

char ***header_value**

For HTTP_EVENT_ON_HEADER event_id, it's store current http header value

struct **esp_http_client_on_data**

Argument structure for HTTP_EVENT_ON_DATA event.

Public Members

esp_http_client_handle_t **client**

Client handle

int64_t **data_process**

Total data processed

struct **esp_http_client_redirect_event_data**

Argument structure for HTTP_EVENT_REDIRECT event.

Public Members

esp_http_client_handle_t **client**

Client handle

int **status_code**

Status Code

struct **esp_http_client_config_t**

HTTP configuration.

Public Members

const char ***url**

HTTP URL, the information on the URL is most important, it overrides the other fields below, if any

const char ***host**

Domain or IP as string

int **port**

Port to connect, default depend on esp_http_client_transport_t (80 or 443)

const char ***username**

Using for Http authentication

const char ***password**

Using for Http authentication

esp_http_client_auth_type_t **auth_type**

Http authentication type, see `esp_http_client_auth_type_t`

const char ***path**

HTTP Path, if not set, default is `/`

const char ***query**

HTTP query

const char ***cert_pem**

SSL server certification, PEM format as string, if the client requires to verify server

size_t **cert_len**

Length of the buffer pointed to by cert_pem. May be 0 for null-terminated pem

const char ***client_cert_pem**

SSL client certification, PEM format as string, if the server requires to verify client

size_t **client_cert_len**

Length of the buffer pointed to by client_cert_pem. May be 0 for null-terminated pem

const char ***client_key_pem**

SSL client key, PEM format as string, if the server requires to verify client

size_t **client_key_len**

Length of the buffer pointed to by client_key_pem. May be 0 for null-terminated pem

const char ***client_key_password**

Client key decryption password string

size_t **client_key_password_len**

String length of the password pointed to by client_key_password

esp_http_client_proto_ver_t **tls_version**

TLS protocol version of the connection, e.g., TLS 1.2, TLS 1.3 (default - no preference)

const char ***user_agent**

The User Agent string to send with HTTP requests

esp_http_client_method_t **method**

HTTP Method

int **timeout_ms**

Network timeout in milliseconds

bool **disable_auto_redirect**

Disable HTTP automatic redirects

int **max_redirection_count**

Max number of redirections on receiving HTTP redirect status code, using default value if zero

int **max_authorization_retries**

Max connection retries on receiving HTTP unauthorized status code, using default value if zero. Disables authorization retry if -1

http_event_handle_cb **event_handler**

HTTP Event Handle

esp_http_client_transport_t **transport_type**

HTTP transport type, see `esp_http_client_transport_t`

int **buffer_size**

HTTP receive buffer size

int **buffer_size_tx**

HTTP transmit buffer size

void ***user_data**

HTTP user_data context

bool **is_async**

Set asynchronous mode, only supported with HTTPS for now

bool **use_global_ca_store**

Use a global ca_store for all the connections in which this bool is set.

bool **skip_cert_common_name_check**

Skip any validation of server certificate CN field

const char ***common_name**

Pointer to the string containing server certificate common name. If non-NULL, server certificate CN must match this name, If NULL, server certificate CN must match hostname.

esp_err_t (***crt_bundle_attach**)(void *conf)

Function pointer to esp_crt_bundle_attach. Enables the use of certification bundle for server verification, must be enabled in menuconfig

bool **keep_alive_enable**

Enable keep-alive timeout

int **keep_alive_idle**

Keep-alive idle time. Default is 5 (second)

int **keep_alive_interval**

Keep-alive interval time. Default is 5 (second)

int **keep_alive_count**

Keep-alive packet retry send count. Default is 3 counts

struct ifreq ***if_name**

The name of interface for data to go through. Use the default interface without setting

void ***ds_data**

Pointer for digital signature peripheral context, see ESP-TLS Documentation for more details

Macros

DEFAULT_HTTP_BUF_SIZE

ESP_ERR_HTTP_BASE

Starting number of HTTP error codes

ESP_ERR_HTTP_MAX_REDIRECT

The error exceeds the number of HTTP redirects

ESP_ERR_HTTP_CONNECT

Error open the HTTP connection

ESP_ERR_HTTP_WRITE_DATA

Error write HTTP data

ESP_ERR_HTTP_FETCH_HEADER

Error read HTTP header from server

ESP_ERR_HTTP_INVALID_TRANSPORT

There are no transport support for the input scheme

ESP_ERR_HTTP_CONNECTING

HTTP connection hasn't been established yet

ESP_ERR_HTTP_EAGAIN

Mapping of errno EAGAIN to esp_err_t

ESP_ERR_HTTP_CONNECTION_CLOSED

Read FIN from peer and the connection closed

Type Definitions

typedef struct esp_http_client ***esp_http_client_handle_t**

typedef struct *esp_http_client_event* ***esp_http_client_event_handle_t**

typedef struct *esp_http_client_event* **esp_http_client_event_t**

HTTP Client events data.

typedef struct *esp_http_client_on_data* **esp_http_client_on_data_t**

Argument structure for HTTP_EVENT_ON_DATA event.

typedef struct *esp_http_client_redirect_event_data* **esp_http_client_redirect_event_data_t**

Argument structure for HTTP_EVENT_REDIRECT event.

typedef *esp_err_t* (***http_event_handle_cb**)(*esp_http_client_event_t* *evt)

Enumerations

enum **esp_http_client_event_id_t**

HTTP Client events id.

Values:

enumerator **HTTP_EVENT_ERROR**

This event occurs when there are any errors during execution

enumerator **HTTP_EVENT_ON_CONNECTED**

Once the HTTP has been connected to the server, no data exchange has been performed

enumerator **HTTP_EVENT_HEADERS_SENT**

After sending all the headers to the server

enumerator **HTTP_EVENT_HEADER_SENT**

This header has been kept for backward compatibility and will be deprecated in future versions esp-idf

enumerator **HTTP_EVENT_ON_HEADER**

Occurs when receiving each header sent from the server

enumerator **HTTP_EVENT_ON_DATA**

Occurs when receiving data from the server, possibly multiple portions of the packet

enumerator **HTTP_EVENT_ON_FINISH**

Occurs when finish a HTTP session

enumerator **HTTP_EVENT_DISCONNECTED**

The connection has been disconnected

enumerator **HTTP_EVENT_REDIRECT**

Intercepting HTTP redirects to handle them manually

enum **esp_http_client_transport_t**

HTTP Client transport.

Values:

enumerator **HTTP_TRANSPORT_UNKNOWN**

Unknown

enumerator **HTTP_TRANSPORT_OVER_TCP**

Transport over tcp

enumerator **HTTP_TRANSPORT_OVER_SSL**

Transport over ssl

enum **esp_http_client_proto_ver_t**

Values:

enumerator **ESP_HTTP_CLIENT_TLS_VER_ANY**

enumerator **ESP_HTTP_CLIENT_TLS_VER_TLS_1_2**

enumerator **ESP_HTTP_CLIENT_TLS_VER_TLS_1_3**

enumerator **ESP_HTTP_CLIENT_TLS_VER_MAX**

enum **esp_http_client_method_t**

HTTP method.

Values:

enumerator **HTTP_METHOD_GET**

HTTP GET Method

enumerator **HTTP_METHOD_POST**

HTTP POST Method

enumerator **HTTP_METHOD_PUT**

HTTP PUT Method

enumerator **HTTP_METHOD_PATCH**
HTTP PATCH Method

enumerator **HTTP_METHOD_DELETE**
HTTP DELETE Method

enumerator **HTTP_METHOD_HEAD**
HTTP HEAD Method

enumerator **HTTP_METHOD_NOTIFY**
HTTP NOTIFY Method

enumerator **HTTP_METHOD_SUBSCRIBE**
HTTP SUBSCRIBE Method

enumerator **HTTP_METHOD_UNSUBSCRIBE**
HTTP UNSUBSCRIBE Method

enumerator **HTTP_METHOD_OPTIONS**
HTTP OPTIONS Method

enumerator **HTTP_METHOD_COPY**
HTTP COPY Method

enumerator **HTTP_METHOD_MOVE**
HTTP MOVE Method

enumerator **HTTP_METHOD_LOCK**
HTTP LOCK Method

enumerator **HTTP_METHOD_UNLOCK**
HTTP UNLOCK Method

enumerator **HTTP_METHOD_PROPFIND**
HTTP PROPFIND Method

enumerator **HTTP_METHOD_PROPPATCH**
HTTP PROPPATCH Method

enumerator **HTTP_METHOD_MKCOL**
HTTP MKCOL Method

enumerator **HTTP_METHOD_MAX**

enum **esp_http_client_auth_type_t**
HTTP Authentication type.

Values:

enumerator **HTTP_AUTH_TYPE_NONE**

No authention

enumerator **HTTP_AUTH_TYPE_BASIC**

HTTP Basic authentication

enumerator **HTTP_AUTH_TYPE_DIGEST**

HTTP Digest authentication

enum **HttpStatus_Code**

Enum for the HTTP status codes.

Values:

enumerator **HttpStatus_Ok**

enumerator **HttpStatus_MultipleChoices**

enumerator **HttpStatus_MovedPermanently**

enumerator **HttpStatus_Found**

enumerator **HttpStatus_SeeOther**

enumerator **HttpStatus_TemporaryRedirect**

enumerator **HttpStatus_PermanentRedirect**

enumerator **HttpStatus_BadRequest**

enumerator **HttpStatus_Unauthorized**

enumerator **HttpStatus_Forbidden**

enumerator **HttpStatus_NotFound**

enumerator **HttpStatus_InternalError**

2.2.6 ESP 本地控制

概述

通过 ESP-IDF 的 ESP 本地控制 (**esp_local_ctrl**) 组件，可使用 HTTPS 或 BLE 协议控制 ESP 设备。通过一系列可配置的处理程序，该组件允许你对应用程序定义的读/写 **属性**进行访问。

同样，对于 HTTP 传输：

```
/* Set the configuration */
httpd_ssl_config_t https_conf = HTTPD_SSL_CONFIG_DEFAULT();

/* Load server certificate */
extern const unsigned char servercert_start[] asm("_binary_servercert_pem_
↪start");
extern const unsigned char servercert_end[]   asm("_binary_servercert_pem_
↪end");
https_conf.servercert = servercert_start;
https_conf.servercert_len = servercert_end - servercert_start;

/* Load server private key */
extern const unsigned char prvtkey_pem_start[] asm("_binary_prvtkey_pem_
↪start");
extern const unsigned char prvtkey_pem_end[]   asm("_binary_prvtkey_pem_
↪end");
https_conf.prvtkey_pem = prvtkey_pem_start;
https_conf.prvtkey_len = prvtkey_pem_end - prvtkey_pem_start;

esp_local_ctrl_config_t config = {
    .transport = ESP_LOCAL_CTRL_TRANSPORT_HTTPD,
    .transport_config = {
        .httpd = &https_conf
    },
    .proto_sec = {
        .version = PROTOCOM_SEC0,
        .custom_handle = NULL,
        .sec_params = NULL,
    },
    .handlers = {
        /* User defined handler functions */
        .get_prop_values = get_property_values,
        .set_prop_values = set_property_values,
        .usr_ctx         = NULL,
        .usr_ctx_free_fn = NULL
    },
    /* Maximum number of properties that may be set */
    .max_properties = 10
};

/* Start esp_local_ctrl service */
ESP_ERROR_CHECK(esp_local_ctrl_start(&config));
```

你可以利用以下选项设置 ESP 本地控制的传输安全性：

1. `PROTOCOM_SEC2`: 指定使用基于 SRP6a 的密钥交换和基于 AES-GCM 的端到端加密。这一选项通过增强的 PAKE 协议（即 SRP6a）提供了强大的安全保障，是最受欢迎的选项。
2. `PROTOCOM_SEC1`: 指定使用基于 Curve25519 的密钥交换和基于 AES-CTR 的端到端加密。
3. `PROTOCOM_SEC0`: 指定以明文（无安全性）方式交换数据。
4. `PROTOCOM_SEC_CUSTOM`: 自定义安全需求。注意，使用这一选项时，必须提供 `protocomm_security_t *` 类型的 `custom_handle`。

备注： 相应的安全方案需通过项目配置菜单启用。要了解详情，请参考*Protocol Communication* 中关于启用 protocomm 安全版本的章节。

创建属性

启用 **esp_local_ctrl** 后，可以为其添加属性。每个属性必须具有唯一的名称 `name`（字符串），还需具有类型 `type`（如枚举）、标记 `flags`（位域）和大小 `size`。

如果希望属性值的长度可变（例如，字符串或字节流），则 `size` 值应保持为 0。对于有固定长度属性值的数据类型，如整型、浮点型等，将 `size` 字段设置为正确的值，有助于 **esp_local_ctrl** 对接收到的写入请求的参数进行内部检查。

`type` 和 `flags` 字段的含义取决于具体的应用程序，它们可以是枚举、位域、甚至整型。可以使用 `type` 表示属性，用 `flags` 指定属性的特征。

例如，以下属性被用作时间戳。此处假设应用程序定义了 `TYPE_TIMESTAMP` 和 `READONLY` 来设置此处的 `type` 和 `flags` 字段：

```
/* Create a timestamp property */
esp_local_ctrl_prop_t timestamp = {
    .name         = "timestamp",
    .type         = TYPE_TIMESTAMP,
    .size         = sizeof(int32_t),
    .flags        = READONLY,
    .ctx          = func_get_time,
    .ctx_free_fn = NULL
};

/* Now register the property */
esp_local_ctrl_add_property(&timestamp);
```

另外，此示例中还设置了一个 ctx 字段，指向自定义的 `func_get_time()`，用于在属性的 get/set 处理程序中检索时间戳。

以下为 `get_prop_values()` 处理程序的一个示例，用于检索时间戳：

```
static esp_err_t get_property_values(size_t props_count,
                                     const esp_local_ctrl_prop_t *props,
                                     esp_local_ctrl_prop_val_t *prop_
↪values,
                                     void *usr_ctx)
{
    for (uint32_t i = 0; i < props_count; i++) {
        ESP_LOGI(TAG, "Reading %s", props[i].name);
        if (props[i].type == TYPE_TIMESTAMP) {
            /* Obtain the timer function from ctx */
            int32_t (*func_get_time)(void) = props[i].ctx;

            /* Use static variable for saving the value. This is␣
↪essential because the value has to be valid even after this function␣
↪returns. Alternative is to use dynamic allocation and set the free_fn␣
↪field */
            static int32_t ts = func_get_time();
            prop_values[i].data = &ts;
        }
    }
    return ESP_OK;
}
```

以下为 `set_prop_values()` 应用程序的一个示例，注意此示例是如何为只读属性限制写入操作：

```
static esp_err_t set_property_values(size_t props_count,
                                     const esp_local_ctrl_prop_t *props,
                                     const esp_local_ctrl_prop_val_t␣
↪*prop_values,
                                     void *usr_ctx)
{
    for (uint32_t i = 0; i < props_count; i++) {
        if (props[i].flags & READONLY) {
            ESP_LOGE(TAG, "Cannot write to read-only property %s",␣
↪props[i].name);
```

(下页继续)

(续上页)

```
            return ESP_ERR_INVALID_ARG;
        } else {
            ESP_LOGI(TAG, "Setting %s", props[i].name);

            /* For keeping it simple, lets only log the incoming data */
            ESP_LOG_BUFFER_HEX_LEVEL(TAG, prop_values[i].data,
                                     prop_values[i].size, ESP_LOG_INFO);
        }
    }
    return ESP_OK;
}
```

完整示例请参见 protocols/esp_local_ctrl。

客户端实现

在客户端的实现过程中，首先，通过支持的传输模式与设备建立 protocomm 会话，然后发送并接收 **esp_local_ctrl** 服务能够处理的 protobuf 信息。**esp_local_ctrl** 服务会将这些信息转换为请求，并发起相应的处理程序 (set/get)。接着，为每个处理程序生成的响应会被再次打包到一条 protobuf 信息中，传输回客户端。

以下是 **esp_local_ctrl** 服务能够处理的各种 protobuf 信息：

1. `get_prop_count`：返回服务支持的属性总数。
2. `get_prop_values`：接受一个索引数组，并返回这些索引相对应的属性信息（名称、类型、标志）和属性值。
3. `set_prop_values`：接受一个索引数组和一个新值数组，用于设置索引对应的属性值。

注意，在多个会话中，一个属性的索引可能相同，也可能不同。因此，客户端必须用唯一的属性名称来识别属性。每次建立新会话时，客户端都应首先调用 `get_prop_count`，然后调用 `get_prop_values`，为所有属性建立从索引到名称的映射。为一组属性调用 `set_prop_values` 时，必须先用创建的映射将名称转换为索引。如前所述，每次使用同一设备建立新会话时，客户端必须刷新该映射。

下面列出了 **esp_local_ctrl** 服务提供的各种 protocomm 端点：

表 1: ESP 本地控制服务提供的端点

端点名称 (BLE + GATT 服务器)	URI (HTTPS 服务器 + mDNS)	描述
esp_local_ctrl/version	*https://<mdns-hostname>.local/esp_local_ctrl/version*	检索版本字符串
esp_local_ctrl/control	*https://<mdns-hostname>.local/esp_local_ctrl/control*	发送或接收控制信息

API 参考

Header File

- components/esp_local_ctrl/include/esp_local_ctrl.h
- This header file can be included with:

```
#include "esp_local_ctrl.h"
```

- This header file is a part of the API provided by the `esp_local_ctrl` component. To declare that your component depends on `esp_local_ctrl`, add the following to your CMakeLists.txt:

```
REQUIRES esp_local_ctrl
```

or

```
PRIV_REQUIRES esp_local_ctrl
```

Functions

const *esp_local_ctrl_transport_t* ***esp_local_ctrl_get_transport_ble**(void)*

Function for obtaining BLE transport mode.

const *esp_local_ctrl_transport_t* ***esp_local_ctrl_get_transport_httpd**(void)*

Function for obtaining HTTPD transport mode.

esp_err_t **esp_local_ctrl_start**(const *esp_local_ctrl_config_t* *config)

Start local control service.

参数 **config** -- **[in]** Pointer to configuration structure

返回

- ESP_OK : Success
- ESP_FAIL : Failure

esp_err_t **esp_local_ctrl_stop**(void)

Stop local control service.

esp_err_t **esp_local_ctrl_add_property**(const *esp_local_ctrl_prop_t* *prop)

Add a new property.

This adds a new property and allocates internal resources for it. The total number of properties that could be added is limited by configuration option `max_properties`

参数 **prop** -- **[in]** Property description structure

返回

- ESP_OK : Success
- ESP_FAIL : Failure

esp_err_t **esp_local_ctrl_remove_property**(const char *name)

Remove a property.

This finds a property by name, and releases the internal resources which are associated with it.

参数 **name** -- **[in]** Name of the property to remove

返回

- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Failure

const *esp_local_ctrl_prop_t* ***esp_local_ctrl_get_property**(const char *name)*

Get property description structure by name.

This API may be used to get a property's context structure `esp_local_ctrl_prop_t` when its name is known

参数 **name** -- **[in]** Name of the property to find

返回

- Pointer to property
- NULL if not found

esp_err_t **esp_local_ctrl_set_handler**(const char *ep_name, *protocomm_req_handler_t* handler, void *user_ctx)

Register protocomm handler for a custom endpoint.

This API can be called by the application to register a protocomm handler for an endpoint after the local control service has started.

备注: In case of BLE transport the names and uuids of all custom endpoints must be provided beforehand as a part of the `protocomm_ble_config_t` structure set in `esp_local_ctrl_config_t`, and passed

to `esp_local_ctrl_start()`.

参数

- **ep_name** -- **[in]** Name of the endpoint
- **handler** -- **[in]** Endpoint handler function
- **user_ctx** -- **[in]** User data

返回

- ESP_OK : Success
- ESP_FAIL : Failure

Unions

union **esp_local_ctrl_transport_config_t**

#include <esp_local_ctrl.h> Transport mode (BLE / HTTPD) configuration.

Public Members

esp_local_ctrl_transport_config_ble_t ***ble**

This is same as `protocomm_ble_config_t`. See `protocomm_ble.h` for available configuration parameters.

esp_local_ctrl_transport_config_httpd_t ***httpd**

This is same as `httpd_ssl_config_t`. See `esp_https_server.h` for available configuration parameters.

Structures

struct **esp_local_ctrl_prop**

Property description data structure, which is to be populated and passed to the `esp_local_ctrl_add_property()` function.

Once a property is added, its structure is available for read-only access inside `get_prop_values()` and `set_prop_values()` handlers.

Public Members

char ***name**

Unique name of property

uint32_t **type**

Type of property. This may be set to application defined enums

size_t **size**

Size of the property value, which:

- if zero, the property can have values of variable size
- if non-zero, the property can have values of fixed size only, therefore, checks are performed internally by esp_local_ctrl when setting the value of such a property

uint32_t **flags**

Flags set for this property. This could be a bit field. A flag may indicate property behavior, e.g. read-only / constant

void ***ctx**

Pointer to some context data relevant for this property. This will be available for use inside the `get_prop_values` and `set_prop_values` handlers as a part of this property structure. When set, this is valid throughout the lifetime of a property, till either the property is removed or the esp_local_ctrl service is stopped.

void (***ctx_free_fn**)(void *ctx)

Function used by esp_local_ctrl to internally free the property context when `esp_local_ctrl_remove_property()` or `esp_local_ctrl_stop()` is called.

struct **esp_local_ctrl_prop_val**

Property value data structure. This gets passed to the `get_prop_values()` and `set_prop_values()` handlers for the purpose of retrieving or setting the present value of a property.

Public Members

void ***data**

Pointer to memory holding property value

size_t **size**

Size of property value

void (***free_fn**)(void *data)

This may be set by the application in `get_prop_values()` handler to tell `esp_local_ctrl` to call this function on the data pointer above, for freeing its resources after sending the `get_prop_values` response.

struct **esp_local_ctrl_handlers**

Handlers for receiving and responding to local control commands for getting and setting properties.

Public Members

esp_err_t (***get_prop_values**)(size_t props_count, const *esp_local_ctrl_prop_t* props[], *esp_local_ctrl_prop_val_t* prop_values[], void *usr_ctx)

Handler function to be implemented for retrieving current values of properties.

备注： If any of the properties have fixed sizes, the size field of corresponding element in `prop_values` need to be set

---

Param props_count **[in]** Total elements in the props array

Param props **[in]** Array of properties, the current values for which have been requested by the client

Param prop_values **[out]** Array of empty property values, the elements of which need to be populated with the current values of those properties specified by props argument

Param usr_ctx **[in]** This provides value of the `usr_ctx` field of `esp_local_ctrl_handlers_t` structure

Return Returning different error codes will convey the corresponding protocol level errors to the client :
- ESP_OK : Success
- ESP_ERR_INVALID_ARG : InvalidArgument
- ESP_ERR_INVALID_STATE : InvalidProto
- All other error codes : InternalError

esp_err_t (***set_prop_values**)(size_t props_count, const *esp_local_ctrl_prop_t* props[], const *esp_local_ctrl_prop_val_t* prop_values[], void *usr_ctx)

Handler function to be implemented for changing values of properties.

备注: If any of the properties have variable sizes, the size field of the corresponding element in `prop_values` must be checked explicitly before making any assumptions on the size.

Param props_count **[in]** Total elements in the props array
Param props **[in]** Array of properties, the values for which the client requests to change
Param prop_values **[in]** Array of property values, the elements of which need to be used for updating those properties specified by props argument
Param usr_ctx **[in]** This provides value of the `usr_ctx` field of `esp_local_ctrl_handlers_t` structure
Return Returning different error codes will convey the corresponding protocol level errors to the client :
- ESP_OK : Success
- ESP_ERR_INVALID_ARG : InvalidArgument
- ESP_ERR_INVALID_STATE : InvalidProto
- All other error codes : InternalError

void ***usr_ctx**

Context pointer to be passed to above handler functions upon invocation. This is different from the property level context, as this is valid throughout the lifetime of the `esp_local_ctrl` service, and freed only when the service is stopped.

void (***usr_ctx_free_fn**)(void *usr_ctx)

Pointer to function which will be internally invoked on `usr_ctx` for freeing the context resources when `esp_local_ctrl_stop()` is called.

struct **esp_local_ctrl_proto_sec_cfg**

Protocom security configs

Public Members

esp_local_ctrl_proto_sec_t **version**

This sets protocom security version, sec0/sec1 or custom If custom, user must provide handle via `proto_sec_custom_handle` below

void ***custom_handle**

Custom security handle if security is set custom via `proto_sec` above This handle must follow `protocomm_security_t` signature

const void ***pop**

Proof of possession to be used for local control. Could be NULL.

const void ***sec_params**

Pointer to security params (NULL if not needed). This is not needed for protocomm security 0 This pointer should hold the struct of type esp_local_ctrl_security1_params_t for protocomm security 1 and esp_local_ctrl_security2_params_t for protocomm security 2 respectively. Could be NULL.

struct **esp_local_ctrl_config**

Configuration structure to pass to `esp_local_ctrl_start()`

Public Members

const *esp_local_ctrl_transport_t* ***transport**

Transport layer over which service will be provided

esp_local_ctrl_transport_config_t **transport_config**

Transport layer over which service will be provided

esp_local_ctrl_proto_sec_cfg_t **proto_sec**

Security version and POP

esp_local_ctrl_handlers_t **handlers**

Register handlers for responding to get/set requests on properties

size_t **max_properties**

This limits the number of properties that are available at a time

Macros

ESP_LOCAL_CTRL_TRANSPORT_BLE

ESP_LOCAL_CTRL_TRANSPORT_HTTPD

Type Definitions

typedef struct *esp_local_ctrl_prop* **esp_local_ctrl_prop_t**

Property description data structure, which is to be populated and passed to the `esp_local_ctrl_add_property()` function.

Once a property is added, its structure is available for read-only access inside `get_prop_values()` and `set_prop_values()` handlers.

typedef struct *esp_local_ctrl_prop_val* **esp_local_ctrl_prop_val_t**

Property value data structure. This gets passed to the `get_prop_values()` and `set_prop_values()` handlers for the purpose of retrieving or setting the present value of a property.

typedef struct *esp_local_ctrl_handlers* **esp_local_ctrl_handlers_t**

Handlers for receiving and responding to local control commands for getting and setting properties.

typedef struct esp_local_ctrl_transport **esp_local_ctrl_transport_t**

Transport mode (BLE / HTTPD) over which the service will be provided.

This is forward declaration of a private structure, implemented internally by `esp_local_ctrl`.

typedef struct protocomm_ble_config **esp_local_ctrl_transport_config_ble_t**

Configuration for transport mode BLE.

This is a forward declaration for `protocomm_ble_config_t`. To use this, application must set CONFIG_BT_ENABLED and include `protocomm_ble.h`.

typedef struct *httpd_config* **esp_local_ctrl_transport_config_httpd_t**

Configuration for transport mode HTTPD.

This is a forward declaration for `httpd_ssl_config_t` (for HTTPS) or `httpd_config_t` (for HTTP)

typedef enum *esp_local_ctrl_proto_sec* **esp_local_ctrl_proto_sec_t**

Security types for esp_local_control.

typedef *protocomm_security1_params_t* **esp_local_ctrl_security1_params_t**

typedef *protocomm_security2_params_t* **esp_local_ctrl_security2_params_t**

typedef struct *esp_local_ctrl_proto_sec_cfg* **esp_local_ctrl_proto_sec_cfg_t**

Protocom security configs

typedef struct *esp_local_ctrl_config* **esp_local_ctrl_config_t**

Configuration structure to pass to `esp_local_ctrl_start()`

Enumerations

enum **esp_local_ctrl_proto_sec**

Security types for esp_local_control.

Values:

enumerator **PROTOCOM_SEC0**

enumerator **PROTOCOM_SEC1**

enumerator **PROTOCOM_SEC2**

enumerator **PROTOCOM_SEC_CUSTOM**

2.2.7 ESP 串行从机链路

概述

乐鑫有多款芯片可用作从机的芯片。这些从机依赖于一些通用总线，并在总线上实现了各自的通信协议。`esp_serial_slave_link` 组件能让主机通过总线驱动和相应的协议与 ESP 从机进行通信。

`esp_serial_slave_link` 设备初始化完成后，应用程序就能通过它与 ESP 从机方便地通信。

备注： `esp_serial_slave_link` 组件自 ESP-IDF 版本 v5.0 起移到了单独的仓库：

- ESSL component on GitHub

运行 `idf.py add-dependency espressif/esp_serial_slave_link` 将 ESSL 组件添加到你的项目中。

乐鑫设备协议

如需了解关于乐鑫设备协议详情，请参考以下文档：

ESP SPI 从机 HD（半双工）模式协议

乐鑫芯片的 SPI 从机功能支持概况

	ESP32	ESP32-S2	ESP32-C3	ESP32-S3	ESP32-C2	ESP32-C6	ESP32-H2	ESP32-P4
SPI 从机 HD	N	Y (v2)	Y (v2)	Y (v2)	Y (v2)	Y (v2)	Y (v2)	Y (v2)
Tohost intr		N	N	N	N	N	N	N
Frhost intr		2 *	2 *	2 *	2 *	2 *	2 *	2 *
TX DMA		Y	Y	Y	Y	Y	Y	Y
RX DMA		Y	Y	Y	Y	Y	Y	Y
共享寄存器		72	64	64	64	64	64	64

概述 在半双工模式下，主机须使用从机定义的协议与从机通信。每个传输事务按顺序可能包括以下阶段：

- 命令阶段：8 位，主机到从机
  此阶段决定事务的其它阶段。参见支持的命令。
- 地址阶段：8 位，主机到从机，可选
  对于某些命令（如 WRBUF、RDBUF），此阶段指定要写入/读取的共享寄存器地址。对于其他包括此阶段的命令，这没有实际意义，但仍必须存在于事务中。
- Dummy 阶段：8 位，浮动，可选
  此阶段是主机和从机在总线上的周转时间，并为从机向主机发送数据提供了充分的准备时间。
- 数据阶段：长度可变，方向由命令确定。
  这一阶段可以输出数据（OUT，方向为由从机向主机），或者输入数据（IN，方向为由主机向从机）。

方向是指哪一方（主机或从机）控制 MOSI、MISO、WP 和 HD 管脚。

数据 IO 模式 在某些 IO 模式下，可以使用更多数据线来发送数据。因此，与 1 位模式相比，发送相同数据量所需的 SPI 时钟周期更少。例如，在 QIO 模式下，地址和数据（IN 和 OUT）应发送到全部 4 条数据线上（MOSI、MISO、WP 和 HD）。下表展示了 ESP32-S2 SPI 从机支持的模式，以及相应模式下使用的数据线数量。

Mode	命令线数	地址线数	dummy 线数	数据线数
1-bit	1	1	1	1
DOUT	1	1	4	2
DIO	1	2	4	2
QOUT	1	1	4	4
QIO	1	4	4	4
QPI	4	4	4	4

除 QPI 模式外，使用哪种模式通常取决于主机发送的命令（请参考支持的命令）。

QPI 模式 QPI 模式是 SPI 从机的一种特殊状态。主机可发送 ENQPI 命令，使从机进入 QPI 模式。在 QPI 模式下，命令以 4 位形式发送，因此与其他正常模式不兼容。只有在从机处于 QPI 模式时，主机才能发送 QPI 命令。主机可发送 EXQPI 命令退出 QPI 模式。

支持的命令

备注： 命令名称是从主机视角确定的。例如，WRBUF 表示由主机向从机的缓冲区写入。

名称	描述	命令	地址	数据
WRBUF	写入缓冲区	0x01	Buf addr	主到从，不超过缓冲区大小
RDBUF	读取缓冲区	0x02	Buf addr	从到主，不超过缓冲区大小
WRDMA	写入 DMA	0x03	8 位	主到从，不超过从机提供的长度
RDDMA	读取 DMA	0x04	8 位	从到主，不超过从机提供的长度
SEG_DONE	段完成	0x05	•	•
ENQPI	进入 QPI 模式	0x06	•	•
WR_DONE	写入段完成	0x07	•	•
CMD8	中断	0x08	•	•
CMD9	中断	0x09	•	•
CMDA	中断	0x0A	•	•
EXQPI	退出 QPI 模式	0xDD	•	•

此外，WRBUF、RDBUF、WRDMA 和 RDDMA 命令都有 2 位和 4 位版本。要在 2 位或 4 位模式下操作，请用下表中的对应命令掩码与原始命令按位或 (bit OR) 后发送。例如，命令 0xA1 表示 QIO 模式下的 WRBUF。

模式	掩码
1-bit	0x00
DOUT	0x10
DIO	0x50
QOUT	0x20
QIO	0xA0
QPI	0xA0

段事务模式 目前，SPI 从机 HD 驱动程序仅支持段事务模式。在此模式下，对于从机加载到 DMA 的事务，主机可以分段读取或写入。这样，主机就无需准备与从机数据大小相同的大缓冲区。主机在一个缓冲区的读取/写入完成后，须向从机发送相应的终止命令作为同步信号。在从机收到终止命令后，从机驱动程序会将新数据（如有）更新到 DMA 上。

WRDMA 的终止命令是 WR_DONE (0x07)，RDDMA 的终止命令是 CMD8 (0x08)。

以下是主机自从机 DMA 读取数据的流程示例：

1. 从机将 4092 字节数据加载到 RDDMA。
2. 主机进行七次 RDDMA 事务，每个事务长 512 字节，并自从机读取前 3584 字节。
3. 主机进行最后一次 RDDMA 事务，长度为 512 字节（长度可以与从机相同、更长或更短）。前 508 字节是从机发送的有效数据，最后 4 字节无意义。
4. 主机向从机发送 CMD8。
5. 从机将其他的 4092 字节数据加载到 RDDMA。
6. 主机发送 CMD8 后，可以开始新的读取事务。

术语解释

- ESSL：ESP 串行从机链路 (ESP Serial Slave Link)，即本文档描述的组件。
- 主机：运行 `esp_serial_slave_link` 组件的设备。
- ESSL 设备：主机上的虚拟设备，关联到一个 ESP 从机，其设备上下文中具有如何通过总线驱动和从机的总线协议与其通信的信息。
- ESSL 设备句柄：ESSL 设备上下文的一个句柄，包含配置信息、状态信息和 ESSL 组件所需的数据。设备上下文中储存了驱动配置、通信状态和主从机共享的数据等。
  - 在使用前，应将上下文初始化；如不再使用，应该反初始化。主机应用程序通过这一句柄操作 ESSL 设备。
- ESP 从机：连接到总线的从机，ESSL 组件的通信对象。
- 总线：特指用于主机和从机相互通信的总线。
- 从机协议：Espressif 硬件和软件在总线上使用的特殊通信协议。
- TX buffer num：计数器，位于从机，可由主机读取。指示由从机加载到硬件上、用于接收主机数据的累计 buffer 总数。
- RX data size：计数器，位于从机，可由主机读取。指示由从机加载到硬件上、需发送给主机的累计数据大小。

ESP 从机提供的服务

Espressif 从机提供下列服务：

1. Tohost 中断：从机可通过中断线向主机通知某些事件。（可选）
2. Frhost 中断：主机可向从机通知某些事件。
3. TX FIFO（主机到从机）：从机能够以接收 buffer 为单位，接收主机发送的数据。
   从机通过更新 TX buffer num 的方式，将可以接收多少数据的信息通知主机。主机读取 TX buffer num，减去已使用的 buffer 数，得到剩余 buffer 数量。
4. RX FIFO（从机到主机）：从机可向主机发送数据流。SDIO 从机也可通过中断线通知主机，从机有待发送的新数据。
   从机通过更新 RX data size，将准备发送的数据大小通知主机。主机读取该数据大小，减去已接收的数据长度，得到剩余数据大小。
5. 共享寄存器：主机可以读取从机寄存器上的部分内容，也可写入从机寄存器供从机读取。

从机提供的服务取决于从机的模型。如需了解详情，请参考*乐鑫芯片的 SPI 从机功能支持概况*。

初始化 ESP 串行从机链路

ESP SDIO 从机 ESP SDIO 从机链路 (ESSL SDIO) 设备依赖于 SDMMC 组件。它可通过 SDMMC Host 或 SDSPI Host 功能均可与 ESP SDIO 从机通信。ESSL 设备初始化步骤如下：

1. 初始化 SDMMC 卡（参考*SDMMC 驱动相关文档*）结构体。
2. 调用 *`sdmmc_card_init()`* 初始化该卡。
3. 用 *`essl_sdio_config_t`* 初始化 ESSL 设备。其中，`card` 成员应为第二步中的 `sdmmc_card_t`，`recv_buffer_size` 成员应填写为预先协商的值。
4. 调用 *`essl_init()`* 对 SDIO 部分进行初始化。
5. 调用 *`essl_wait_for_ready()`* 等待从机就绪。

ESP SPI 从机

备注： 如果你正在用 SPI 接口与 ESP SDIO 从机进行通信，则应该用*SDIO 接口* 代替。

暂不支持。

API

初始化完成后，你可调用以下 API 使用从机提供的服务：

Tohost 中断（可选）

1. 调用*essl_get_intr_ena()* 了解哪些事件触发了对主机的中断。
2. 调用*essl_set_intr_ena()* 设置能够触发主机中断的事件。
3. 调用*essl_wait_int()* 等待从机中断或超时。
4. 中断触发后，调用*essl_get_intr()* 了解哪些事件处于活跃状态，并调用*essl_clear_intr()* 将其清空。

Frhost 中断

1. 调用*essl_send_slave_intr()* 触发从机的通用中断。

TX FIFO

1. 调用*essl_get_tx_buffer_num()* 了解从机准备用于接收主机数据的 buffer 数。你可选择是否调用该函数。主机向从机发送数据包前，也会轮询 `tx_buffer_num`，直到从机的 buffer 数量足够或超时。
2. 调用*essl_send_packet()* 向从机发送数据。

RX FIFO

1. 调用*essl_get_rx_data_size()* 了解从机需发送给主机的数据大小。你可选择是否调用该函数。当主机尝试接收数据时，如果目前的 `rx_data_size` 小于主机准备接收数据的 buffer 大小，就会对 `rx_data_size` 进行一次更新。如果 `rx_data_size` 一直为 0，主机可能会轮询 `rx_data_size` 直到超时。
2. 调用*essl_get_packet()* 接收来自从机的数据。

重置计数器（可选） 如果从机计数器已重置，调用*essl_reset_cnt()* 重置内部计数器。

应用示例

以下示例展示了 ESP32-P4 SDIO 主机如何与从机互相通信，其中主机使用了 ESSL SDIO：

peripherals/sdio

如需了解详情，请参考 README.md 中的示例。

API 参考

Header File

- components/driver/test_apps/components/esp_serial_slave_link/include/esp_serial_slave_link/essl.h

Functions

esp_err_t **essl_init**(*essl_handle_t* handle, uint32_t wait_ms)

Initialize the slave.

参数

- **handle** -- Handle of an ESSL device.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: If success
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- Other value returned from lower layer `init`.

esp_err_t **essl_wait_for_ready**(*essl_handle_t* handle, uint32_t wait_ms)

Wait for interrupt of an ESSL slave device.

参数

- **handle** -- Handle of an ESSL device.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: If success
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- One of the error codes from SDMMC host controller

esp_err_t **essl_get_tx_buffer_num**(*essl_handle_t* handle, uint32_t *out_tx_num, uint32_t wait_ms)

Get buffer num for the host to send data to the slave. The buffers are size of `buffer_size`.

参数

- **handle** -- Handle of a ESSL device.
- **out_tx_num** -- Output of buffer num that host can send data to ESSL slave.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode
- One of the error codes from SDMMC/SPI host controller

esp_err_t **essl_get_rx_data_size**(*essl_handle_t* handle, uint32_t *out_rx_size, uint32_t wait_ms)

Get the size, in bytes, of the data that the ESSL slave is ready to send

参数

- **handle** -- Handle of an ESSL device.
- **out_rx_size** -- Output of data size to read from slave, in bytes
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode
- One of the error codes from SDMMC/SPI host controller

esp_err_t **essl_reset_cnt**(*essl_handle_t* handle)

Reset the counters of this component. Usually you don't need to do this unless you know the slave is reset.

参数 **handle** -- Handle of an ESSL device.

返回

- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode
- ESP_ERR_INVALID_ARG: Invalid argument, handle is not init.

esp_err_t **essl_send_packet**(*essl_handle_t* handle, const void *start, size_t length, uint32_t wait_ms)

Send a packet to the ESSL Slave. The Slave receives the packet into buffers whose size is `buffer_size` (configured during initialization).

参数

- **handle** -- Handle of an ESSL device.
- **start** -- Start address of the packet to send

- **length** -- Length of data to send, if the packet is over-size, the it will be divided into blocks and hold into different buffers automatically.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG: Invalid argument, handle is not init or other argument is not valid.
- ESP_ERR_TIMEOUT: No buffer to use, or error ftrom SDMMC host controller.
- ESP_ERR_NOT_FOUND: Slave is not ready for receiving.
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode
- One of the error codes from SDMMC/SPI host controller.

esp_err_t **essl_get_packet**(*essl_handle_t* handle, void *out_data, size_t size, size_t *out_length, uint32_t wait_ms)

Get a packet from ESSL slave.

参数

- **handle** -- Handle of an ESSL device.
- **out_data** -- **[out]** Data output address
- **size** -- The size of the output buffer, if the buffer is smaller than the size of data to receive from slave, the driver returns `ESP_ERR_NOT_FINISHED`
- **out_length** -- **[out]** Output of length the data actually received from slave.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK Success: All the data has been read from the slave.
- ESP_ERR_INVALID_ARG: Invalid argument, The handle is not initialized or the other arguments are invalid.
- ESP_ERR_NOT_FINISHED: Read was successful, but there is still data remaining.
- ESP_ERR_NOT_FOUND: Slave is not ready to send data.
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode
- One of the error codes from SDMMC/SPI host controller.

esp_err_t **essl_write_reg**(*essl_handle_t* handle, uint8_t addr, uint8_t value, uint8_t *value_o, uint32_t wait_ms)

Write general purpose R/W registers (8-bit) of ESSL slave.

备注: sdio 28-31 are reserved, the lower API helps to skip.

参数

- **handle** -- Handle of an ESSL device.
- **addr** -- Address of register to write. For SDIO, valid address: 0-59. For SPI, see `essl_spi.h`
- **value** -- Value to write to the register.
- **value_o** -- Output of the returned written value.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK Success
- One of the error codes from SDMMC/SPI host controller

esp_err_t **essl_read_reg**(*essl_handle_t* handle, uint8_t add, uint8_t *value_o, uint32_t wait_ms)

Read general purpose R/W registers (8-bit) of ESSL slave.

参数

- **handle** -- Handle of a `essl` device.
- **add** -- Address of register to read. For SDIO, Valid address: 0-27, 32-63 (28-31 reserved, return interrupt bits on read). For SPI, see `essl_spi.h`
- **value_o** -- Output value read from the register.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK Success
- One of the error codes from SDMMC/SPI host controller

esp_err_t **essl_wait_int**(*essl_handle_t* handle, uint32_t wait_ms)

wait for an interrupt of the slave

参数

- **handle** -- Handle of an ESSL device.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: If interrupt is triggered.
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- ESP_ERR_TIMEOUT: No interrupts before timeout.

esp_err_t **essl_clear_intr**(*essl_handle_t* handle, uint32_t intr_mask, uint32_t wait_ms)

Clear interrupt bits of ESSL slave. All the bits set in the mask will be cleared, while other bits will stay the same.

参数

- **handle** -- Handle of an ESSL device.
- **intr_mask** -- Mask of interrupt bits to clear.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- One of the error codes from SDMMC host controller

esp_err_t **essl_get_intr**(*essl_handle_t* handle, uint32_t *intr_raw, uint32_t *intr_st, uint32_t wait_ms)

Get interrupt bits of ESSL slave.

参数

- **handle** -- Handle of an ESSL device.
- **intr_raw** -- Output of the raw interrupt bits. Set to NULL if only masked bits are read.
- **intr_st** -- Output of the masked interrupt bits. set to NULL if only raw bits are read.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: Success
- ESP_INVALID_ARG: If both `intr_raw` and `intr_st` are NULL.
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- One of the error codes from SDMMC host controller

esp_err_t **essl_set_intr_ena**(*essl_handle_t* handle, uint32_t ena_mask, uint32_t wait_ms)

Set interrupt enable bits of ESSL slave. The slave only sends interrupt on the line when there is a bit both the raw status and the enable are set.

参数

- **handle** -- Handle of an ESSL device.
- **ena_mask** -- Mask of the interrupt bits to enable.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- One of the error codes from SDMMC host controller

esp_err_t **essl_get_intr_ena**(*essl_handle_t* handle, uint32_t *ena_mask_o, uint32_t wait_ms)

Get interrupt enable bits of ESSL slave.

参数

- **handle** -- Handle of an ESSL device.
- **ena_mask_o** -- Output of interrupt bit enable mask.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK Success
- One of the error codes from SDMMC host controller

esp_err_t **essl_send_slave_intr**(*essl_handle_t* handle, uint32_t intr_mask, uint32_t wait_ms)

Send interrupts to slave. Each bit of the interrupt will be triggered.

参数

- **handle** -- Handle of an ESSL device.
- **intr_mask** -- Mask of interrupt bits to send to slave.
- **wait_ms** -- Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- One of the error codes from SDMMC host controller

Type Definitions

typedef struct essl_dev_t ***essl_handle_t**

Handle of an ESSL device.

Header File

- components/driver/test_apps/components/esp_serial_slave_link/include/esp_serial_slave_link/essl_sdio.h

Functions

esp_err_t **essl_sdio_init_dev**(*essl_handle_t* *out_handle, const *essl_sdio_config_t* *config)

Initialize the ESSL SDIO device and get its handle.

参数

- **out_handle** -- Output of the handle.
- **config** -- Configuration for the ESSL SDIO device.

返回

- ESP_OK: on success
- ESP_ERR_NO_MEM: memory exhausted.

esp_err_t **essl_sdio_deinit_dev**(*essl_handle_t* handle)

Deinitialize and free the space used by the ESSL SDIO device.

参数 **handle** -- Handle of the ESSL SDIO device to deinit.

返回

- ESP_OK: on success
- ESP_ERR_INVALID_ARG: wrong handle passed

Structures

struct **essl_sdio_config_t**

Configuration for the ESSL SDIO device.

Public Members

sdmmc_card_t ***card**

The initialized sdmmc card pointer of the slave.

int **recv_buffer_size**

The pre-negotiated recv buffer size used by both the host and the slave.

Header File

- components/driver/test_apps/components/esp_serial_slave_link/include/esp_serial_slave_link/essl_spi.h

Functions

esp_err_t **essl_spi_init_dev**(*essl_handle_t* *out_handle, const *essl_spi_config_t* *init_config)

Initialize the ESSL SPI device function list and get its handle.

参数

- **out_handle** -- **[out]** Output of the handle
- **init_config** -- Configuration for the ESSL SPI device

返回

- ESP_OK: On success
- ESP_ERR_NO_MEM: Memory exhausted
- ESP_ERR_INVALID_STATE: SPI driver is not initialized
- ESP_ERR_INVALID_ARG: Wrong register ID

esp_err_t **essl_spi_deinit_dev**(*essl_handle_t* handle)

Deinitialize the ESSL SPI device and free the memory used by the device.

参数 **handle** -- Handle of the ESSL SPI device

返回

- ESP_OK: On success
- ESP_ERR_INVALID_STATE: ESSL SPI is not in use

esp_err_t **essl_spi_read_reg**(void *arg, uint8_t addr, uint8_t *out_value, uint32_t wait_ms)

Read from the shared registers.

备注: The registers for Master/Slave synchronization are reserved. Do not use them. (see `rx_sync_reg` in *`essl_spi_config_t`*)

参数

- **arg** -- Context of the component. (Member `arg` from `essl_handle_t`)
- **addr** -- Address of the shared registers. (Valid: 0 ~ SOC_SPI_MAXIMUM_BUFFER_SIZE, registers for M/S sync are reserved, see note1).
- **out_value** -- **[out]** Read buffer for the shared registers.
- **wait_ms** -- Time to wait before timeout (reserved for future use, user should set this to 0).

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: ESSL SPI has not been initialized.
- ESP_ERR_INVALID_ARG: The address argument is not valid. See note 1.
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_get_packet**(void *arg, void *out_data, size_t size, uint32_t wait_ms)

Get a packet from Slave.

参数

- **arg** -- Context of the component. (Member `arg` from `essl_handle_t`)
- **out_data** -- **[out]** Output data address
- **size** -- The size of the output data.
- **wait_ms** -- Time to wait before timeout (reserved for future use, user should set this to 0).

返回

- ESP_OK: On Success
- ESP_ERR_INVALID_STATE: ESSL SPI has not been initialized.
- ESP_ERR_INVALID_ARG: The output data address is neither DMA capable nor 4 byte-aligned

- ESP_ERR_INVALID_SIZE: Master requires `size` bytes of data but Slave did not load enough bytes.

esp_err_t **essl_spi_write_reg**(void *arg, uint8_t addr, uint8_t value, uint8_t *out_value, uint32_t wait_ms)

Write to the shared registers.

备注： The registers for Master/Slave synchronization are reserved. Do not use them. (see `tx_sync_reg` in *essl_spi_config_t*)

备注： Feature of checking the actual written value (`out_value`) is not supported.

参数

- **arg** -- Context of the component. (Member `arg` from `essl_handle_t`)
- **addr** -- Address of the shared registers. (Valid: 0 ~ SOC_SPI_MAXIMUM_BUFFER_SIZE, registers for M/S sync are reserved, see note1)
- **value** -- Buffer for data to send, should be align to 4.
- **out_value** -- **[out]** Not supported, should be set to NULL.
- **wait_ms** -- Time to wait before timeout (reserved for future use, user should set this to 0).

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: ESSL SPI has not been initialized.
- ESP_ERR_INVALID_ARG: The address argument is not valid. See note 1.
- ESP_ERR_NOT_SUPPORTED: Should set `out_value` to NULL. See note 2.
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_send_packet**(void *arg, const void *data, size_t size, uint32_t wait_ms)

Send a packet to Slave.

参数

- **arg** -- Context of the component. (Member `arg` from `essl_handle_t`)
- **data** -- Address of the data to send
- **size** -- Size of the data to send.
- **wait_ms** -- Time to wait before timeout (reserved for future use, user should set this to 0).

返回

- ESP_OK: On success
- ESP_ERR_INVALID_STATE: ESSL SPI has not been initialized.
- ESP_ERR_INVALID_ARG: The data address is not DMA capable
- ESP_ERR_INVALID_SIZE: Master will send `size` bytes of data but Slave did not load enough RX buffer

void **essl_spi_reset_cnt**(void *arg)

Reset the counter in Master context.

备注： Shall only be called if the slave has reset its counter. Else, Slave and Master would be desynchronized

---

参数 **arg** -- Context of the component. (Member `arg` from `essl_handle_t`)

esp_err_t **essl_spi_rdbuf**(*spi_device_handle_t* spi, uint8_t *out_data, int addr, int len, uint32_t flags)

Read the shared buffer from the slave in ISR way.

备注： The slave's HW doesn't guarantee the data in one SPI transaction is consistent. It sends data in unit of byte. In other words, if the slave SW attempts to update the shared register when a rdbuf SPI transaction is in-flight, the data got by the master will be the combination of bytes of different writes of slave SW.

备注： `out_data` should be prepared in words and in the DRAM. The buffer may be written in words by the DMA. When a byte is written, the remaining bytes in the same word will also be overwritten, even the `len` is shorter than a word.

参数

- **spi** -- SPI device handle representing the slave
- **out_data** -- **[out]** Buffer for read data, strongly suggested to be in the DRAM and aligned to 4
- **addr** -- Address of the slave shared buffer
- **len** -- Length to read
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: on success
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_rdbuf_polling**(*spi_device_handle_t* spi, uint8_t *out_data, int addr, int len, uint32_t flags)

Read the shared buffer from the slave in polling way.

备注： `out_data` should be prepared in words and in the DRAM. The buffer may be written in words by the DMA. When a byte is written, the remaining bytes in the same word will also be overwritten, even the `len` is shorter than a word.

参数

- **spi** -- SPI device handle representing the slave
- **out_data** -- **[out]** Buffer for read data, strongly suggested to be in the DRAM and aligned to 4
- **addr** -- Address of the slave shared buffer
- **len** -- Length to read
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: on success
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_wrbuf**(*spi_device_handle_t* spi, const uint8_t *data, int addr, int len, uint32_t flags)

Write the shared buffer of the slave in ISR way.

备注： `out_data` should be prepared in words and in the DRAM. The buffer may be written in words by the DMA. When a byte is written, the remaining bytes in the same word will also be overwritten, even the `len` is shorter than a word.

参数

- **spi** -- SPI device handle representing the slave
- **data** -- Buffer for data to send, strongly suggested to be in the DRAM
- **addr** -- Address of the slave shared buffer,
- **len** -- Length to write
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: success
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_wrbuf_polling**(*spi_device_handle_t* spi, const uint8_t *data, int addr, int len, uint32_t flags)

Write the shared buffer of the slave in polling way.

备注: `out_data` should be prepared in words and in the DRAM. The buffer may be written in words by the DMA. When a byte is written, the remaining bytes in the same word will also be overwritten, even the `len` is shorter than a word.

参数

- **spi** -- SPI device handle representing the slave
- **data** -- Buffer for data to send, strongly suggested to be in the DRAM
- **addr** -- Address of the slave shared buffer,
- **len** -- Length to write
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: success
- or other return value from :cpp:func:`spi_device_polling_transmit`.

esp_err_t **essl_spi_rddma**(*spi_device_handle_t* spi, uint8_t *out_data, int len, int seg_len, uint32_t flags)

Receive long buffer in segments from the slave through its DMA.

备 注: This function combines several :cpp:func:`essl_spi_rddma_seg` and one :cpp:func:`essl_spi_rddma_done` at the end. Used when the slave is working in segment mode.

参数

- **spi** -- SPI device handle representing the slave
- **out_data** -- **[out]** Buffer to hold the received data, strongly suggested to be in the DRAM and aligned to 4
- **len** -- Total length of data to receive.
- **seg_len** -- Length of each segment, which is not larger than the maximum transaction length allowed for the spi device. Suggested to be multiples of 4. When set < 0, means send all data in one segment (the `rddma_done` will still be sent.)
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: success
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_rddma_seg**(*spi_device_handle_t* spi, uint8_t *out_data, int seg_len, uint32_t flags)

Read one data segment from the slave through its DMA.

备注: To read long buffer, call :cpp:func:`essl_spi_rddma` instead.

参数

- **spi** -- SPI device handle representing the slave
- **out_data** -- **[out]** Buffer to hold the received data. strongly suggested to be in the DRAM and aligned to 4
- **seg_len** -- Length of this segment
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: success
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_rddma_done**(*spi_device_handle_t* spi, uint32_t flags)

Send the `rddma_done` command to the slave. Upon receiving this command, the slave will stop sending the current buffer even there are data unsent, and maybe prepare the next buffer to send.

备注: This is required only when the slave is working in segment mode.

参数

- **spi** -- SPI device handle representing the slave
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: success
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_wrdma**(*spi_device_handle_t* spi, const uint8_t *data, int len, int seg_len, uint32_t flags)

Send long buffer in segments to the slave through its DMA.

备 注: This function combines several :cpp:func:`essl_spi_wrdma_seg` and one :cpp:func:`essl_spi_wrdma_done` at the end. Used when the slave is working in segment mode.

参数

- **spi** -- SPI device handle representing the slave
- **data** -- Buffer for data to send, strongly suggested to be in the DRAM
- **len** -- Total length of data to send.
- **seg_len** -- Length of each segment, which is not larger than the maximum transaction length allowed for the spi device. Suggested to be multiples of 4. When set < 0, means send all data in one segment (the `wrdma_done` will still be sent.)
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: success
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_wrdma_seg**(*spi_device_handle_t* spi, const uint8_t *data, int seg_len, uint32_t flags)

Send one data segment to the slave through its DMA.

备注: To send long buffer, call :cpp:func:`essl_spi_wrdma` instead.

参数

- **spi** -- SPI device handle representing the slave
- **data** -- Buffer for data to send, strongly suggested to be in the DRAM
- **seg_len** -- Length of this segment
- **flags** -- `SPI_TRANS_*` flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: success
- or other return value from :cpp:func:`spi_device_transmit`.

esp_err_t **essl_spi_wrdma_done**(*spi_device_handle_t* spi, uint32_t flags)

Send the `wrdma_done` command to the slave. Upon receiving this command, the slave will stop receiving, process the received data, and maybe prepare the next buffer to receive.

备注： This is required only when the slave is working in segment mode.

参数

- **spi** -- SPI device handle representing the slave
- **flags** -- SPI_TRANS_* flags to control the transaction mode of the transaction to send.

返回

- ESP_OK: success
- or other return value from :cpp:func:spi_device_transmit.

Structures

struct **essl_spi_config_t**

Configuration of ESSL SPI device.

Public Members

spi_device_handle_t ***spi**

Pointer to SPI device handle.

uint32_t **tx_buf_size**

The pre-negotiated Master TX buffer size used by both the host and the slave.

uint8_t **tx_sync_reg**

The pre-negotiated register ID for Master-TX-SLAVE-RX synchronization. 1 word (4 Bytes) will be reserved for the synchronization.

uint8_t **rx_sync_reg**

The pre-negotiated register ID for Master-RX-Slave-TX synchronization. 1 word (4 Bytes) will be reserved for the synchronization.

2.2.8 ESP x509 证书包

概述

ESP x509 证书包 API 提供了一种简便的方法，帮助你安装自定义 x509 根证书用于 TLS 服务器验证。

备注： 目前在使用 WolfSSL 时该证书包不可用。

该证书包中包括 Mozilla NSS 根证书存储区的完整根证书列表。使用 gen_crt_bundle.py python 程序，可将证书的主题名称和公钥存储在文件中，并嵌入 ESP32-P4 二进制文件。

生成证书包时，你需选择：

- 来自 Mozilla 的完整根证书包，包含超过 130 份证书。目前提供的证书包更新于 2024 年 3 月 11 日，星期一，15:25:27 (GMT)。
- 一组预先筛选的常用根证书。其中仅包含约 41 份证书，但根据 SSL 证书颁发机构统计数据，其绝对使用率约达到 90%，市场覆盖率约达 99%。

此外，还可指定证书文件的路径或包含证书的目录，将其他证书添加到生成的证书包中。

备注： 信任所有根证书意味着如果任何证书被收回，就必须更新证书列表，包括从 `cacrt_all.pem` 中将其删除。

配置

多数配置可通过 menuconfig 完成。CMake 会根据配置信息生成及嵌入证书包。

- *CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*：自动创建并附加证书包。
- *CONFIG_MBEDTLS_DEFAULT_CERTIFICATE_BUNDLE*：决定添加证书列表中的哪些证书。
- *CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE_PATH*：指定要在证书包中嵌入的其他证书的路径。

要在使用 ESP-TLS 时启用证书包，将函数指针指向证书包的 attach 函数：

```
esp_tls_cfg_t cfg = {
     .crt_bundle_attach = esp_crt_bundle_attach,
};
```

此步骤是为了避免在用户未使能的情况下嵌入证书包。

如果直接使用 mbedTLS 包，在设置阶段直接调用 attach 函数可以激活证书包：

```
mbedtls_ssl_config conf;
mbedtls_ssl_config_init(&conf);

esp_crt_bundle_attach(&conf);
```

生成根证书列表

根证书列表来自 Mozilla 的 NSS 根证书商店。

运行 `mk-ca-bundle.pl` 脚本可下载和创建列表。脚本发布于 curl。

还可通过 curl 网站直接下载完整列表：从 Mozilla 提取的 CA 证书。

常用证书包是通过筛选出市场份额超过 1% 的授权机构来决定的，筛选数据来自 w3tech 的 SSL Survey。

根据这些授权机构，从 Mozilla 提供的 列表 的 `cmn_crt_authorities.csv` 中筛选证书名称。

更新证书包

证书包嵌入到应用程序中，通过 OTA 更新与应用程序一起更新。如果想使用比目前 ESP-IDF 中的证书包更新的包，则可按照生成根证书列表 中的说明从 Mozilla 下载证书列表。

定期同步

证书包会与 Mozilla 的 NSS 根证书商店定期同步。在 ESP-IDF 的次要版本或补丁版本中，为了保证兼容性，会将上游证书包中已弃用的证书添加到弃用列表。如有需要，可以通过*CONFIG_MBEDTLS_CERTIFICATE_BUNDLE_DEPRECATED_LIST* 将弃用证书加入默认证书包。这些弃用证书将在下一个 ESP-IDF 主要版本中移除。

应用示例

使用 ESP-TLS 创建安全套接字连接的简单 HTTPS 示例：protocols/https_x509_bundle，该示例使用了证书包并添加了两个自定义证书用于验证。

使用 ESP-TLS 和默认证书包的 HTTPS 示例：protocols/https_request。

使用 mbedTLS 和默认证书包的 HTTPS 示例：protocols/https_mbedtls。

API 参考

Header File

- components/mbedtls/esp_crt_bundle/include/esp_crt_bundle.h
- This header file can be included with:

```
#include "esp_crt_bundle.h"
```

- This header file is a part of the API provided by the `mbedtls` component. To declare that your component depends on `mbedtls`, add the following to your CMakeLists.txt:

```
REQUIRES mbedtls
```

or

```
PRIV_REQUIRES mbedtls
```

Functions

esp_err_t **esp_crt_bundle_attach**(void *conf)

Attach and enable use of a bundle for certificate verification.

Attach and enable use of a bundle for certificate verification through a verification callback. If no specific bundle has been set through esp_crt_bundle_set() it will default to the bundle defined in menuconfig and embedded in the binary.

参数 **conf** -- **[in]** The config struct for the SSL connection.

返回

- ESP_OK if adding certificates was successful.
- Other if an error occurred or an action must be taken by the calling process.

void **esp_crt_bundle_detach**(mbedtls_ssl_config *conf)

Disable and dealloc the certification bundle.

Removes the certificate verification callback and deallocates used resources

参数 **conf** -- **[in]** The config struct for the SSL connection.

esp_err_t **esp_crt_bundle_set**(const uint8_t *x509_bundle, size_t bundle_size)

Set the default certificate bundle used for verification.

Overrides the default certificate bundle only in case of successful initialization. In most use cases the bundle should be set through menuconfig. The bundle needs to be sorted by subject name since binary search is used to find certificates.

参数

- **x509_bundle** -- **[in]** A pointer to the certificate bundle.
- **bundle_size** -- **[in]** Size of the certificate bundle in bytes.

返回

- ESP_OK if adding certificates was successful.
- Other if an error occurred or an action must be taken by the calling process.

bool **esp_crt_bundle_in_use**(const mbedtls_x509_crt *ca_chain)

Check if the given CA certificate chain is the default "dummy" certificate chain attached by the esp_crt_bundle.

参数 **ca_chain** -- A pointer to the CA chain.
返回 true if the ca_chain is the dummy CA chain attached by esp_crt_bundle
返回 false otherwise

2.2.9 HTTP 服务器

概述

HTTP Server 组件提供了在 ESP32 上运行轻量级 Web 服务器的功能，下面介绍使用 HTTP Server 组件 API 的详细步骤：

- *httpd_start()*：创建 HTTP 服务器的实例，根据具体的配置为其分配内存和资源，并返回该服务器实例的句柄。服务器使用了两个套接字，一个用来监听 HTTP 流量（TCP 类型），另一个用来处理控制信号（UDP 类型），它们在服务器的任务循环中轮流使用。通过向 `httpd_start()` 传递 `httpd_config_t` 结构体，可以在创建服务器实例时配置任务的优先级和堆栈的大小。TCP 流量被解析为 HTTP 请求，根据请求的 URI 来调用用户注册的处理程序，在处理程序中需要发送回 HTTP 响应数据包。
- *httpd_stop()*：根据传入的句柄停止服务器，并释放相关联的内存和资源。这是一个阻塞函数，首先给服务器任务发送停止信号，然后等待其终止。期间服务器任务会关闭所有已打开的连接，删除已注册的 URI 处理程序，并将所有会话的上下文数据重置为空。
- *httpd_register_uri_handler()*：通过传入 `httpd_uri_t` 结构体类型的对象来注册 URI 处理程序。该结构体包含如下成员：`uri` 名字，`method` 类型（比如 `HTTPD_GET/HTTPD_POST/HTTPD_PUT` 等等），`esp_err_t *handler (httpd_req_t *req)` 类型的函数指针，指向用户上下文数据的 `user_ctx` 指针。

应用示例

```
/* URI 处理函数，在客户端发起 GET /uri 请求时被调用 */
esp_err_t get_handler(httpd_req_t *req)
{
    /* 发送回简单的响应数据包 */
    const char resp[] = "URI GET Response";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

/* URI 处理函数，在客户端发起 POST/uri 请求时被调用 */
esp_err_t post_handler(httpd_req_t *req)
{
    /* 定义 HTTP POST 请求数据的目标缓存区
     * httpd_req_recv() 只接收 char* 数据，但也可以是
     * 任意二进制数据（需要类型转换）
     * 对于字符串数据，null 终止符会被省略，
     * content_len 会给出字符串的长度 */
    char content[100];

    /* 如果内容长度大于缓冲区则截断 */
    size_t recv_size = MIN(req->content_len, sizeof(content));

    int ret = httpd_req_recv(req, content, recv_size);
    if (ret <= 0) {  /* 返回 0 表示连接已关闭 */
        /* 检查是否超时 */
```

（下页继续）

(续上页)

```
        if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
            /* 如果是超时，可以调用 httpd_req_recv() 重试
             * 简单起见，这里我们直接
             * 响应 HTTP 408 (请求超时) 错误给客户端 */
            httpd_resp_send_408(req);
        }
        /* 如果发生了错误，返回 ESP_FAIL 可以确保
         * 底层套接字被关闭 */
        return ESP_FAIL;
    }

    /* 发送简单的响应数据包 */
    const char resp[] = "URI POST Response";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

/* GET /uri 的 URI 处理结构 */
httpd_uri_t uri_get = {
    .uri      = "/uri",
    .method   = HTTP_GET,
    .handler  = get_handler,
    .user_ctx = NULL
};

/* POST/uri 的 URI 处理结构 */
httpd_uri_t uri_post = {
    .uri      = "/uri",
    .method   = HTTP_POST,
    .handler  = post_handler,
    .user_ctx = NULL
};

/* 启动 Web 服务器的函数 */
httpd_handle_t start_webserver(void)
{
    /* 生成默认的配置参数 */
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    /* 置空 esp_http_server 的实例句柄 */
    httpd_handle_t server = NULL;

    /* 启动 httpd server */
    if (httpd_start(&server, &config) == ESP_OK) {
        /* 注册 URI 处理程序 */
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &uri_post);
    }
    /* 如果服务器启动失败，返回的句柄是 NULL */
    return server;
}

/* 停止 Web 服务器的函数 */
void stop_webserver(httpd_handle_t server)
{
    if (server) {
        /* 停止 httpd server */
        httpd_stop(server);
    }
}
```

简单 HTTP 服务器示例 请查看位于 protocols/http_server/simple 的 HTTP 服务器示例，该示例演示了如何处理任意内容长度的数据，读取请求头和 URL 查询参数，设置响应头。

HTTP 长连接

HTTP 服务器具有长连接的功能，允许重复使用同一个连接（会话）进行多次传输，同时保持会话的上下文数据。上下文数据可由处理程序动态分配，在这种情况下需要提前指定好自定义的回调函数，以便在连接/会话被关闭时释放这部分内存资源。

长连接示例

```
/* 自定义函数，用来释放上下文数据 */
void free_ctx_func(void *ctx)
{
    /* 也可以是 free 以外的代码逻辑 */
    free(ctx);
}

esp_err_t adder_post_handler(httpd_req_t *req)
{
    /* 若上下文中不存在会话，则新建一个 */
    if (! req->sess_ctx) {
        req->sess_ctx = malloc(sizeof(ANY_DATA_TYPE));   /*!< 指向上下文数据 */
        req->free_ctx = free_ctx_func;                   /*!< 释放上下文数据的函数
↪*/
    }

    /* 访问上下文数据 */
    ANY_DATA_TYPE *ctx_data = (ANY_DATA_TYPE *)req->sess_ctx;

    /* 响应 */
    ..............
    ..............
    ..............

    return ESP_OK;
}
```

详情请参考位于 protocols/http_server/persistent_sockets 的示例代码。

Websocket 服务器

HTTP 服务器组件提供 websocket 支持。可以在 menuconfig 中使用*CONFIG_HTTPD_WS_SUPPORT* 选项启用 websocket 功能。有关如何使用 websocket 功能，请参阅 protocols/http_server/ws_echo_server 目录下的示例代码。

事件处理

ESP HTTP 服务器有各种事件，当特定事件发生时，事件循环库 可以触发处理程序。必须使用*esp_event_handler_register()* 注册处理程序以便 ESP HTTP 服务器进行事件处理。

esp_http_server_event_id_t 包含 ESP HTTP 服务器可能发生的所有事件。

以下为事件循环中不同 ESP HTTP 服务器事件的预期数据类型：

- HTTP_SERVER_EVENT_ERROR : `httpd_err_code_t`
- HTTP_SERVER_EVENT_START : `NULL`
- HTTP_SERVER_EVENT_ON_CONNECTED : `int`
- HTTP_SERVER_EVENT_ON_HEADER : `int`

- HTTP_SERVER_EVENT_HEADERS_SENT : `int`
- HTTP_SERVER_EVENT_ON_DATA : `esp_http_server_event_data`
- HTTP_SERVER_EVENT_SENT_DATA : `esp_http_server_event_data`
- HTTP_SERVER_EVENT_DISCONNECTED : `int`
- HTTP_SERVER_EVENT_STOP : `NULL`

API 参考

Header File

- components/esp_http_server/include/esp_http_server.h
- This header file can be included with:

```
#include "esp_http_server.h"
```

- This header file is a part of the API provided by the `esp_http_server` component. To declare that your component depends on `esp_http_server`, add the following to your CMakeLists.txt:

```
REQUIRES esp_http_server
```

or

```
PRIV_REQUIRES esp_http_server
```

Functions

esp_err_t **httpd_start** (*httpd_handle_t* *handle, const *httpd_config_t* *config)

Starts the web server.

Create an instance of HTTP server and allocate memory/resources for it depending upon the specified configuration.

Example usage:

```
//Function for starting the webserver
httpd_handle_t start_webserver(void)
{
// Generate default configuration
httpd_config_t config = HTTPD_DEFAULT_CONFIG();

// Empty handle to http_server
     httpd_handle_t server = NULL;

// Start the httpd server
if (httpd_start(&server, &config) == ESP_OK) {
// Register URI handlers
          httpd_register_uri_handler(server, &uri_get);
          httpd_register_uri_handler(server, &uri_post);
     }
// If server failed to start, handle will be NULL
return server;
}
```

参数

- **config** -- **[in]** Configuration for new instance of the server
- **handle** -- **[out]** Handle to newly created instance of the server. NULL on error

返回

- ESP_OK : Instance created successfully
- ESP_ERR_INVALID_ARG : Null argument(s)
- ESP_ERR_HTTPD_ALLOC_MEM : Failed to allocate memory for instance
- ESP_ERR_HTTPD_TASK : Failed to launch server task

esp_err_t **httpd_stop**(*httpd_handle_t* handle)

Stops the web server.

Deallocates memory/resources used by an HTTP server instance and deletes it. Once deleted the handle can no longer be used for accessing the instance.

Example usage:

```
// Function for stopping the webserver
void stop_webserver(httpd_handle_t server)
{
// Ensure handle is non NULL
if (server != NULL) {
// Stop the httpd server
        httpd_stop(server);
    }
}
```

参数 **handle** -- **[in]** Handle to server returned by httpd_start

返回

- ESP_OK : Server stopped successfully
- ESP_ERR_INVALID_ARG : Handle argument is Null

esp_err_t **httpd_register_uri_handler**(*httpd_handle_t* handle, const *httpd_uri_t* *uri_handler)

Registers a URI handler.

Example usage:

```
esp_err_t my_uri_handler(httpd_req_t* req)
{
// Recv , Process and Send
    ....
    ....
    ....

// Fail condition
if (....) {
// Return fail to close session //
return ESP_FAIL;
    }

// On success
return ESP_OK;
}

// URI handler structure
httpd_uri_t my_uri {
    .uri      = "/my_uri/path/xyz",
    .method   = HTTPD_GET,
    .handler  = my_uri_handler,
    .user_ctx = NULL
};

// Register handler
if (httpd_register_uri_handler(server_handle, &my_uri) != ESP_OK) {
// If failed to register handler
   ....
}
```

备注： URI handlers can be registered in real time as long as the server handle is valid.

参数

- **handle** -- **[in]** handle to HTTPD server instance
- **uri_handler** -- **[in]** pointer to handler that needs to be registered

返回

- ESP_OK : On successfully registering the handler
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_HANDLERS_FULL : If no slots left for new handler
- ESP_ERR_HTTPD_HANDLER_EXISTS : If handler with same URI and method is already registered

esp_err_t **httpd_unregister_uri_handler**(*httpd_handle_t* handle, const char *uri, *httpd_method_t* method)

Unregister a URI handler.

参数

- **handle** -- **[in]** handle to HTTPD server instance
- **uri** -- **[in]** URI string
- **method** -- **[in]** HTTP method

返回

- ESP_OK : On successfully deregistering the handler
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_NOT_FOUND : Handler with specified URI and method not found

esp_err_t **httpd_unregister_uri**(*httpd_handle_t* handle, const char *uri)

Unregister all URI handlers with the specified uri string.

参数

- **handle** -- **[in]** handle to HTTPD server instance
- **uri** -- **[in]** uri string specifying all handlers that need to be deregisterd

返回

- ESP_OK : On successfully deregistering all such handlers
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_NOT_FOUND : No handler registered with specified uri string

esp_err_t **httpd_sess_set_recv_override**(*httpd_handle_t* hd, int sockfd, *httpd_recv_func_t* recv_func)

Override web server's receive function (by session FD)

This function overrides the web server's receive function. This same function is used to read HTTP request packets.

备注： This API is supposed to be called either from the context of

- an http session APIs where sockfd is a valid parameter
- a URI handler where sockfd is obtained using httpd_req_to_sockfd()

参数

- **hd** -- **[in]** HTTPD instance handle
- **sockfd** -- **[in]** Session socket FD
- **recv_func** -- **[in]** The receive function to be set for this session

返回

- ESP_OK : On successfully registering override
- ESP_ERR_INVALID_ARG : Null arguments

esp_err_t **httpd_sess_set_send_override** (*httpd_handle_t* hd, int sockfd, *httpd_send_func_t* send_func)

Override web server's send function (by session FD)

This function overrides the web server's send function. This same function is used to send out any response to any HTTP request.

备注: This API is supposed to be called either from the context of

- an http session APIs where sockfd is a valid parameter
- a URI handler where sockfd is obtained using httpd_req_to_sockfd()

参数

- **hd** -- **[in]** HTTPD instance handle
- **sockfd** -- **[in]** Session socket FD
- **send_func** -- **[in]** The send function to be set for this session

返回

- ESP_OK : On successfully registering override
- ESP_ERR_INVALID_ARG : Null arguments

esp_err_t **httpd_sess_set_pending_override** (*httpd_handle_t* hd, int sockfd, *httpd_pending_func_t* pending_func)

Override web server's pending function (by session FD)

This function overrides the web server's pending function. This function is used to test for pending bytes in a socket.

备注: This API is supposed to be called either from the context of

- an http session APIs where sockfd is a valid parameter
- a URI handler where sockfd is obtained using httpd_req_to_sockfd()

参数

- **hd** -- **[in]** HTTPD instance handle
- **sockfd** -- **[in]** Session socket FD
- **pending_func** -- **[in]** The receive function to be set for this session

返回

- ESP_OK : On successfully registering override
- ESP_ERR_INVALID_ARG : Null arguments

esp_err_t **httpd_req_async_handler_begin** (*httpd_req_t* *r, *httpd_req_t* **out)

Start an asynchronous request. This function can be called in a request handler to get a request copy that can be used on a async thread.

备注:

- This function is necessary in order to handle multiple requests simultaneously. See examples/async_requests for example usage.
- You must call httpd_req_async_handler_complete() when you are done with the request.

参数

- **r** -- **[in]** The request to create an async copy of
- **out** -- **[out]** A newly allocated request which can be used on an async thread

返回

- ESP_OK : async request object created

esp_err_t **httpd_req_async_handler_complete**(*httpd_req_t* *r)

Mark an asynchronous request as completed. This will.

- free the request memory
- relinquish ownership of the underlying socket, so it can be reused.
- allow the http server to close our socket if needed (lru_purge_enable)

备注: If async requests are not marked completed, eventually the server will no longer accept incoming connections. The server will log a "httpd_accept_conn: error in accept (23)" message if this happens.

参数 **r** -- **[in]** The request to mark async work as completed

返回

- ESP_OK : async request was marked completed

int **httpd_req_to_sockfd**(*httpd_req_t* *r)

Get the Socket Descriptor from the HTTP request.

This API will return the socket descriptor of the session for which URI handler was executed on reception of HTTP request. This is useful when user wants to call functions that require session socket fd, from within a URI handler, ie. : httpd_sess_get_ctx(), httpd_sess_trigger_close(), httpd_sess_update_lru_counter().

备注: This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.

参数 **r** -- **[in]** The request whose socket descriptor should be found

返回

- Socket descriptor : The socket descriptor for this request
- -1 : Invalid/NULL request pointer

int **httpd_req_recv**(*httpd_req_t* *r, char *buf, size_t buf_len)

API to read content data from the HTTP request.

This API will read HTTP content data from the HTTP request into provided buffer. Use content_len provided in httpd_req_t structure to know the length of data to be fetched. If content_len is too large for the buffer then user may have to make multiple calls to this function, each time fetching 'buf_len' number of bytes, while the pointer to content data is incremented internally by the same number.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- If an error is returned, the URI handler must further return an error. This will ensure that the erroneous socket is closed and cleaned up by the web server.
- Presently Chunked Encoding is not supported

参数

- **r** -- **[in]** The request being responded to
- **buf** -- **[in]** Pointer to a buffer that the data will be read into
- **buf_len** -- **[in]** Length of the buffer

返回

- Bytes : Number of bytes read into the buffer successfully
- 0 : Buffer length parameter is zero / connection closed by peer
- HTTPD_SOCK_ERR_INVALID : Invalid arguments

- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket recv()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket recv()

size_t **httpd_req_get_hdr_value_len**(*httpd_req_t* *r, const char *field)

Search for a field in request headers and return the string length of it's value.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once httpd_resp_send() API is called all request headers are purged, so request headers need be copied into separate buffers if they are required later.

参数

- **r** -- **[in]** The request being responded to
- **field** -- **[in]** The header field to be searched in the request

返回

- Length : If field is found in the request URL
- Zero : Field not found / Invalid request / Null arguments

esp_err_t **httpd_req_get_hdr_value_str**(*httpd_req_t* *r, const char *field, char *val, size_t val_size)

Get the value string of a field from the request headers.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once httpd_resp_send() API is called all request headers are purged, so request headers need be copied into separate buffers if they are required later.
- If output size is greater than input, then the value is truncated, accompanied by truncation error as return value.
- Use httpd_req_get_hdr_value_len() to know the right buffer length

---

参数

- **r** -- **[in]** The request being responded to
- **field** -- **[in]** The field to be searched in the header
- **val** -- **[out]** Pointer to the buffer into which the value will be copied if the field is found
- **val_size** -- **[in]** Size of the user buffer "val"

返回

- ESP_OK : Field found in the request header and value string copied
- ESP_ERR_NOT_FOUND : Key not found
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_INVALID_REQ : Invalid HTTP request pointer
- ESP_ERR_HTTPD_RESULT_TRUNC : Value string truncated

size_t **httpd_req_get_url_query_len**(*httpd_req_t* *r)

Get Query string length from the request URL.

备注: This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid

---

参数 **r** -- **[in]** The request being responded to

返回

- Length : Query is found in the request URL

- Zero : Query not found / Null arguments / Invalid request

esp_err_t **httpd_req_get_url_query_str** (*httpd_req_t* *r, char *buf, size_t buf_len)

Get Query string from the request URL.

备注:

- Presently, the user can fetch the full URL query string, but decoding will have to be performed by the user. Request headers can be read using httpd_req_get_hdr_value_str() to know the 'Content-Type' (eg. Content-Type: application/x-www-form-urlencoded) and then the appropriate decoding algorithm needs to be applied.
- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid
- If output size is greater than input, then the value is truncated, accompanied by truncation error as return value
- Prior to calling this function, one can use httpd_req_get_url_query_len() to know the query string length beforehand and hence allocate the buffer of right size (usually query string length + 1 for null termination) for storing the query string

---

参数

- **r** -- **[in]** The request being responded to
- **buf** -- **[out]** Pointer to the buffer into which the query string will be copied (if found)
- **buf_len** -- **[in]** Length of output buffer

返回

- ESP_OK : Query is found in the request URL and copied to buffer
- ESP_ERR_NOT_FOUND : Query not found
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_INVALID_REQ : Invalid HTTP request pointer
- ESP_ERR_HTTPD_RESULT_TRUNC : Query string truncated

esp_err_t **httpd_query_key_value** (const char *qry, const char *key, char *val, size_t val_size)

Helper function to get a URL query tag from a query string of the type param1=val1&param2=val2.

备注:

- The components of URL query string (keys and values) are not URLdecoded. The user must check for 'Content-Type' field in the request headers and then depending upon the specified encoding (URLencoded or otherwise) apply the appropriate decoding algorithm.
- If actual value size is greater than val_size, then the value is truncated, accompanied by truncation error as return value.

参数

- **qry** -- **[in]** Pointer to query string
- **key** -- **[in]** The key to be searched in the query string
- **val** -- **[out]** Pointer to the buffer into which the value will be copied if the key is found
- **val_size** -- **[in]** Size of the user buffer "val"

返回

- ESP_OK : Key is found in the URL query string and copied to buffer
- ESP_ERR_NOT_FOUND : Key not found
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESULT_TRUNC : Value string truncated

esp_err_t **httpd_req_get_cookie_val** (*httpd_req_t* *req, const char *cookie_name, char *val, size_t *val_size)

Get the value string of a cookie value from the "Cookie" request headers by cookie name.

参数

- **req** -- **[in]** Pointer to the HTTP request
- **cookie_name** -- **[in]** The cookie name to be searched in the request
- **val** -- **[out]** Pointer to the buffer into which the value of cookie will be copied if the cookie is found
- **val_size** -- **[inout]** Pointer to size of the user buffer "val". This variable will contain cookie length if ESP_OK is returned and required buffer length in case ESP_ERR_HTTPD_RESULT_TRUNC is returned.

返回

- ESP_OK : Key is found in the cookie string and copied to buffer
- ESP_ERR_NOT_FOUND : Key not found
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESULT_TRUNC : Value string truncated
- ESP_ERR_NO_MEM : Memory allocation failure

bool **httpd_uri_match_wildcard**(const char *uri_template, const char *uri_to_match, size_t match_upto)

Test if a URI matches the given wildcard template.

Template may end with "?" to make the previous character optional (typically a slash), "*" for a wildcard match, and "?*" to make the previous character optional, and if present, allow anything to follow.

Example:

- * matches everything
- /api/? matches /api and /api/
- /api/* (sans the backslash) matches /api/ and /api/status, but not /api or /ap
- /api/?* or /api/*? (sans the backslash) matches /api/, /api/status, and also /api, but not /apix or /ap

The special characters "?" and "*" anywhere else in the template will be taken literally.

参数

- **uri_template** -- **[in]** URI template (pattern)
- **uri_to_match** -- **[in]** URI to be matched
- **match_upto** -- **[in]** how many characters of the URI buffer to test (there may be trailing query string etc.)

返回 true if a match was found

esp_err_t **httpd_resp_send**(*httpd_req_t* *r, const char *buf, ssize_t buf_len)

API to send a complete HTTP response.

This API will send the data as an HTTP response to the request. This assumes that you have the entire response ready in a single buffer. If you wish to send response in incremental chunks use httpd_resp_send_chunk() instead.

If no status code and content-type were set, by default this will send 200 OK status code and content type as text/html. You may call the following functions before this API to configure the response headers : httpd_resp_set_status() - for setting the HTTP status string, httpd_resp_set_type() - for setting the Content Type, httpd_resp_set_hdr() - for appending any additional field value entries in the response header

---

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, the request has been responded to.
- No additional data can then be sent for the request.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

参数

- **r** -- **[in]** The request being responded to

- **buf** -- **[in]** Buffer from where the content is to be fetched
- **buf_len** -- **[in]** Length of the buffer, HTTPD_RESP_USE_STRLEN to use strlen()

返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null request pointer
- ESP_ERR_HTTPD_RESP_HDR : Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request

esp_err_t **httpd_resp_send_chunk** (*httpd_req_t* *r, const char *buf, ssize_t buf_len)

API to send one HTTP chunk.

This API will send the data as an HTTP response to the request. This API will use chunked-encoding and send the response in the form of chunks. If you have the entire response contained in a single buffer, please use httpd_resp_send() instead.

If no status code and content-type were set, by default this will send 200 OK status code and content type as text/html. You may call the following functions before this API to configure the response headers httpd_resp_set_status() - for setting the HTTP status string, httpd_resp_set_type() - for setting the Content Type, httpd_resp_set_hdr() - for appending any additional field value entries in the response header

---

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- When you are finished sending all your chunks, you must call this function with buf_len as 0.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

参数

- **r** -- **[in]** The request being responded to
- **buf** -- **[in]** Pointer to a buffer that stores the data
- **buf_len** -- **[in]** Length of the buffer, HTTPD_RESP_USE_STRLEN to use strlen()

返回

- ESP_OK : On successfully sending the response packet chunk
- ESP_ERR_INVALID_ARG : Null request pointer
- ESP_ERR_HTTPD_RESP_HDR : Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

static inline *esp_err_t* **httpd_resp_sendstr** (*httpd_req_t* *r, const char *str)

API to send a complete string as HTTP response.

This API simply calls http_resp_send with buffer length set to string length assuming the buffer contains a null terminated string

参数

- **r** -- **[in]** The request being responded to
- **str** -- **[in]** String to be sent as response body

返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null request pointer
- ESP_ERR_HTTPD_RESP_HDR : Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request

static inline *esp_err_t* **httpd_resp_sendstr_chunk** (*httpd_req_t* *r, const char *str)

API to send a string as an HTTP response chunk.

This API simply calls http_resp_send_chunk with buffer length set to string length assuming the buffer contains a null terminated string

参数

- **r** -- **[in]** The request being responded to
- **str** -- **[in]** String to be sent as response body (NULL to finish response packet)

返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null request pointer
- ESP_ERR_HTTPD_RESP_HDR : Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request

esp_err_t **httpd_resp_set_status** (*httpd_req_t* *r, const char *status)

API to set the HTTP status code.

This API sets the status of the HTTP response to the value specified. By default, the '200 OK' response is sent as the response.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- This API only sets the status to this value. The status isn't sent out until any of the send APIs is executed.
- Make sure that the lifetime of the status string is valid till send function is called.

参数

- **r** -- **[in]** The request being responded to
- **status** -- **[in]** The HTTP status code of this response

返回

- ESP_OK : On success
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

esp_err_t **httpd_resp_set_type** (*httpd_req_t* *r, const char *type)

API to set the HTTP content type.

This API sets the 'Content Type' field of the response. The default content type is 'text/html'.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- This API only sets the content type to this value. The type isn't sent out until any of the send APIs is executed.
- Make sure that the lifetime of the type string is valid till send function is called.

参数

- **r** -- **[in]** The request being responded to
- **type** -- **[in]** The Content Type of the response

返回

- ESP_OK : On success
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

esp_err_t **httpd_resp_set_hdr** (*httpd_req_t* *r, const char *field, const char *value)

API to append any additional headers.

This API sets any additional header fields that need to be sent in the response.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- The header isn't sent out until any of the send APIs is executed.
- The maximum allowed number of additional headers is limited to value of max_resp_headers in config structure.
- Make sure that the lifetime of the field value strings are valid till send function is called.

参数

- **`r`** -- **[in]** The request being responded to
- **`field`** -- **[in]** The field name of the HTTP header
- **`value`** -- **[in]** The value of this HTTP header

返回

- ESP_OK : On successfully appending new header
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_HDR : Total additional headers exceed max allowed
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

esp_err_t **`httpd_resp_send_err`** (*httpd_req_t* *req, *httpd_err_code_t* error, const char *msg)

For sending out error code in response to HTTP request.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.
- If you wish to send additional data in the body of the response, please use the lower-level functions directly.

参数

- **`req`** -- **[in]** Pointer to the HTTP request for which the response needs to be sent
- **`error`** -- **[in]** Error type to send
- **`msg`** -- **[in]** Error message string (pass NULL for default message)

返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

esp_err_t **`httpd_resp_send_custom_err`** (*httpd_req_t* *req, const char *status, const char *msg)

For sending out custom error code in response to HTTP request.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.
- If you wish to send additional data in the body of the response, please use the lower-level functions directly.

参数

- **req** -- **[in]** Pointer to the HTTP request for which the response needs to be sent
- **status** -- **[in]** Error status to send
- **msg** -- **[in]** Error message string

返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

static inline *esp_err_t* **httpd_resp_send_404** (*httpd_req_t* *r)

Helper function for HTTP 404.

Send HTTP 404 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

参数 **r** -- **[in]** The request being responded to

返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

static inline *esp_err_t* **httpd_resp_send_408** (*httpd_req_t* *r)

Helper function for HTTP 408.

Send HTTP 408 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

参数 **r** -- **[in]** The request being responded to

返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

static inline *esp_err_t* **httpd_resp_send_500** (*httpd_req_t* *r)

Helper function for HTTP 500.

Send HTTP 500 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

参数 **r** -- **[in]** The request being responded to

返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

int **httpd_send**(*httpd_req_t* *r, const char *buf, size_t buf_len)

Raw HTTP send.

Call this API if you wish to construct your custom response packet. When using this, all essential header, eg. HTTP version, Status Code, Content Type and Length, Encoding, etc. will have to be constructed manually, and HTTP delimiters (CRLF) will need to be placed correctly for separating sub-sections of the HTTP response packet.

If the send override function is set, this API will end up calling that function eventually to send data out.

备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Unless the response has the correct HTTP structure (which the user must now ensure) it is not guaranteed that it will be recognized by the client. For most cases, you wouldn't have to call this API, but you would rather use either of : httpd_resp_send(), httpd_resp_send_chunk()

参数

- **r** -- **[in]** The request being responded to
- **buf** -- **[in]** Buffer from where the fully constructed packet is to be read
- **buf_len** -- **[in]** Length of the buffer

返回

- Bytes : Number of bytes that were sent successfully
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket send()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket send()

int **httpd_socket_send**(*httpd_handle_t* hd, int sockfd, const char *buf, size_t buf_len, int flags)

A low level API to send data on a given socket

This internally calls the default send function, or the function registered by httpd_sess_set_send_override().

备注: This API is not recommended to be used in any request handler. Use this only for advanced use cases, wherein some asynchronous data is to be sent over a socket.

参数

- **hd** -- **[in]** server instance
- **sockfd** -- **[in]** session socket file descriptor
- **buf** -- **[in]** buffer with bytes to send
- **buf_len** -- **[in]** data size
- **flags** -- **[in]** flags for the send() function

返回

- Bytes : The number of bytes sent successfully
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket send()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket send()

int **httpd_socket_recv**(*httpd_handle_t* hd, int sockfd, char *buf, size_t buf_len, int flags)

A low level API to receive data from a given socket

This internally calls the default recv function, or the function registered by httpd_sess_set_recv_override().

备注: This API is not recommended to be used in any request handler. Use this only for advanced use cases, wherein some asynchronous communication is required.

参数

- **hd** -- **[in]** server instance
- **sockfd** -- **[in]** session socket file descriptor
- **buf** -- **[in]** buffer with bytes to send
- **buf_len** -- **[in]** data size
- **flags** -- **[in]** flags for the send() function

返回

- Bytes : The number of bytes received successfully
- 0 : Buffer length parameter is zero / connection closed by peer
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket recv()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket recv()

esp_err_t **httpd_register_err_handler**(*httpd_handle_t* handle, *httpd_err_code_t* error, *httpd_err_handler_func_t* handler_fn)

Function for registering HTTP error handlers.

This function maps a handler function to any supported error code given by `httpd_err_code_t`. See prototype `httpd_err_handler_func_t` above for details.

参数

- **handle** -- **[in]** HTTP server handle
- **error** -- **[in]** Error type
- **handler_fn** -- **[in]** User implemented handler function (Pass NULL to unset any previously set handler)

返回

- ESP_OK : handler registered successfully
- ESP_ERR_INVALID_ARG : invalid error code or server handle

esp_err_t **httpd_queue_work**(*httpd_handle_t* handle, *httpd_work_fn_t* work, void *arg)

Queue execution of a function in HTTPD's context.

This API queues a work function for asynchronous execution

备注: Some protocols require that the web server generate some asynchronous data and send it to the persistently opened connection. This facility is for use by such protocols.

参数

- **handle** -- **[in]** Handle to server returned by httpd_start
- **work** -- **[in]** Pointer to the function to be executed in the HTTPD's context
- **arg** -- **[in]** Pointer to the arguments that should be passed to this function

返回

- ESP_OK : On successfully queueing the work
- ESP_FAIL : Failure in ctrl socket
- ESP_ERR_INVALID_ARG : Null arguments

void ***httpd_sess_get_ctx**(*httpd_handle_t* handle, int sockfd)

Get session context from socket descriptor.

Typically if a session context is created, it is available to URI handlers through the httpd_req_t structure. But, there are cases where the web server's send/receive functions may require the context (for example, for accessing keying information etc). Since the send/receive function only have the socket descriptor at their disposal, this API provides them with a way to retrieve the session context.

参数

- **handle** -- **[in]** Handle to server returned by httpd_start
- **sockfd** -- **[in]** The socket descriptor for which the context should be extracted.

返回

- void* : Pointer to the context associated with this session
- NULL : Empty context / Invalid handle / Invalid socket fd

void **httpd_sess_set_ctx**(*httpd_handle_t* handle, int sockfd, void *ctx, *httpd_free_ctx_fn_t* free_fn)

Set session context by socket descriptor.

参数

- **handle** -- **[in]** Handle to server returned by httpd_start
- **sockfd** -- **[in]** The socket descriptor for which the context should be extracted.
- **ctx** -- **[in]** Context object to assign to the session
- **free_fn** -- **[in]** Function that should be called to free the context

void ***httpd_sess_get_transport_ctx**(*httpd_handle_t* handle, int sockfd)

Get session 'transport' context by socket descriptor.

This context is used by the send/receive functions, for example to manage SSL context.

参见:

httpd_sess_get_ctx()

参数

- **handle** -- **[in]** Handle to server returned by httpd_start
- **sockfd** -- **[in]** The socket descriptor for which the context should be extracted.

返回

- void* : Pointer to the transport context associated with this session
- NULL : Empty context / Invalid handle / Invalid socket fd

void **httpd_sess_set_transport_ctx**(*httpd_handle_t* handle, int sockfd, void *ctx, *httpd_free_ctx_fn_t* free_fn)

Set session 'transport' context by socket descriptor.

参见:

httpd_sess_set_ctx()

参数

- **handle** -- **[in]** Handle to server returned by httpd_start
- **sockfd** -- **[in]** The socket descriptor for which the context should be extracted.
- **ctx** -- **[in]** Transport context object to assign to the session
- **free_fn** -- **[in]** Function that should be called to free the transport context

void *httpd_get_global_user_ctx(*httpd_handle_t* handle)

Get HTTPD global user context (it was set in the server config struct)

参数 **handle** -- **[in]** Handle to server returned by httpd_start
返回 global user context

void *httpd_get_global_transport_ctx(*httpd_handle_t* handle)

Get HTTPD global transport context (it was set in the server config struct)

参数 **handle** -- **[in]** Handle to server returned by httpd_start
返回 global transport context

esp_err_t **httpd_sess_trigger_close**(*httpd_handle_t* handle, int sockfd)

Trigger an httpd session close externally.

备注: Calling this API is only required in special circumstances wherein some application requires to close an httpd client session asynchronously.

参数

- **handle** -- **[in]** Handle to server returned by httpd_start
- **sockfd** -- **[in]** The socket descriptor of the session to be closed

返回

- ESP_OK : On successfully initiating closure
- ESP_FAIL : Failure to queue work
- ESP_ERR_NOT_FOUND : Socket fd not found
- ESP_ERR_INVALID_ARG : Null arguments

esp_err_t **httpd_sess_update_lru_counter**(*httpd_handle_t* handle, int sockfd)

Update LRU counter for a given socket.

LRU Counters are internally associated with each session to monitor how recently a session exchanged traffic. When LRU purge is enabled, if a client is requesting for connection but maximum number of sockets/sessions is reached, then the session having the earliest LRU counter is closed automatically.

Updating the LRU counter manually prevents the socket from being purged due to the Least Recently Used (LRU) logic, even though it might not have received traffic for some time. This is useful when all open sockets/session are frequently exchanging traffic but the user specifically wants one of the sessions to be kept open, irrespective of when it last exchanged a packet.

备注: Calling this API is only necessary if the LRU Purge Enable option is enabled.

参数

- **handle** -- **[in]** Handle to server returned by httpd_start
- **sockfd** -- **[in]** The socket descriptor of the session for which LRU counter is to be updated

返回

- ESP_OK : Socket found and LRU counter updated
- ESP_ERR_NOT_FOUND : Socket not found
- ESP_ERR_INVALID_ARG : Null arguments

esp_err_t **httpd_get_client_list**(*httpd_handle_t* handle, size_t *fds, int *client_fds)

Returns list of current socket descriptors of active sessions.

备注: Size of provided array has to be equal or greater then maximum number of opened sockets, configured upon initialization with max_open_sockets field in httpd_config_t structure.

参数

- **handle** -- **[in]** Handle to server returned by httpd_start
- **fds** -- **[inout]** In: Size of provided client_fds array Out: Number of valid client fds returned in client_fds,
- **client_fds** -- **[out]** Array of client fds

返回

- ESP_OK : Successfully retrieved session list
- ESP_ERR_INVALID_ARG : Wrong arguments or list is longer than provided array

Structures

struct **esp_http_server_event_data**

Argument structure for HTTP_SERVER_EVENT_ON_DATA and HTTP_SERVER_EVENT_SENT_DATA event

Public Members

int **fd**

Session socket file descriptor

int **data_len**

Data length

struct **httpd_config**

HTTP Server Configuration Structure.

备注： Use HTTPD_DEFAULT_CONFIG() to initialize the configuration to a default value and then modify only those fields that are specifically determined by the use case.

Public Members

unsigned **task_priority**

Priority of FreeRTOS task which runs the server

size_t **stack_size**

The maximum stack size allowed for the server task

BaseType_t **core_id**

The core the HTTP server task will run on

uint32_t **task_caps**

The memory capabilities to use when allocating the HTTP server task's stack

uint16_t **server_port**

TCP Port number for receiving and transmitting HTTP traffic

uint16_t **ctrl_port**

UDP Port number for asynchronously exchanging control signals between various components of the server

uint16_t **max_open_sockets**

Max number of sockets/clients connected at any time (3 sockets are reserved for internal working of the HTTP server)

uint16_t **max_uri_handlers**

Maximum allowed uri handlers

uint16_t **max_resp_headers**

Maximum allowed additional headers in HTTP response

uint16_t **backlog_conn**

Number of backlog connections

bool **lru_purge_enable**

Purge "Least Recently Used" connection

uint16_t **recv_wait_timeout**

Timeout for recv function (in seconds)

uint16_t **send_wait_timeout**

Timeout for send function (in seconds)

void ***global_user_ctx**

Global user context.

This field can be used to store arbitrary user data within the server context. The value can be retrieved using the server handle, available e.g. in the httpd_req_t struct.

When shutting down, the server frees up the user context by calling free() on the global_user_ctx field. If you wish to use a custom function for freeing the global user context, please specify that here.

httpd_free_ctx_fn_t **global_user_ctx_free_fn**

Free function for global user context

void ***global_transport_ctx**

Global transport context.

Similar to global_user_ctx, but used for session encoding or encryption (e.g. to hold the SSL context). It will be freed using free(), unless global_transport_ctx_free_fn is specified.

httpd_free_ctx_fn_t **global_transport_ctx_free_fn**

Free function for global transport context

bool **enable_so_linger**

bool to enable/disable linger

int **linger_timeout**

linger timeout (in seconds)

bool **keep_alive_enable**

Enable keep-alive timeout

int **keep_alive_idle**

Keep-alive idle time. Default is 5 (second)

int **keep_alive_interval**

Keep-alive interval time. Default is 5 (second)

int **keep_alive_count**

Keep-alive packet retry send count. Default is 3 counts

httpd_open_func_t **open_fn**

Custom session opening callback.

Called on a new session socket just after accept(), but before reading any data.

This is an opportunity to set up e.g. SSL encryption using global_transport_ctx and the send/recv/pending session overrides.

If a context needs to be maintained between these functions, store it in the session using httpd_sess_set_transport_ctx() and retrieve it later with httpd_sess_get_transport_ctx()

Returning a value other than ESP_OK will immediately close the new socket.

httpd_close_func_t **close_fn**

Custom session closing callback.

Called when a session is deleted, before freeing user and transport contexts and before closing the socket. This is a place for custom de-init code common to all sockets.

The server will only close the socket if no custom session closing callback is set. If a custom callback is used, `close(sockfd)` should be called in here for most cases.

Set the user or transport context to NULL if it was freed here, so the server does not try to free it again.

This function is run for all terminated sessions, including sessions where the socket was closed by the network stack - that is, the file descriptor may not be valid anymore.

httpd_uri_match_func_t **uri_match_fn**

URI matcher function.

Called when searching for a matching URI: 1) whose request handler is to be executed right after an HTTP request is successfully parsed 2) in order to prevent duplication while registering a new URI handler using `httpd_register_uri_handler()`

Available options are: 1) NULL : Internally do basic matching using `strncmp()` 2) `httpd_uri_match_wildcard()` : URI wildcard matcher

Users can implement their own matching functions (See description of the `httpd_uri_match_func_t` function prototype)

struct **httpd_req**

HTTP Request Data Structure.

Public Members

httpd_handle_t **handle**

Handle to server instance

int **method**

The type of HTTP request, -1 if unsupported method, HTTP_ANY for wildcard method to support every method

const char **uri**[HTTPD_MAX_URI_LEN + 1]

The URI of this request (1 byte extra for null termination)

size_t **content_len**

Length of the request body

void ***aux**

Internally used members

void ***user_ctx**

User context pointer passed during URI registration.

void ***sess_ctx**

Session Context Pointer

A session context. Contexts are maintained across 'sessions' for a given open TCP connection. One session could have multiple request responses. The web server will ensure that the context persists across all these request and responses.

By default, this is NULL. URI Handlers can set this to any meaningful value.

If the underlying socket gets closed, and this pointer is non-NULL, the web server will free up the context by calling free(), unless free_ctx function is set.

httpd_free_ctx_fn_t **free_ctx**

Pointer to free context hook

Function to free session context

If the web server's socket closes, it frees up the session context by calling free() on the sess_ctx member. If you wish to use a custom function for freeing the session context, please specify that here.

bool **ignore_sess_ctx_changes**

Flag indicating if Session Context changes should be ignored

By default, if you change the sess_ctx in some URI handler, the http server will internally free the earlier context (if non NULL), after the URI handler returns. If you want to manage the allocation/reallocation/freeing of sess_ctx yourself, set this flag to true, so that the server will not perform any checks on it. The context will be cleared by the server (by calling free_ctx or free()) only if the socket gets closed.

struct **httpd_uri**

Structure for URI handler.

Public Members

const char ***uri**

The URI to handle

httpd_method_t **method**

Method supported by the URI, HTTP_ANY for wildcard method to support all methods

esp_err_t (***handler**)(*httpd_req_t* *r)

Handler to call for supported request method. This must return ESP_OK, or else the underlying socket will be closed.

void ***user_ctx**

Pointer to user context data which will be available to handler

Macros

HTTP_ANY

HTTPD_MAX_REQ_HDR_LEN

HTTPD_MAX_URI_LEN

HTTPD_SOCK_ERR_FAIL

HTTPD_SOCK_ERR_INVALID

HTTPD_SOCK_ERR_TIMEOUT

HTTPD_200

HTTP Response 200

HTTPD_204

HTTP Response 204

HTTPD_207

HTTP Response 207

HTTPD_400

HTTP Response 400

HTTPD_404

HTTP Response 404

HTTPD_408

HTTP Response 408

HTTPD_500

HTTP Response 500

HTTPD_TYPE_JSON

HTTP Content type JSON

HTTPD_TYPE_TEXT

HTTP Content type text/HTML

HTTPD_TYPE_OCTET

HTTP Content type octext-stream

ESP_HTTPD_DEF_CTRL_PORT

HTTP Server control socket port

HTTPD_DEFAULT_CONFIG()

ESP_ERR_HTTPD_BASE

Starting number of HTTPD error codes

ESP_ERR_HTTPD_HANDLERS_FULL

All slots for registering URI handlers have been consumed

ESP_ERR_HTTPD_HANDLER_EXISTS

URI handler with same method and target URI already registered

ESP_ERR_HTTPD_INVALID_REQ

Invalid request pointer

ESP_ERR_HTTPD_RESULT_TRUNC

Result string truncated

ESP_ERR_HTTPD_RESP_HDR

Response header field larger than supported

ESP_ERR_HTTPD_RESP_SEND

Error occurred while sending response packet

ESP_ERR_HTTPD_ALLOC_MEM

Failed to dynamically allocate memory for resource

ESP_ERR_HTTPD_TASK

Failed to launch server task/thread

HTTPD_RESP_USE_STRLEN

Type Definitions

typedef void ***httpd_handle_t**

HTTP Server Instance Handle.

Every instance of the server will have a unique handle.

typedef enum http_method **httpd_method_t**

HTTP Method Type wrapper over ”enum http_method” available in ”http_parser” library.

typedef void (***httpd_free_ctx_fn_t**)(void *ctx)

Prototype for freeing context data (if any)

Param ctx [in] object to free

typedef *esp_err_t* (*__httpd_open_func_t__)(*httpd_handle_t* hd, int sockfd)

Function prototype for opening a session.

Called immediately after the socket was opened to set up the send/recv functions and other parameters of the socket.

Param hd [in] server instance
Param sockfd [in] session socket file descriptor
Return
- ESP_OK : On success
- Any value other than ESP_OK will signal the server to close the socket immediately

typedef void (*__httpd_close_func_t__)(*httpd_handle_t* hd, int sockfd)

Function prototype for closing a session.

备注： It's possible that the socket descriptor is invalid at this point, the function is called for all terminated sessions. Ensure proper handling of return codes.

Param hd [in] server instance
Param sockfd [in] session socket file descriptor

typedef bool (*__httpd_uri_match_func_t__)(const char *reference_uri, const char *uri_to_match, size_t match_upto)

Function prototype for URI matching.

Param reference_uri [in] URI/template with respect to which the other URI is matched
Param uri_to_match [in] URI/template being matched to the reference URI/template
Param match_upto [in] For specifying the actual length of `uri_to_match` up to which the matching algorithm is to be applied (The maximum value is `strlen(uri_to_match)`, independent of the length of `reference_uri`)
Return true on match

typedef struct *httpd_config* **httpd_config_t**

HTTP Server Configuration Structure.

备注： Use HTTPD_DEFAULT_CONFIG() to initialize the configuration to a default value and then modify only those fields that are specifically determined by the use case.

---

typedef struct *httpd_req* **httpd_req_t**

HTTP Request Data Structure.

typedef struct *httpd_uri* **httpd_uri_t**

Structure for URI handler.

typedef int (*__httpd_send_func_t__)(*httpd_handle_t* hd, int sockfd, const char *buf, size_t buf_len, int flags)

Prototype for HTTPDs low-level send function.

备注： User specified send function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will eventually be conveyed as return value of httpd_send() function

Param hd [in] server instance
Param sockfd [in] session socket file descriptor
Param buf [in] buffer with bytes to send
Param buf_len [in] data size
Param flags [in] flags for the send() function
Return

- Bytes : The number of bytes sent successfully
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket send()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket send()

typedef int (***httpd_recv_func_t**)(*httpd_handle_t* hd, int sockfd, char *buf, size_t buf_len, int flags)

Prototype for HTTPDs low-level recv function.

备注： User specified recv function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will eventually be conveyed as return value of httpd_req_recv() function

Param hd [in] server instance
Param sockfd [in] session socket file descriptor
Param buf [in] buffer with bytes to send
Param buf_len [in] data size
Param flags [in] flags for the send() function
Return

- Bytes : The number of bytes received successfully
- 0 : Buffer length parameter is zero / connection closed by peer
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket recv()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket recv()

typedef int (***httpd_pending_func_t**)(*httpd_handle_t* hd, int sockfd)

Prototype for HTTPDs low-level ”get pending bytes” function.

备注： User specified pending function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will be handled accordingly in the server task.

Param hd [in] server instance
Param sockfd [in] session socket file descriptor
Return

- Bytes : The number of bytes waiting to be received
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket pending()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket pending()

typedef *esp_err_t* (***httpd_err_handler_func_t**)(*httpd_req_t* *req, *httpd_err_code_t* error)

Function prototype for HTTP error handling.

This function is executed upon HTTP errors generated during internal processing of an HTTP request. This is used to override the default behavior on error, which is to send HTTP error response and close the underlying socket.

备注:

- If implemented, the server will not automatically send out HTTP error response codes, therefore, httpd_resp_send_err() must be invoked inside this function if user wishes to generate HTTP error responses.
- When invoked, the validity of `uri`, `method`, `content_len` and `user_ctx` fields of the httpd_req_t parameter is not guaranteed as the HTTP request may be partially received/parsed.
- The function must return ESP_OK if underlying socket needs to be kept open. Any other value will ensure that the socket is closed. The return value is ignored when error is of type `HTTPD_500_INTERNAL_SERVER_ERROR` and the socket closed anyway.

Param req **[in]** HTTP request for which the error needs to be handled
Param error **[in]** Error type
Return

- ESP_OK : error handled successful
- ESP_FAIL : failure indicates that the underlying socket needs to be closed

typedef void (***httpd_work_fn_t**)(void *arg)

Prototype of the HTTPD work function Please refer to httpd_queue_work() for more details.

Param arg **[in]** The arguments for this work function

Enumerations

enum **httpd_err_code_t**

Error codes sent as HTTP response in case of errors encountered during processing of an HTTP request.

Values:

enumerator **HTTPD_500_INTERNAL_SERVER_ERROR**

enumerator **HTTPD_501_METHOD_NOT_IMPLEMENTED**

enumerator **HTTPD_505_VERSION_NOT_SUPPORTED**

enumerator **HTTPD_400_BAD_REQUEST**

enumerator **HTTPD_401_UNAUTHORIZED**

enumerator **HTTPD_403_FORBIDDEN**

enumerator **HTTPD_404_NOT_FOUND**

enumerator **HTTPD_405_METHOD_NOT_ALLOWED**

enumerator **HTTPD_408_REQ_TIMEOUT**

enumerator **HTTPD_411_LENGTH_REQUIRED**

enumerator **HTTPD_414_URI_TOO_LONG**

enumerator **HTTPD_431_REQ_HDR_FIELDS_TOO_LARGE**

enumerator **HTTPD_ERR_CODE_MAX**

enum **esp_http_server_event_id_t**

HTTP Server events id.

Values:

enumerator **HTTP_SERVER_EVENT_ERROR**

This event occurs when there are any errors during execution

enumerator **HTTP_SERVER_EVENT_START**

This event occurs when HTTP Server is started

enumerator **HTTP_SERVER_EVENT_ON_CONNECTED**

Once the HTTP Server has been connected to the client, no data exchange has been performed

enumerator **HTTP_SERVER_EVENT_ON_HEADER**

Occurs when receiving each header sent from the client

enumerator **HTTP_SERVER_EVENT_HEADERS_SENT**

After sending all the headers to the client

enumerator **HTTP_SERVER_EVENT_ON_DATA**

Occurs when receiving data from the client

enumerator **HTTP_SERVER_EVENT_SENT_DATA**

Occurs when an ESP HTTP server session is finished

enumerator **HTTP_SERVER_EVENT_DISCONNECTED**

The connection has been disconnected

enumerator **HTTP_SERVER_EVENT_STOP**

This event occurs when HTTP Server is stopped

2.2.10 HTTPS 服务器

概述

HTTPS 服务器组件建立在*HTTP 服务器* 组件的基础上。该服务器借助常规 HTTP 服务器中的钩子注册函数，注册 SSL 会话回调处理函数。

HTTP 服务器 组件的所有文档同样适用于用户按照本文档搭建的服务器。

API 说明

下列*HTTP 服务器* 的 API 已不适用于*HTTPS 服务器*。这些 API 仅限内部使用，用于处理安全会话和维护内部状态。

- ”send”、”receive” 和”pending” 回调注册函数——处理安全套接字
  - *httpd_sess_set_send_override()*
  - *httpd_sess_set_recv_override()*
  - *httpd_sess_set_pending_override()*
- ”transport context”——传输层上下文
  - *httpd_sess_get_transport_ctx()*：返回会话使用的 SSL
  - *httpd_sess_set_transport_ctx()*
  - *httpd_get_global_transport_ctx()*：返回共享的 SSL 上下文
  - *httpd_config::global_transport_ctx*
  - *httpd_config::global_transport_ctx_free_fn*
  - *httpd_config::open_fn*：用于设置安全套接字

其他 API 均可使用，没有其他限制。

如何使用

请参考示例 protocols/https_server 来学习如何搭建安全的服务器。

总体而言，只需生成证书，将其嵌入到固件中，并且在初始化结构体中配置好正确的证书地址和长度后，将其传入服务器启动函数。

通过改变初始化配置结构体中的标志*httpd_ssl_config::transport_mode*，可以选择是否需要 SSL 连接来启动服务器。在测试时或在速度比安全性更重要的可信环境中，可以使用此功能。

性能

建立起始会话大约需要两秒，在时钟速度较慢或日志记录冗余信息较多的情况下，可能需要花费更多时间。后续通过已打开的安全套接字建立请求的速度会更快，最快只需不到 100 ms。

事件处理

ESP HTTPS 服务器在特定事件发生时，可以通过*事件循环库* 触发事件处理程序。处理程序必须使用*esp_event_handler_register()* 进行注册，以帮助 ESP HTTPS 服务器处理事件。

esp_https_server_event_id_t 包含了 ESP HTTPS 服务器可能发生的所有事件。

事件循环中不同 ESP HTTPS 服务器事件的预期数据类型如下所示：

- HTTPS_SERVER_EVENT_ERROR : `esp_https_server_last_error_t`
- HTTPS_SERVER_EVENT_START : `NULL`
- HTTPS_SERVER_EVENT_ON_CONNECTED : `NULL`
- HTTPS_SERVER_EVENT_ON_DATA : `int`
- HTTPS_SERVER_EVENT_SENT_DATA : `NULL`
- HTTPS_SERVER_EVENT_DISCONNECTED : `NULL`
- HTTPS_SERVER_EVENT_STOP : `NULL`

API 参考

Header File

- components/esp_https_server/include/esp_https_server.h
- This header file can be included with:

```
#include "esp_https_server.h"
```

- This header file is a part of the API provided by the `esp_https_server` component. To declare that your component depends on `esp_https_server`, add the following to your CMakeLists.txt:

```
REQUIRES esp_https_server
```

or

```
PRIV_REQUIRES esp_https_server
```

Functions

esp_err_t **httpd_ssl_start** (*httpd_handle_t* *handle, *httpd_ssl_config_t* *config)

Create a SSL capable HTTP server (secure mode may be disabled in config)

参数

- **config** -- **[inout]** - server config, must not be const. Does not have to stay valid after calling this function.
- **handle** -- **[out]** - storage for the server handle, must be a valid pointer

返回 success

esp_err_t **httpd_ssl_stop** (*httpd_handle_t* handle)

Stop the server. Blocks until the server is shut down.

参数 **handle** -- **[in]**

返回

- ESP_OK: Server stopped successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_FAIL: Failure to shut down server

Structures

struct **esp_https_server_user_cb_arg**

Callback data struct, contains the ESP-TLS connection handle and the connection state at which the callback is executed.

Public Members

httpd_ssl_user_cb_state_t **user_cb_state**

State of user callback

esp_tls_t ***tls**

ESP-TLS connection handle

struct **httpd_ssl_config**

HTTPS server config struct

Please use HTTPD_SSL_CONFIG_DEFAULT() to initialize it.

Public Members

httpd_config_t **httpd**

Underlying HTTPD server config

Parameters like task stack size and priority can be adjusted here.

const uint8_t ***servercert**

Server certificate

size_t **servercert_len**

Server certificate byte length

const uint8_t ***cacert_pem**

CA certificate ((CA used to sign clients, or client cert itself)

size_t **cacert_len**

CA certificate byte length

const uint8_t ***prvtkey_pem**

Private key

size_t **prvtkey_len**

Private key byte length

bool **use_ecdsa_peripheral**

Use ECDSA peripheral to use private key

uint8_t **ecdsa_key_efuse_blk**

The efuse block where ECDSA key is stored

httpd_ssl_transport_mode_t **transport_mode**

Transport Mode (default secure)

uint16_t **port_secure**

Port used when transport mode is secure (default 443)

uint16_t **port_insecure**

Port used when transport mode is insecure (default 80)

bool **session_tickets**

Enable tls session tickets

bool **use_secure_element**

Enable secure element for server session

esp_https_server_user_cb ***user_cb**

User callback for esp_https_server

void ***ssl_userdata**

User data to add to the ssl context

esp_tls_handshake_callback **cert_select_cb**

Certificate selection callback to use. The callback is only applicable when CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK is enabled in menuconfig

const char ***alpn_protos**

Application protocols the server supports in order of prefernece. Used for negotiating during the TLS handshake, first one the client supports is selected. The data structure must live as long as the https server itself

Macros

HTTPD_SSL_CONFIG_DEFAULT()

Default config struct init Notes:

- port is set when starting the server, according to 'transport_mode'
- one socket uses ~ 40kB RAM with SSL, we reduce the default socket count to 4
- SSL sockets are usually long-lived, closing LRU prevents pool exhaustion DOS
- Stack size may need adjustments depending on the user application

Type Definitions

typedef struct *esp_https_server_user_cb_arg* **esp_https_server_user_cb_arg_t**

Callback data struct, contains the ESP-TLS connection handle and the connection state at which the callback is executed.

typedef *esp_tls_last_error_t* **esp_https_server_last_error_t**

typedef void **esp_https_server_user_cb**(*esp_https_server_user_cb_arg_t* *user_cb)

Callback function prototype Can be used to get connection or client information (SSL context) E.g. Client certificate, Socket FD, Connection state, etc.

Param user_cb Callback data struct

typedef struct *httpd_ssl_config* **httpd_ssl_config_t**

Enumerations

enum **esp_https_server_event_id_t**

Values:

enumerator **HTTPS_SERVER_EVENT_ERROR**

This event occurs when there are any errors during execution

enumerator **HTTPS_SERVER_EVENT_START**

This event occurs when HTTPS Server is started

enumerator **HTTPS_SERVER_EVENT_ON_CONNECTED**

Once the HTTPS Server has been connected to the client

enumerator **HTTPS_SERVER_EVENT_ON_DATA**

Occurs when receiving data from the client

enumerator **HTTPS_SERVER_EVENT_SENT_DATA**

Occurs when an ESP HTTPS server sends data to the client

enumerator **HTTPS_SERVER_EVENT_DISCONNECTED**

The connection has been disconnected

enumerator **HTTPS_SERVER_EVENT_STOP**

This event occurs when HTTPS Server is stopped

enum **httpd_ssl_transport_mode_t**

Values:

enumerator **HTTPD_SSL_TRANSPORT_SECURE**

enumerator **HTTPD_SSL_TRANSPORT_INSECURE**

enum **httpd_ssl_user_cb_state_t**

Indicates the state at which the user callback is executed, i.e at session creation or session close.

Values:

enumerator **HTTPD_SSL_USER_CB_SESS_CREATE**

enumerator **HTTPD_SSL_USER_CB_SESS_CLOSE**

2.2.11 ICMP 回显

概述

网际控制报文协议 (ICMP) 通常用于诊断或控制目的，或响应 IP 操作中的错误。常用的网络工具 `ping` 的应答，即 `Echo Reply`，就是基于类型字段值为 0 的 ICMP 数据包实现的。

在 ping 会话中，首先由源主机发出请求包 (ICMP echo request)，然后等待应答包 (ICMP echo reply)，等待具有超时限制。通过这一过程，还能测量出信息的往返用时。收到有效的应答包后，源主机会生成 IP 链路层的统计数据（如失包率、运行时间等）。

IoT 设备通常需要检查远程服务器是否可用。如果服务器离线，设备应向用户发出警告。通过创建 ping 会话，定期发送或解析 ICMP echo 数据包，就能实现这一功能。

为简化这一过程方便用户操作，ESP-IDF 提供了一些好用的 API。

创建 ping 会话 要创建 ping 会话，首先需填写 `esp_ping_config_t`，指定目标芯片 IP 地址、间隔时间等配置。此外，还可以通过 `esp_ping_callbacks_t` 注册回调函数。

创建 ping 会话并注册回调函数示例：

```
static void test_on_ping_success(esp_ping_handle_t hdl, void *args)
{
    // optionally, get callback arguments
    // const char* str = (const char*) args;
    // printf("%s\r\n", str); // "foo"
    uint8_t ttl;
    uint16_t seqno;
    uint32_t elapsed_time, recv_len;
    ip_addr_t target_addr;
    esp_ping_get_profile(hdl, ESP_PING_PROF_SEQNO, &seqno, sizeof(seqno));
    esp_ping_get_profile(hdl, ESP_PING_PROF_TTL, &ttl, sizeof(ttl));
    esp_ping_get_profile(hdl, ESP_PING_PROF_IPADDR, &target_addr, sizeof(target_
↪addr));
```

(下页继续)

(续上页)

```
    esp_ping_get_profile(hdl, ESP_PING_PROF_SIZE, &recv_len, sizeof(recv_len));
    esp_ping_get_profile(hdl, ESP_PING_PROF_TIMEGAP, &elapsed_time, sizeof(elapsed_
↪time));
    printf("%d bytes from %s icmp_seq=%d ttl=%d time=%d ms\n",
           recv_len, inet_ntoa(target_addr.u_addr.ip4), seqno, ttl, elapsed_time);
}

static void test_on_ping_timeout(esp_ping_handle_t hdl, void *args)
{
    uint16_t seqno;
    ip_addr_t target_addr;
    esp_ping_get_profile(hdl, ESP_PING_PROF_SEQNO, &seqno, sizeof(seqno));
    esp_ping_get_profile(hdl, ESP_PING_PROF_IPADDR, &target_addr, sizeof(target_
↪addr));
    printf("From %s icmp_seq=%d timeout\n", inet_ntoa(target_addr.u_addr.ip4),␣
↪seqno);
}

static void test_on_ping_end(esp_ping_handle_t hdl, void *args)
{
    uint32_t transmitted;
    uint32_t received;
    uint32_t total_time_ms;

    esp_ping_get_profile(hdl, ESP_PING_PROF_REQUEST, &transmitted,␣
↪sizeof(transmitted));
    esp_ping_get_profile(hdl, ESP_PING_PROF_REPLY, &received, sizeof(received));
    esp_ping_get_profile(hdl, ESP_PING_PROF_DURATION, &total_time_ms, sizeof(total_
↪time_ms));
    printf("%d packets transmitted, %d received, time %dms\n", transmitted,␣
↪received, total_time_ms);
}

void initialize_ping()
{
    /* convert URL to IP address */
    ip_addr_t target_addr;
    struct addrinfo hint;
    struct addrinfo *res = NULL;
    memset(&hint, 0, sizeof(hint));
    memset(&target_addr, 0, sizeof(target_addr));
    getaddrinfo("www.espressif.com", NULL, &hint, &res);
    struct in_addr addr4 = ((struct sockaddr_in *) (res->ai_addr))->sin_addr;
    inet_addr_to_ip4addr(ip_2_ip4(&target_addr), &addr4);
    freeaddrinfo(res);

    esp_ping_config_t ping_config = ESP_PING_DEFAULT_CONFIG();
    ping_config.target_addr = target_addr;          // target IP address
    ping_config.count = ESP_PING_COUNT_INFINITE;    // ping in infinite mode, esp_
↪ping_stop can stop it

    /* set callback functions */
    esp_ping_callbacks_t cbs;
    cbs.on_ping_success = test_on_ping_success;
    cbs.on_ping_timeout = test_on_ping_timeout;
    cbs.on_ping_end = test_on_ping_end;
    cbs.cb_args = "foo";  // arguments that will feed to all callback functions,␣
↪can be NULL
    cbs.cb_args = eth_event_group;

    esp_ping_handle_t ping;
```

(下页继续)

(续上页)

```
    esp_ping_new_session(&ping_config, &cbs, &ping);
}
```

启动和停止 ping 会话 使用 `esp_ping_new_session` 返回的句柄可以启动或停止 ping 会话。注意，ping 会话在创建后不会自动启动。如果 ping 会话停止后重启，ICMP 数据包的序列号会归零重新计数。

删除 ping 会话 如果不再使用 ping 会话，可用 `esp_ping_delete_session` 将其删除。在删除 ping 会话时，确保该会话已处于停止状态（即已调用了 `esp_ping_stop` ，或该会话已完成所有步骤）。

获取运行时间数据 在回调函数中调用 `esp_ping_get_profile`，可获取 ping 会话的不同运行时间数据，如上文代码示例所示。

应用示例

ICMP echo 示例：protocols/icmp_echo

API 参考

Header File

- components/lwip/include/apps/ping/ping_sock.h
- This header file can be included with:

```
#include "ping/ping_sock.h"
```

- This header file is a part of the API provided by the `lwip` component. To declare that your component depends on `lwip`, add the following to your CMakeLists.txt:

```
REQUIRES lwip
```

or

```
PRIV_REQUIRES lwip
```

Functions

esp_err_t **esp_ping_new_session**(const *esp_ping_config_t* *config, const *esp_ping_callbacks_t* *cbs, *esp_ping_handle_t* *hdl_out)

Create a ping session.

参数

- **config** -- ping configuration
- **cbs** -- a bunch of callback functions invoked by internal ping task
- **hdl_out** -- handle of ping session

返回

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. configuration is null, etc)
- ESP_ERR_NO_MEM: out of memory
- ESP_FAIL: other internal error (e.g. socket error)
- ESP_OK: create ping session successfully, user can take the ping handle to do follow-on jobs

esp_err_t **esp_ping_delete_session**(*esp_ping_handle_t* hdl)

Delete a ping session.

参数 **hdl** -- handle of ping session

返回

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: delete ping session successfully

esp_err_t **esp_ping_start**(*esp_ping_handle_t* hdl)

Start the ping session.

参数 **hdl** -- handle of ping session

返回

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: start ping session successfully

esp_err_t **esp_ping_stop**(*esp_ping_handle_t* hdl)

Stop the ping session.

参数 **hdl** -- handle of ping session

返回

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: stop ping session successfully

esp_err_t **esp_ping_get_profile**(*esp_ping_handle_t* hdl, *esp_ping_profile_t* profile, void *data, uint32_t size)

Get runtime profile of ping session.

参数

- **hdl** -- handle of ping session
- **profile** -- type of profile
- **data** -- profile data
- **size** -- profile data size

返回

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_ERR_INVALID_SIZE: the actual profile data size doesn't match the "size" parameter
- ESP_OK: get profile successfully

Structures

struct **esp_ping_callbacks_t**

Type of "ping" callback functions.

Public Members

void ***cb_args**

arguments for callback functions

void (***on_ping_success**)(*esp_ping_handle_t* hdl, void *args)

Invoked by internal ping thread when received ICMP echo reply packet.

void (***on_ping_timeout**)(*esp_ping_handle_t* hdl, void *args)

Invoked by internal ping thread when receive ICMP echo reply packet timeout.

void (***on_ping_end**)(*esp_ping_handle_t* hdl, void *args)

Invoked by internal ping thread when a ping session is finished.

struct **esp_ping_config_t**

Type of "ping" configuration.

Public Members

uint32_t **count**

A ”ping” session contains count procedures

uint32_t **interval_ms**

Milliseconds between each ping procedure

uint32_t **timeout_ms**

Timeout value (in milliseconds) of each ping procedure

uint32_t **data_size**

Size of the data next to ICMP packet header

int **tos**

Type of Service, a field specified in the IP header

int **ttl**

Time to Live,a field specified in the IP header

ip_addr_t **target_addr**

Target IP address, either IPv4 or IPv6

uint32_t **task_stack_size**

Stack size of internal ping task

uint32_t **task_prio**

Priority of internal ping task

uint32_t **interface**

Netif index, interface=0 means NETIF_NO_INDEX

Macros

ESP_PING_DEFAULT_CONFIG()

Default ping configuration.

ESP_PING_COUNT_INFINITE

Set ping count to zero will ping target infinitely

Type Definitions

typedef void ***esp_ping_handle_t**

Type of ”ping” session handle.

Enumerations

enum **esp_ping_profile_t**

Profile of ping session.

Values:

enumerator **ESP_PING_PROF_SEQNO**
Sequence number of a ping procedure

enumerator **ESP_PING_PROF_TOS**
Type of service of a ping procedure

enumerator **ESP_PING_PROF_TTL**
Time to live of a ping procedure

enumerator **ESP_PING_PROF_REQUEST**
Number of request packets sent out

enumerator **ESP_PING_PROF_REPLY**
Number of reply packets received

enumerator **ESP_PING_PROF_IPADDR**
IP address of replied target

enumerator **ESP_PING_PROF_SIZE**
Size of received packet

enumerator **ESP_PING_PROF_TIMEGAP**
Elapsed time between request and reply packet

enumerator **ESP_PING_PROF_DURATION**
Elapsed time of the whole ping session

2.2.12 mDNS 服务

mDNS 是一种组播 UDP 服务，用来提供本地网络服务和主机发现。

自 v5.0 版本起，ESP-IDF 组件 `mDNS` 已从 ESP-IDF 中迁出至独立的仓库：

- GitHub 上 mDNS 组件

运行 `idf.py add-dependency espressif/mdns`，在项目中添加 mDNS 组件。

托管的文档

请点击如下链接，查看 mDNS 的相关文档：

- mDNS 文档

2.2.13 Mbed TLS

Mbed TLS 是一个 C 代码库，用于实现加密基元、X.509 证书操作以及 SSL/TLS 和 DTLS 协议。该库代码占用空间小，适合嵌入式系统使用。

备注： ESP-IDF 使用的 Mbed TLS 复刻仓库 中包含对原生 Mbed TLS 的补丁。这些补丁与某些模块的硬件例程有关，如 `bignum (MPI)` 和 `ECC`。

Mbed TLS 通过提供以下内容支持 TLS 1.2、TLS 1.3 和 DTLS 1.2 通信：

- TCP/IP 通信功能：监听、连接、接收、读/写。
- SSL/TLS 通信功能：初始化、握手、读/写。
- X.509 功能：CRT、CRL 和密钥处理
- 随机数生成
- 哈希
- 加密/解密

备注： Mbed TLS v3.x.x 系列仅支持 TLS 1.2 和 TLS 1.3 协议。对 SSL 3.0、TLS 1.0/1.1 和 DTLS 1.0 的支持已被删除（弃用）。从 Mbed TLS v3.6.0 版本开始完全支持 TLS 1.3，在此版本之前，某些功能仍处于试验状态。有关更多详细信息，请参阅 Mbed TLS ChangeLog 。

Mbed TLS 文档

Mbed TLS 文档请参阅以下（上游）指针：

- API Reference
- Knowledge Base

ESP-IDF 的 Mbed TLS 支持

请在 此处 查找 ESP-IDF 不同分支上的 Mbed TLS 版本信息。

备注： 参考*Mbed TLS* 从 Mbed TLS 2.x 版本迁移到 3.0 及以上版本。

应用示例

ESP-IDF 中的示例使用*ESP-TLS*，为访问常用的 TLS 功能提供了一个简化 API 接口。

参考示例 protocols/https_server/simple（简单的 HTTPS 服务器）和 protocols/https_request（发出 HTTPS 请求）了解更多信息。

如需直接使用 Mbed TLS API，请参考示例 protocols/https_mbedtls。

其他选项

ESP-TLS 是底层 SSL/TLS 库的抽象层，因此可以选择使用 Mbed TLS 或 wolfSSL 作为底层库。默认情况下，仅 Mbed TLS 可在 ESP-IDF 中使用，而 wolfSSL 在 https://github.com/espressif/esp-wolfSSL 公开，还提供了上游子模块指针的相关信息。

如需了解更多相关信息或比较 Mbed TLS 和 wolfSSL，请参考文档*ESP-TLS: Underlying SSL/TLS Library Options*。

重要配置

`Component Config -> mbedTLS` 中的部分重要配置选项如下表所示。点击此处 获取完整配置选项列表。

- *CONFIG_MBEDTLS_SSL_PROTO_TLS1_2*: 支持 TLS 1.2
- *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3*: 支持 TLS 1.3
- *CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*: 支持受信任的根证书包（更多信息请参考*ESP x509 证书包*）
- *CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS*: 支持 TLS 会话恢复：客户端会话票证
- *CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS*: 支持 TLS 会话恢复：服务会话票证
- *CONFIG_MBEDTLS_HARDWARE_SHA*: 支持硬件 SHA 加速
- *CONFIG_MBEDTLS_HARDWARE_AES*: 支持硬件 AES 加速
- *CONFIG_MBEDTLS_HARDWARE_MPI*: 支持硬件 MPI (bignum) 加速
- *CONFIG_MBEDTLS_HARDWARE_ECC*: 支持硬件 ECC 加速

性能和内存调整

减少内存使用 下表展示了在不同配置下，用 Mbed TLS 作为 SSL/TLS 库运行示例 protocols/https_request（启用服务器验证）时，内存的实际使用情况。

Mbed TLS 测试	相关配置	堆使用（近似）
默认	NA	42196 B
启用 SSL 动态 buffer 长度	*CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH*	42120 B
禁用保留对端证书	*CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE*	38533 B
启用动态 buffer 功能	*CONFIG_MBEDTLS_DYNAMIC_BUFFER* *CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA* *CONFIG_MBEDTLS_DYNAMIC_FREE_CA_CERT*	22013 B

备注： 这些值会随着配置选项和 Mbed TLS 版本的变化而变化。

减小固件大小 在 `Component Config -> mbedTLS` 中，有多个 Mbed TLS 功能默认为启用状态。如果不需要这些功能，可将其禁用以减小固件大小。要了解更多信息，请参考*Minimizing Binary Size* 文档。

此 API 部分的示例代码存放在 ESP-IDF 示例项目的 protocols 目录下。

2.2.14 IP 网络层协议

IP 网络层协议（应用层协议之下）的文档存放在*连网 API* 目录下。

2.3 错误代码参考

本节列出了 ESP-IDF 中定义的各种错误代码常量。

有关 ESP-IDF 中出错处理的通用信息，请参见*错误处理*。

ESP_FAIL (-1): Generic esp_err_t code indicating failure

ESP_OK (0): esp_err_t value indicating success (no error)

ESP_ERR_NO_MEM **(0x101)**: Out of memory

ESP_ERR_INVALID_ARG **(0x102)**: Invalid argument

ESP_ERR_INVALID_STATE **(0x103)**: Invalid state

ESP_ERR_INVALID_SIZE **(0x104)**: Invalid size

ESP_ERR_NOT_FOUND **(0x105)**: Requested resource not found

ESP_ERR_NOT_SUPPORTED **(0x106)**: Operation or feature not supported

ESP_ERR_TIMEOUT **(0x107)**: Operation timed out

ESP_ERR_INVALID_RESPONSE **(0x108)**: Received response was invalid

ESP_ERR_INVALID_CRC **(0x109)**: CRC or checksum was invalid

ESP_ERR_INVALID_VERSION **(0x10a)**: Version was invalid

ESP_ERR_INVALID_MAC **(0x10b)**: MAC address was invalid

ESP_ERR_NOT_FINISHED **(0x10c)**: Operation has not fully completed

ESP_ERR_NOT_ALLOWED **(0x10d)**: Operation is not allowed

ESP_ERR_NVS_BASE **(0x1100)**: Starting number of error codes

ESP_ERR_NVS_NOT_INITIALIZED **(0x1101)**: The storage driver is not initialized

ESP_ERR_NVS_NOT_FOUND **(0x1102)**: A requested entry couldn't be found or namespace doesn't exist yet and mode is NVS_READONLY

ESP_ERR_NVS_TYPE_MISMATCH **(0x1103)**: The type of set or get operation doesn't match the type of value stored in NVS

ESP_ERR_NVS_READ_ONLY **(0x1104)**: Storage handle was opened as read only

ESP_ERR_NVS_NOT_ENOUGH_SPACE **(0x1105)**: There is not enough space in the underlying storage to save the value

ESP_ERR_NVS_INVALID_NAME **(0x1106)**: Namespace name doesn't satisfy constraints

ESP_ERR_NVS_INVALID_HANDLE **(0x1107)**: Handle has been closed or is NULL

ESP_ERR_NVS_REMOVE_FAILED **(0x1108)**: The value wasn't updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn't fail again.

ESP_ERR_NVS_KEY_TOO_LONG **(0x1109)**: Key name is too long

ESP_ERR_NVS_PAGE_FULL **(0x110a)**: Internal error; never returned by nvs API functions

ESP_ERR_NVS_INVALID_STATE **(0x110b)**: NVS is in an inconsistent state due to a previous error. Call nvs_flash_init and nvs_open again, then retry.

ESP_ERR_NVS_INVALID_LENGTH **(0x110c)**: String or blob length is not sufficient to store data

ESP_ERR_NVS_NO_FREE_PAGES **(0x110d)**: NVS partition doesn't contain any empty pages. This may happen if NVS partition was truncated. Erase the whole partition and call nvs_flash_init again.

ESP_ERR_NVS_VALUE_TOO_LONG **(0x110e)**: Value doesn't fit into the entry or string or blob length is longer than supported by the implementation

ESP_ERR_NVS_PART_NOT_FOUND **(0x110f)**: Partition with specified name is not found in the partition table

ESP_ERR_NVS_NEW_VERSION_FOUND **(0x1110)**: NVS partition contains data in new format and cannot be recognized by this version of code

ESP_ERR_NVS_XTS_ENCR_FAILED **(0x1111)**: XTS encryption failed while writing NVS entry

ESP_ERR_NVS_XTS_DECR_FAILED **(0x1112)**: XTS decryption failed while reading NVS entry

ESP_ERR_NVS_XTS_CFG_FAILED **(0x1113)**: XTS configuration setting failed

ESP_ERR_NVS_XTS_CFG_NOT_FOUND **(0x1114)**: XTS configuration not found

ESP_ERR_NVS_ENCR_NOT_SUPPORTED **(0x1115)**: NVS encryption is not supported in this version

ESP_ERR_NVS_KEYS_NOT_INITIALIZED **(0x1116)**: NVS key partition is uninitialized

ESP_ERR_NVS_CORRUPT_KEY_PART **(0x1117)**: NVS key partition is corrupt

ESP_ERR_NVS_CONTENT_DIFFERS **(0x1118)**: Internal error; never returned by nvs API functions. NVS key is different in comparison

ESP_ERR_NVS_WRONG_ENCRYPTION **(0x1119)**: NVS partition is marked as encrypted with generic flash encryption. This is forbidden since the NVS encryption works differently.

ESP_ERR_ULP_BASE **(0x1200)**: Offset for ULP-related error codes

ESP_ERR_ULP_SIZE_TOO_BIG **(0x1201)**: Program doesn't fit into RTC memory reserved for the ULP

ESP_ERR_ULP_INVALID_LOAD_ADDR **(0x1202)**: Load address is outside of RTC memory reserved for the ULP

ESP_ERR_ULP_DUPLICATE_LABEL **(0x1203)**: More than one label with the same number was defined

ESP_ERR_ULP_UNDEFINED_LABEL **(0x1204)**: Branch instructions references an undefined label

ESP_ERR_ULP_BRANCH_OUT_OF_RANGE **(0x1205)**: Branch target is out of range of B instruction (try replacing with BX)

ESP_ERR_OTA_BASE **(0x1500)**: Base error code for ota_ops api

ESP_ERR_OTA_PARTITION_CONFLICT **(0x1501)**: Error if request was to write or erase the current running partition

ESP_ERR_OTA_SELECT_INFO_INVALID **(0x1502)**: Error if OTA data partition contains invalid content

ESP_ERR_OTA_VALIDATE_FAILED **(0x1503)**: Error if OTA app image is invalid

ESP_ERR_OTA_SMALL_SEC_VER **(0x1504)**: Error if the firmware has a secure version less than the running firmware.

ESP_ERR_OTA_ROLLBACK_FAILED **(0x1505)**: Error if flash does not have valid firmware in passive partition and hence rollback is not possible

ESP_ERR_OTA_ROLLBACK_INVALID_STATE **(0x1506)**: Error if current active firmware is still marked in pending validation state (ESP_OTA_IMG_PENDING_VERIFY), essentially first boot of firmware image post upgrade and hence firmware upgrade is not possible

ESP_ERR_EFUSE **(0x1600)**: Base error code for efuse api.

ESP_OK_EFUSE_CNT **(0x1601)**: OK the required number of bits is set.

ESP_ERR_EFUSE_CNT_IS_FULL **(0x1602)**: Error field is full.

ESP_ERR_EFUSE_REPEATED_PROG **(0x1603)**: Error repeated programming of programmed bits is strictly forbidden.

ESP_ERR_CODING **(0x1604)**: Error while a encoding operation.

ESP_ERR_NOT_ENOUGH_UNUSED_KEY_BLOCKS **(0x1605)**: Error not enough unused key blocks available

ESP_ERR_DAMAGED_READING **(0x1606)**: Error. Burn or reset was done during a reading operation leads to damage read data. This error is internal to the efuse component and not returned by any public API.

ESP_ERR_IMAGE_BASE **(0x2000)**

ESP_ERR_IMAGE_FLASH_FAIL **(0x2001)**

ESP_ERR_IMAGE_INVALID **(0x2002)**

ESP_ERR_WIFI_BASE **(0x3000)**: Starting number of WiFi error codes

ESP_ERR_WIFI_NOT_INIT **(0x3001)**: WiFi driver was not installed by esp_wifi_init

ESP_ERR_WIFI_NOT_STARTED **(0x3002)**: WiFi driver was not started by esp_wifi_start

ESP_ERR_WIFI_NOT_STOPPED **(0x3003)**: WiFi driver was not stopped by esp_wifi_stop

ESP_ERR_WIFI_IF **(0x3004)**: WiFi interface error

ESP_ERR_WIFI_MODE **(0x3005)**: WiFi mode error

ESP_ERR_WIFI_STATE **(0x3006)**: WiFi internal state error

ESP_ERR_WIFI_CONN **(0x3007)**: WiFi internal control block of station or soft-AP error

ESP_ERR_WIFI_NVS **(0x3008)**: WiFi internal NVS module error

ESP_ERR_WIFI_MAC **(0x3009)**: MAC address is invalid

ESP_ERR_WIFI_SSID **(0x300a)**: SSID is invalid

ESP_ERR_WIFI_PASSWORD **(0x300b)**: Password is invalid

ESP_ERR_WIFI_TIMEOUT **(0x300c)**: Timeout error

ESP_ERR_WIFI_WAKE_FAIL **(0x300d)**: WiFi is in sleep state(RF closed) and wakeup fail

ESP_ERR_WIFI_WOULD_BLOCK **(0x300e)**: The caller would block

ESP_ERR_WIFI_NOT_CONNECT **(0x300f)**: Station still in disconnect status

ESP_ERR_WIFI_POST **(0x3012)**: Failed to post the event to WiFi task

ESP_ERR_WIFI_INIT_STATE **(0x3013)**: Invalid WiFi state when init/deinit is called

ESP_ERR_WIFI_STOP_STATE **(0x3014)**: Returned when WiFi is stopping

ESP_ERR_WIFI_NOT_ASSOC **(0x3015)**: The WiFi connection is not associated

ESP_ERR_WIFI_TX_DISALLOW **(0x3016)**: The WiFi TX is disallowed

ESP_ERR_WIFI_TWT_FULL **(0x3017)**: no available flow id

ESP_ERR_WIFI_TWT_SETUP_TIMEOUT **(0x3018)**: Timeout of receiving twt setup response frame, timeout times can be set during twt setup

ESP_ERR_WIFI_TWT_SETUP_TXFAIL **(0x3019)**: TWT setup frame tx failed

ESP_ERR_WIFI_TWT_SETUP_REJECT **(0x301a)**: The twt setup request was rejected by the AP

ESP_ERR_WIFI_DISCARD **(0x301b)**: Discard frame

ESP_ERR_WIFI_ROC_IN_PROGRESS **(0x301c)**: ROC op is in progress

ESP_ERR_WIFI_REGISTRAR **(0x3033)**: WPS registrar is not supported

ESP_ERR_WIFI_WPS_TYPE **(0x3034)**: WPS type error

ESP_ERR_WIFI_WPS_SM **(0x3035)**: WPS state machine is not initialized

ESP_ERR_ESPNOW_BASE **(0x3064)**: ESPNOW error number base.

ESP_ERR_ESPNOW_NOT_INIT **(0x3065)**: ESPNOW is not initialized.

ESP_ERR_ESPNOW_ARG **(0x3066)**: Invalid argument

ESP_ERR_ESPNOW_NO_MEM **(0x3067)**: Out of memory

ESP_ERR_ESPNOW_FULL **(0x3068)**: ESPNOW peer list is full

ESP_ERR_ESPNOW_NOT_FOUND **(0x3069)**: ESPNOW peer is not found

ESP_ERR_ESPNOW_INTERNAL **(0x306a)**: Internal error

ESP_ERR_ESPNOW_EXIST **(0x306b)**: ESPNOW peer has existed

ESP_ERR_ESPNOW_IF **(0x306c)**: Interface error

ESP_ERR_ESPNOW_CHAN **(0x306d)**: Channel error

ESP_ERR_DPP_FAILURE **(0x3097)**: Generic failure during DPP Operation

ESP_ERR_DPP_TX_FAILURE **(0x3098)**: DPP Frame Tx failed OR not Acked

ESP_ERR_DPP_INVALID_ATTR **(0x3099)**: Encountered invalid DPP Attribute

ESP_ERR_DPP_AUTH_TIMEOUT **(0x309a)**: DPP Auth response was not received in time

ESP_ERR_DPP_INVALID_LIST **(0x309b)**: Channel list given in esp_supp_dpp_bootstrap_gen() is not valid or too big

ESP_ERR_MESH_BASE **(0x4000)**: Starting number of MESH error codes

ESP_ERR_MESH_WIFI_NOT_START **(0x4001)**

ESP_ERR_MESH_NOT_INIT **(0x4002)**

ESP_ERR_MESH_NOT_CONFIG **(0x4003)**

ESP_ERR_MESH_NOT_START **(0x4004)**

ESP_ERR_MESH_NOT_SUPPORT **(0x4005)**

ESP_ERR_MESH_NOT_ALLOWED **(0x4006)**

ESP_ERR_MESH_NO_MEMORY **(0x4007)**

ESP_ERR_MESH_ARGUMENT **(0x4008)**

ESP_ERR_MESH_EXCEED_MTU **(0x4009)**

ESP_ERR_MESH_TIMEOUT **(0x400a)**

ESP_ERR_MESH_DISCONNECTED **(0x400b)**

ESP_ERR_MESH_QUEUE_FAIL **(0x400c)**

ESP_ERR_MESH_QUEUE_FULL **(0x400d)**

ESP_ERR_MESH_NO_PARENT_FOUND **(0x400e)**

ESP_ERR_MESH_NO_ROUTE_FOUND **(0x400f)**

ESP_ERR_MESH_OPTION_NULL **(0x4010)**

ESP_ERR_MESH_OPTION_UNKNOWN **(0x4011)**

ESP_ERR_MESH_XON_NO_WINDOW **(0x4012)**

ESP_ERR_MESH_INTERFACE **(0x4013)**

ESP_ERR_MESH_DISCARD_DUPLICATE **(0x4014)**

ESP_ERR_MESH_DISCARD **(0x4015)**

ESP_ERR_MESH_VOTING **(0x4016)**

ESP_ERR_MESH_XMIT **(0x4017)**

ESP_ERR_MESH_QUEUE_READ **(0x4018)**

ESP_ERR_MESH_PS **(0x4019)**

ESP_ERR_MESH_RECV_RELEASE **(0x401a)**

ESP_ERR_ESP_NETIF_BASE **(0x5000)**

ESP_ERR_ESP_NETIF_INVALID_PARAMS **(0x5001)**

ESP_ERR_ESP_NETIF_IF_NOT_READY **(0x5002)**

ESP_ERR_ESP_NETIF_DHCPC_START_FAILED **(0x5003)**

ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED **(0x5004)**

ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED **(0x5005)**

ESP_ERR_ESP_NETIF_NO_MEM **(0x5006)**

ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED **(0x5007)**

ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED **(0x5008)**

ESP_ERR_ESP_NETIF_INIT_FAILED **(0x5009)**

ESP_ERR_ESP_NETIF_DNS_NOT_CONFIGURED **(0x500a)**

ESP_ERR_ESP_NETIF_MLD6_FAILED **(0x500b)**

ESP_ERR_ESP_NETIF_IP6_ADDR_FAILED **(0x500c)**

ESP_ERR_ESP_NETIF_DHCPS_START_FAILED **(0x500d)**

ESP_ERR_FLASH_BASE **(0x6000)**: Starting number of flash error codes

ESP_ERR_FLASH_OP_FAIL **(0x6001)**

ESP_ERR_FLASH_OP_TIMEOUT **(0x6002)**

ESP_ERR_FLASH_NOT_INITIALISED **(0x6003)**

ESP_ERR_FLASH_UNSUPPORTED_HOST **(0x6004)**

ESP_ERR_FLASH_UNSUPPORTED_CHIP **(0x6005)**

ESP_ERR_FLASH_PROTECTED **(0x6006)**

ESP_ERR_HTTP_BASE **(0x7000)**: Starting number of HTTP error codes

ESP_ERR_HTTP_MAX_REDIRECT **(0x7001)**: The error exceeds the number of HTTP redirects

ESP_ERR_HTTP_CONNECT **(0x7002)**: Error open the HTTP connection

ESP_ERR_HTTP_WRITE_DATA **(0x7003)**: Error write HTTP data

ESP_ERR_HTTP_FETCH_HEADER **(0x7004)**: Error read HTTP header from server

ESP_ERR_HTTP_INVALID_TRANSPORT **(0x7005)**: There are no transport support for the input scheme

ESP_ERR_HTTP_CONNECTING **(0x7006)**: HTTP connection hasn't been established yet

ESP_ERR_HTTP_EAGAIN **(0x7007)**: Mapping of errno EAGAIN to esp_err_t

ESP_ERR_HTTP_CONNECTION_CLOSED **(0x7008)**: Read FIN from peer and the connection closed

ESP_ERR_ESP_TLS_BASE **(0x8000)**: Starting number of ESP-TLS error codes

ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME **(0x8001)**: Error if hostname couldn't be resolved upon tls connection

ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET **(0x8002)**: Failed to create socket

ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY **(0x8003)**: Unsupported protocol family

ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST **(0x8004)**: Failed to connect to host

ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED **(0x8005)**: failed to set/get socket option

ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT **(0x8006)**: new connection in esp_tls_low_level_conn connection timeouted

ESP_ERR_ESP_TLS_SE_FAILED **(0x8007)**

ESP_ERR_ESP_TLS_TCP_CLOSED_FIN **(0x8008)**

ESP_ERR_MBEDTLS_CERT_PARTLY_OK **(0x8010)**: mbedtls parse certificates was partly successful

ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED **(0x8011)**: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED **(0x8012)**: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED **(0x8013)**: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED **(0x8014)**: mbedtls api returned error

ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED **(0x8015)**: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_OWN_CERT_FAILED **(0x8016)**: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SETUP_FAILED **(0x8017)**: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_WRITE_FAILED **(0x8018)**: mbedtls api returned error

ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED **(0x8019)**: mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED **(0x801a)**: mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED **(0x801b)**: mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_TICKET_SETUP_FAILED **(0x801c)**: mbedtls api returned failed

ESP_ERR_WOLFSSL_SSL_SET_HOSTNAME_FAILED **(0x8031)**: wolfSSL api returned error

ESP_ERR_WOLFSSL_SSL_CONF_ALPN_PROTOCOLS_FAILED **(0x8032)**: wolfSSL api returned error

ESP_ERR_WOLFSSL_CERT_VERIFY_SETUP_FAILED **(0x8033)**: wolfSSL api returned error

ESP_ERR_WOLFSSL_KEY_VERIFY_SETUP_FAILED **(0x8034)**: wolfSSL api returned error

ESP_ERR_WOLFSSL_SSL_HANDSHAKE_FAILED **(0x8035)**: wolfSSL api returned failed

ESP_ERR_WOLFSSL_CTX_SETUP_FAILED **(0x8036)**: wolfSSL api returned failed

ESP_ERR_WOLFSSL_SSL_SETUP_FAILED **(0x8037)**: wolfSSL api returned failed

ESP_ERR_WOLFSSL_SSL_WRITE_FAILED **(0x8038)**: wolfSSL api returned failed

ESP_ERR_HTTPS_OTA_BASE **(0x9000)**

ESP_ERR_HTTPS_OTA_IN_PROGRESS **(0x9001)**

ESP_ERR_PING_BASE **(0xa000)**

ESP_ERR_PING_INVALID_PARAMS **(0xa001)**

ESP_ERR_PING_NO_MEM **(0xa002)**

ESP_ERR_HTTPD_BASE **(0xb000)**: Starting number of HTTPD error codes

ESP_ERR_HTTPD_HANDLERS_FULL **(0xb001)**: All slots for registering URI handlers have been consumed

ESP_ERR_HTTPD_HANDLER_EXISTS **(0xb002)**: URI handler with same method and target URI already registered

ESP_ERR_HTTPD_INVALID_REQ **(0xb003)**: Invalid request pointer

ESP_ERR_HTTPD_RESULT_TRUNC **(0xb004)**: Result string truncated

ESP_ERR_HTTPD_RESP_HDR **(0xb005)**: Response header field larger than supported

ESP_ERR_HTTPD_RESP_SEND **(0xb006)**: Error occurred while sending response packet

ESP_ERR_HTTPD_ALLOC_MEM **(0xb007)**: Failed to dynamically allocate memory for resource

ESP_ERR_HTTPD_TASK **(0xb008)**: Failed to launch server task/thread

ESP_ERR_HW_CRYPTO_BASE **(0xc000)**: Starting number of HW cryptography module error codes

ESP_ERR_HW_CRYPTO_DS_HMAC_FAIL **(0xc001)**: HMAC peripheral problem

ESP_ERR_HW_CRYPTO_DS_INVALID_KEY **(0xc002)**

ESP_ERR_HW_CRYPTO_DS_INVALID_DIGEST **(0xc004)**

ESP_ERR_HW_CRYPTO_DS_INVALID_PADDING **(0xc005)**

ESP_ERR_MEMPROT_BASE **(0xd000)**: Starting number of Memory Protection API error codes

ESP_ERR_MEMPROT_MEMORY_TYPE_INVALID **(0xd001)**

ESP_ERR_MEMPROT_SPLIT_ADDR_INVALID **(0xd002)**

ESP_ERR_MEMPROT_SPLIT_ADDR_OUT_OF_RANGE **(0xd003)**

ESP_ERR_MEMPROT_SPLIT_ADDR_UNALIGNED **(0xd004)**

ESP_ERR_MEMPROT_UNIMGMT_BLOCK_INVALID **(0xd005)**

ESP_ERR_MEMPROT_WORLD_INVALID **(0xd006)**

ESP_ERR_MEMPROT_AREA_INVALID **(0xd007)**

ESP_ERR_MEMPROT_CPUID_INVALID **(0xd008)**

ESP_ERR_TCP_TRANSPORT_BASE **(0xe000)**: Starting number of TCP Transport error codes

ESP_ERR_TCP_TRANSPORT_CONNECTION_TIMEOUT **(0xe001)**: Connection has timed out

ESP_ERR_TCP_TRANSPORT_CONNECTION_CLOSED_BY_FIN **(0xe002)**: Read FIN from peer and the connection has closed (in a clean way)

ESP_ERR_TCP_TRANSPORT_CONNECTION_FAILED **(0xe003)**: Failed to connect to the peer

ESP_ERR_TCP_TRANSPORT_NO_MEM **(0xe004)**: Memory allocation failed

ESP_ERR_NVS_SEC_BASE **(0xf000)**: Starting number of error codes

ESP_ERR_NVS_SEC_HMAC_KEY_NOT_FOUND **(0xf001)**: HMAC Key required to generate the NVS encryption keys not found

ESP_ERR_NVS_SEC_HMAC_KEY_BLK_ALREADY_USED **(0xf002)**: Provided eFuse block for HMAC key generation is already in use

ESP_ERR_NVS_SEC_HMAC_KEY_GENERATION_FAILED **(0xf003)**: Failed to generate/write the HMAC key to eFuse

ESP_ERR_NVS_SEC_HMAC_XTS_KEYS_DERIV_FAILED **(0xf004)**: Failed to derive the NVS encryption keys based on the HMAC-based scheme

警告： This document is not updated for ESP32P4 yet, so some of the content may not be correct.

This warning was automatically inserted due to the source file being in the *add_warnings_pages* list.

2.4 连网 API

2.4.1 以太网

以太网

概述 ESP-IDF 提供一系列灵活度高且兼具一致性的 API，为内部以太网 MAC (EMAC) 控制器和外部 SPI-Ethernet 模块提供支持。

本编程指南分为以下几个部分：

1. 以太网基本概念
2. 配置 *MAC* 和 *PHY*
3. 连接驱动程序至 *TCP/IP* 协议栈
4. 以太网驱动程序的杂项控制

以太网基本概念 以太网是一种异步的带冲突检测的载波侦听多路访问 (CSMA/CD) 协议/接口。通常来说，以太网不太适用于低功耗应用。然而，得益于其广泛的部署、高效的网络连接、高数据率以及范围不限的可扩展性，几乎所有的有线通信都可以通过以太网进行。

符合 IEEE 802.3 标准的正常以太网帧的长度在 64 至 1518 字节之间，由五个或六个不同的字段组成：目的地 MAC 地址 (DA)、源 MAC 地址 (SA)、类型/长度字段、数据有效载荷字段、可选的填充字段和帧校验序列字段 (CRC)。此外，在以太网上传输时，以太网数据包的开头需附加 7 字节的前导码和 1 字节的帧起始符 (SOF)。

因此，双绞线上的通信如图所示：

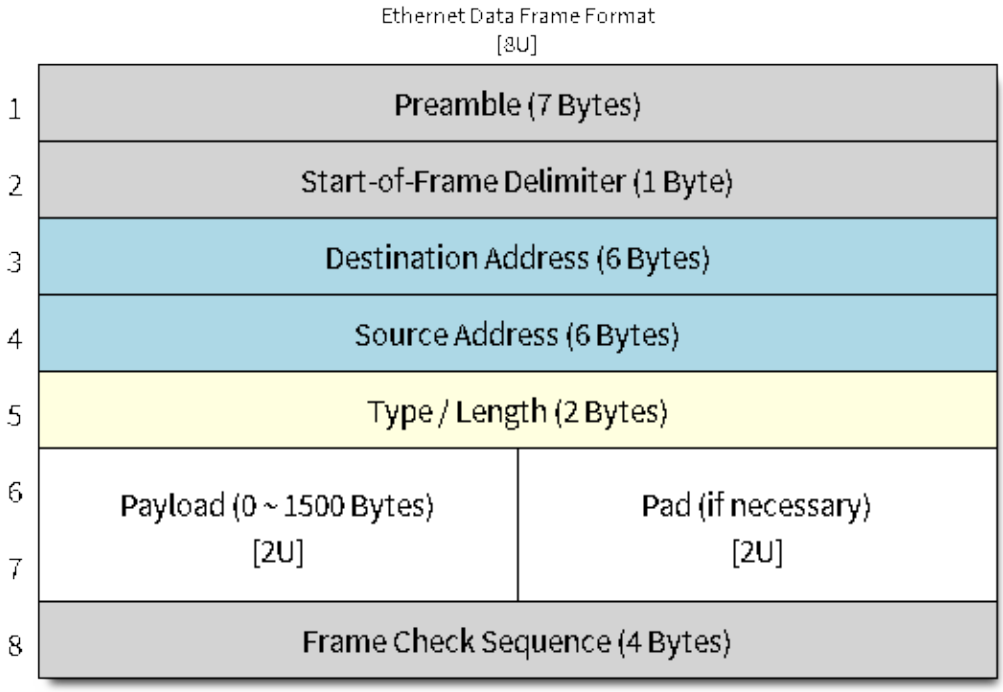


图 1: 以太网数据帧格式

前导码和帧起始符 前导码包含 7 字节的 `55H`，作用是使接收器在实际帧到达之前锁定数据流。

帧前界定符 (SFD) 为二进制序列 `10101011` （物理介质层可见）。有时它也被视作前导码的一部分。

在传输和接收数据时，协议将自动从数据包中生成/移除前导码和帧起始符。

目的地址 (DA) 目的地址字段包含一个 6 字节长的设备 MAC 地址，数据包将发送到该地址。如果 MAC 地址第一个字节中的最低有效位是 1，则该地址为组播地址。例如，01-00-00-F0-00 和 33-45-67-89-AB-CD 是组播地址，而 00-00-00-F0-00 和 32-45-67-89-AB-CD 不是。

带有组播地址的数据包将到达选定的一组以太网节点，并发挥重要作用。如果目的地址字段是保留的多播地址，即 FF-FF-FF-FF-FF-FF，则该数据包是一个广播数据包，指向共享网络中的每个对象。如果 MAC 地址的第一个字节中的最低有效位为 0，则该地址为单播地址，仅供寻址节点使用。

通常，EMAC 控制器会集成接收过滤器，用于丢弃或接收带有组播、广播和/或单播目的地址的数据包。传输数据包时，由主机控制器将所需的目标地址写入传输缓冲区。

源地址 (SA) 源地址字段包含一个 6 字节长的节点 MAC 地址，以太网数据包通过该节点创建。以太网的用户需为所使用的任意控制器生成唯一的 MAC 地址。MAC 地址由两部分组成：前三个字节称为组织唯一标识符 (OUI)，由 IEEE 分配；后三个字节是地址字节，由购买 OUI 的公司配置。有关 ESP-IDF 中使用的 MAC 地址的详细信息，请参见*MAC 地址分配*。

传输数据包时，由主机控制器将分配的源 MAC 地址写入传输缓冲区。

类型/长度 类型/长度字段长度为 2 字节。如果其值 <= 1500（十进制），则该字段为长度字段，指定在数据字段后的非填充数据量；如果其值 >= 1536，则该字段值表示后续数据包所属的协议。以下为该字段的常见值：

- IPv4 = 0800H

- IPv6 = 86DDH
- ARP = 0806H

使用专有网络的用户可以将此字段配置为长度字段。然而，对于使用互联网协议 (IP) 或地址解析协议 (ARP) 等协议的应用程序，在传输数据包时，应将此字段配置为协议规范定义的适当类型。

数据有效载荷 数据有效载荷字段是一个可变长度的字段，长度从 0 到 1500 字节不等。更大的数据包会因违反以太网标准而被大多数以太网节点丢弃。

数据有效载荷字段包含客户端数据，如 IP 数据报。

填充及帧校验序列 (FCS) 填充字段是一个可变长度的字段。数据有效载荷较小时，将添加填充字段以满足 IEEE 802.3 规范的要求。

以太网数据包的 DA、SA、类型、数据有效载荷和填充字段共计必须不小于 60 字节。加上所需的 4 字节 FCS 字段，数据包的长度必须不小于 64 字节。如果数据有效载荷字段小于 46 字节，则需要加上一个填充字段。

帧校验序列字段 (FCS) 长度为 4 字节，其中包含一个行业标准的 32 位 CRC，该 CRC 是根据 DA、SA、类型、数据有效载荷和填充字段的数据计算的。鉴于计算 CRC 的复杂性，硬件通常会自动生成一个有效的 CRC 进行传输。否则，需由主机控制器生成 CRC 并将其写入传输缓冲区。

通常情况下，主机控制器无需关注填充字段和 CRC 字段，因为这两部分可以在传输或接收时由硬件 EMAC 自动生成或验证。然而，当数据包到达时，填充字段和 CRC 字段将被写入接收缓冲区。因此，如果需要的话，主机控制器也可以对它们进行评估。

备注： 除了上述的基本数据帧，在 10/100 Mbps 以太网中还有两种常见的帧类型：控制帧和 VLAN 标记帧。ESP-IDF 不支持这两种帧类型。

配置 MAC 和 PHY 以太网驱动器由两部分组成：MAC 和 PHY。

MAC 和 PHY 之间的通信可以通过多种方式进行，如：**MII** （媒体独立接口）、**RMII** （简化媒体独立接口）等。

MII 和 RMII 的一个明显区别在于其所需的信号数。MII 通常需要多达 18 个信号，RMII 接口则仅需要 9 个信号。

备注： ESP-IDF 只支持 RMII 接口，所以请将*eth_esp32_emac_config_t::interface* 设置为*eth_data_interface_t::EMAC_DATA_INTERFACE_RMII*。

在 RMII 模式下，接收器和发射器信号的参考时钟为 REF_CLK。**在访问 PHY 和 MAC 时，REF_CLK 必须保持稳定**。一般来说，根据设计中 PHY 设备的特征，可通过以下三种方式生成 REF_CLK：

- 一些 PHY 芯片可以从其外部连接的 25 MHz 晶体振荡器中衍生出 REF_CLK （如图中的选项 **a** 所示） 。对于此类芯片，请将*eth_esp32_emac_config_t::clock_config* 中的*eth_mac_clock_config_t::clock_mode* 设置为*emac_rmii_clock_mode_t::EMAC_CLK_EXT_IN*。
- 一些 PHY 芯片使用外接的 50 MHz 晶体振荡器或其他时钟源作为 MAC 端的 REF_CLK （如图中的选项 **b** 所示）。对于此类芯片，请同样将*eth_esp32_emac_config_t::clock_config* 中的*eth_mac_clock_config_t::clock_mode* 设置为*emac_rmii_clock_mode_t::EMAC_CLK_EXT_IN*。
- 一些 EMAC 控制器可以使用其内部的高精度 PLL 生成 REF_CLK （如图中的选项 **c** 所示） 。此种情况下，请将*eth_esp32_emac_config_t::clock_config* 中的*eth_mac_clock_config_t::clock_mode* 设置为*emac_rmii_clock_mode_t::EMAC_CLK_OUT*。

备注： 如果 RMII 时钟模式配置为*emac_rmii_clock_mode_t::EMAC_CLK_OUT*，则可以通过 IO_MUX 将 GPIO23 and GPIO39 选择为 REF_CLK 信号的输出管脚。

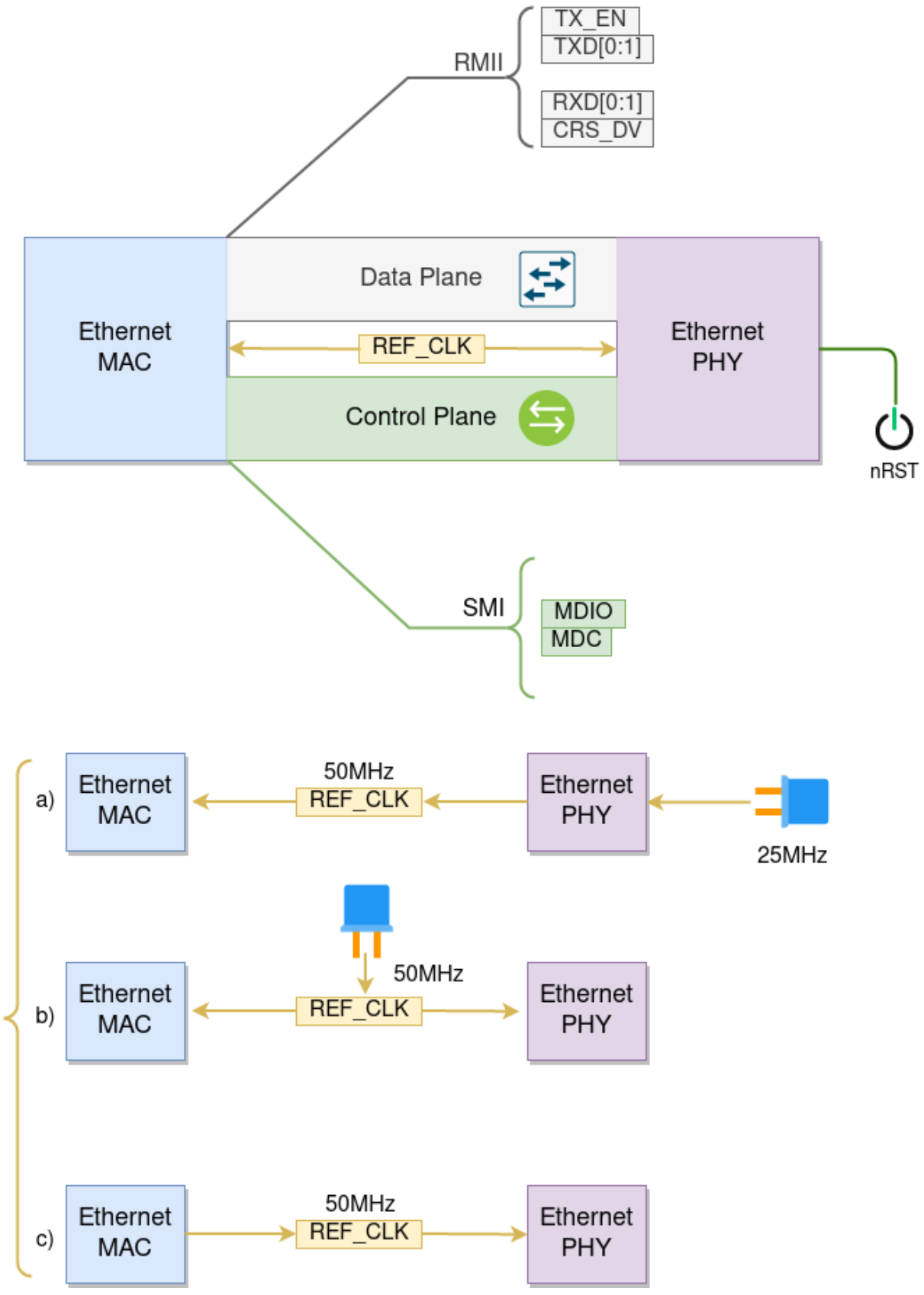


图 2: 以太网 RMII 接口

如果 RMII 时钟模式配置为*emac_rmii_clock_mode_t::EMAC_CLK_EXT_IN*，则可以通过 IO_MUX 将 GPIO32, GPIO44 and GPIO50 选择为 REF_CLK 信号的输入管脚。

警告： 如果 RMII 时钟模式配置为*emac_rmii_clock_mode_t::EMAC_CLK_OUT*，则必须从外部将 REF_CLK 输出信号回环到 EMAC。请将*eth_esp32_emac_config_t::clock_config_out_in* 中的*eth_mac_clock_config_t::clock_mode* 配置为*emac_rmii_clock_mode_t::EMAC_CLK_EXT_IN*，并选择与 REF_CLK 输入 GPIO (GPIO32, GPIO44 and GPIO50) 相关联的 GPIO 编号。

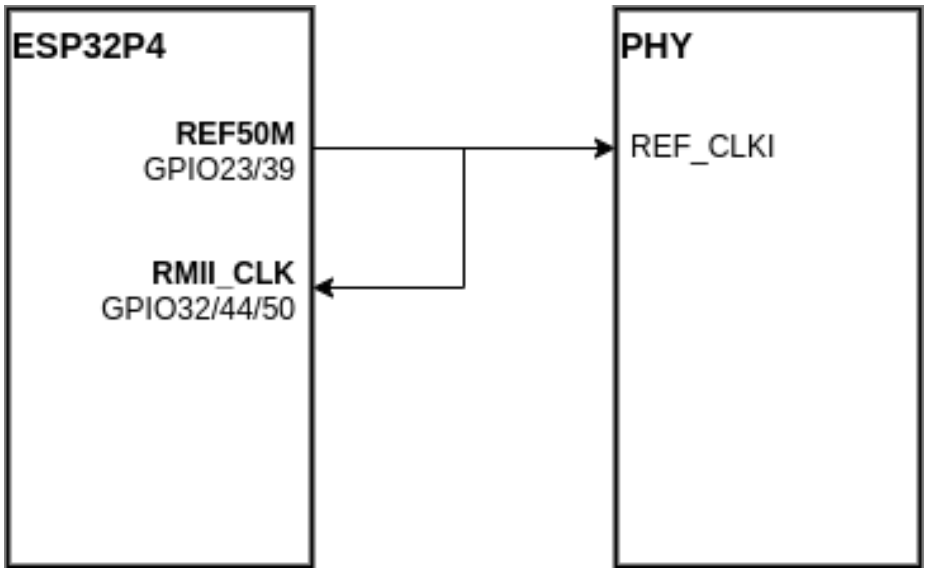


图 3: RMII REF_CKL 输出回环

无论选择哪种 RMII 时钟模式，都请确保硬件设计中 REF_CLK 的信号完整性！信号线越短越好，并远离 RF 设备和电感。

备注： 数据平面中使用的信号可以通过 IO_MUX 配置为 RMII 预定义的 GPIO，请参阅下表。数据平面 GPIO 配置由驱动程序根据*eth_esp32_emac_config_t::emac_dataif_gpio* 的内容执行。控制平面中使用的信号可以通过 GPIO 矩阵路由到任何空闲的 GPIO。

表 2: ESP32-P4 RMII 数据平面 GPIO

管脚名	GPIO 编号
TX_EN	GPIO33, GPIO40 and GPIO49
TXD0	GPIO34 and GPIO41
TXD1	GPIO35 and GPIO42
CRS_DV	GPIO28, GPIO45 and GPIO51
RXD0	GPIO29, GPIO46 and GPIO52
RXD1	GPIO30, GPIO47 and GPIO53

根据以太网板设计，需要分别为 MAC 和 PHY 配置必要的参数，通过两者完成驱动程序的安装。

MAC 的相关配置可以在*eth_mac_config_t* 中找到，具体包括：

- *eth_mac_config_t::sw_reset_timeout_ms*：软件复位超时值，单位为毫秒。通常，MAC 复位应在 100 ms 内完成。
- *eth_mac_config_t::rx_task_stack_size* 和*eth_mac_config_t::rx_task_prio*：MAC 驱动会创建一个专门的任务来处理传入的数据包，这两个参数用于设置该任务的堆栈大小和

优先级。
- *eth_mac_config_t::flags*：指定 MAC 驱动应支持的额外功能，尤其适用于某些特殊情况。这个字段的值支持与以 ETH_MAC_FLAG_ 为前缀的宏进行 OR 运算。例如，如果要求 MAC 驱动程序在 cache 禁用时仍能正常工作，那么则需要用*ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE* 配置这个字段。

eth_esp32_emac_config_t 描述了 **内部 MAC 模块**的特定配置，其中包括：

- *eth_esp32_emac_config_t::smi_mdc_gpio_num*和*eth_esp32_emac_config_t::smi_mdio_gpio_nu* 连接 SMI 信号的 GPIO 编号。
- *eth_esp32_emac_config_t::interface*：配置到 PHY (MII/RMII) 的 MAC 数据接口。
- *eth_esp32_emac_config_t::clock_config*：配置 EMAC 接口时钟（RMII 模式下的 REF_CLK 模式以及 GPIO 编号）。
- *eth_esp32_emac_config_t::intr_priority*：设置 MAC 中断的优先级。如果设置为 0 或负值，则驱动程序将分配一个具有默认优先级的中断。否则，驱动程序将使用给定的优先级。请注意，可以设置 低、中中断优先级（1 到 3），因为这可以在 C 中处理。
- *eth_esp32_emac_config_t::emac_dataif_gpio*：EMAC MII/RMII 数据平面 GPIO 编号配置。
- *eth_esp32_emac_config_t::clock_config_out_in*：当 REF_CLK 信号在内部生成并从外部回环到 EMAC 时，配置 EMAC 输入接口时钟。必须始终将 EMAC 的模式配置为*emac_rmii_clock_mode_t::EMAC_CLK_EXT_IN*。此选项仅在*eth_esp32_emac_config_t::clock_config* 的配置设置为*emac_rmii_clock_mode_t::EMAC_CLK_OUT* 时有效。

PHY 的相关配置可以在*eth_phy_config_t* 中找到，具体包括：

- *eth_phy_config_t::phy_addr*：同一条 SMI 总线上可以存在多个 PHY 设备，所以有必要为各个 PHY 设备分配唯一地址。通常，这个地址是在硬件设计期间，通过拉高/拉低一些 PHY strapping 管脚来配置的。根据不同的以太网开发板，可配置值为 0 到 15。需注意，如果 SMI 总线上仅有一个 PHY 设备，将该值配置为 -1，即可使驱动程序自动检测 PHY 地址。
- *eth_phy_config_t::reset_timeout_ms*：复位超时值，单位为毫秒。通常，PHY 复位应在 100 ms 内完成。
- *eth_phy_config_t::autonego_timeout_ms*：自动协商超时值，单位为毫秒。以太网驱动程序会与链路另一端的设备进行自协商，以确定连接的最佳双工模式和速率。此值通常取决于电路板上 PHY 设备的性能。
- *eth_phy_config_t::reset_gpio_num*：如果开发板同时将 PHY 复位管脚连接至了任意 GPIO 管脚，请使用该字段进行配置。否则，配置为 -1。

ESP-IDF 在宏*ETH_MAC_DEFAULT_CONFIG* 和*ETH_PHY_DEFAULT_CONFIG* 中为 MAC 和 PHY 提供了默认配置。

创建 MAC 和 PHY 实例 以太网驱动是以面向对象的方式实现的。对 MAC 和 PHY 的任何操作都应基于实例。

内部 EMAC + 外部 PHY

```
eth_mac_config_t mac_config = ETH_MAC_DEFAULT_CONFIG();                          //␣
↪应用默认的通用 MAC 配置
eth_esp32_emac_config_t esp32_emac_config = ETH_ESP32_EMAC_DEFAULT_CONFIG(); //␣
↪应用默认的供应商特定 MAC 配置
esp32_emac_config.smi_mdc_gpio_num = CONFIG_EXAMPLE_ETH_MDC_GPIO;            //␣
↪更改用于 MDC 信号的 GPIO
esp32_emac_config.smi_mdio_gpio_num = CONFIG_EXAMPLE_ETH_MDIO_GPIO;          //␣
↪更改用于 MDIO 信号的 GPIO
esp_eth_mac_t *mac = esp_eth_mac_new_esp32(&esp32_emac_config, &mac_config); //␣
↪创建 MAC 实例
```

(下页继续)

(续上页)

```
eth_phy_config_t phy_config = ETH_PHY_DEFAULT_CONFIG();      // 应用默认的 PHY 配置
phy_config.phy_addr = CONFIG_EXAMPLE_ETH_PHY_ADDR;           // 根据开发板设计更改␣
↪PHY 地址
phy_config.reset_gpio_num = CONFIG_EXAMPLE_ETH_PHY_RST_GPIO; // 更改用于 PHY␣
↪复位的 GPIO
esp_eth_phy_t *phy = esp_eth_phy_new_ip101(&phy_config);     // 创建 PHY 实例
// ESP-IDF 为数种以太网 PHY 芯片驱动提供官方支持
// esp_eth_phy_t *phy = esp_eth_phy_new_rtl8201(&phy_config);
// esp_eth_phy_t *phy = esp_eth_phy_new_lan8720(&phy_config);
// esp_eth_phy_t *phy = esp_eth_phy_new_dp83848(&phy_config);
```

可选的运行时 MAC 时钟配置 可以通过用户应用程序代码，选择性配置 EMAC 中的 REF_CLK。

```
eth_esp32_emac_config_t esp32_emac_config = ETH_ESP32_EMAC_DEFAULT_CONFIG(); //␣
↪应用默认的供应商特定 MAC 配置

// ...

esp32_emac_config.interface = EMAC_DATA_INTERFACE_RMII;                      //␣
↪更改 EMAC 数据接口
esp32_emac_config.clock_config.rmii.clock_mode = EMAC_CLK_OUT;               //␣
↪配置 EMAC REF_CLK 模式
esp32_emac_config.clock_config.rmii.clock_gpio = EMAC_CLK_OUT_GPIO;          //␣
↪配置用于输入/输出 EMAC REF_CLK 的 GPIO 编号
esp_eth_mac_t *mac = esp_eth_mac_new_esp32(&esp32_emac_config, &mac_config); //␣
↪创建 MAC 实例
```

SPI-Ethernet 模块

```
eth_mac_config_t mac_config = ETH_MAC_DEFAULT_CONFIG();      // 应用默认的通用 MAC␣
↪配置
eth_phy_config_t phy_config = ETH_PHY_DEFAULT_CONFIG();      // 应用默认的 PHY 配置
phy_config.phy_addr = CONFIG_EXAMPLE_ETH_PHY_ADDR;           // 根据开发板设计更改␣
↪PHY 地址
phy_config.reset_gpio_num = CONFIG_EXAMPLE_ETH_PHY_RST_GPIO; // 更改用于 PHY␣
↪复位的 GPIO
// 安装 GPIO 中断服务（因为 SPI-Ethernet 模块为中断驱动）
gpio_install_isr_service(0);
// 配置 SPI 总线
spi_device_handle_t spi_handle = NULL;
spi_bus_config_t buscfg = {
    .miso_io_num = CONFIG_EXAMPLE_ETH_SPI_MISO_GPIO,
    .mosi_io_num = CONFIG_EXAMPLE_ETH_SPI_MOSI_GPIO,
    .sclk_io_num = CONFIG_EXAMPLE_ETH_SPI_SCLK_GPIO,
    .quadwp_io_num = -1,
    .quadhd_io_num = -1,
};
ESP_ERROR_CHECK(spi_bus_initialize(CONFIG_EXAMPLE_ETH_SPI_HOST, &buscfg, 1));
// 配置 SPI 从机设备
spi_device_interface_config_t spi_devcfg = {
    .mode = 0,
    .clock_speed_hz = CONFIG_EXAMPLE_ETH_SPI_CLOCK_MHZ * 1000 * 1000,
    .spics_io_num = CONFIG_EXAMPLE_ETH_SPI_CS_GPIO,
    .queue_size = 20
};
/* dm9051 ethernet driver is based on spi driver */
eth_dm9051_config_t dm9051_config = ETH_DM9051_DEFAULT_CONFIG(CONFIG_EXAMPLE_ETH_
↪SPI_HOST, &spi_devcfg);
dm9051_config.int_gpio_num = CONFIG_EXAMPLE_ETH_SPI_INT_GPIO;
```

(下页继续)

(续上页)

```
esp_eth_mac_t *mac = esp_eth_mac_new_dm9051(&dm9051_config, &mac_config);
esp_eth_phy_t *phy = esp_eth_phy_new_dm9051(&phy_config);
```

备注:

- 当为 SPI-Ethernet 模块（例如 DM9051）创建 MAC 和 PHY 实例时，由于 PHY 是集成在模块中的，因此调用的实例创建函数的后缀须保持一致（例如 *esp_eth_mac_new_dm9051* 和 *esp_eth_phy_new_dm9051* 搭配使用）。
- 针对不同的以太网模块，或是为了满足特定 PCB 上的 SPI 时序，SPI 从机设备配置（即 *spi_device_interface_config_t*）可能略有不同。具体配置请查看模块规格以及 ESP-IDF 中的示例。

安装驱动程序 安装以太网驱动程序需要结合 MAC 和 PHY 实例，并在*esp_eth_config_t* 中配置一些额外的高级选项（即不仅限于 MAC 或 PHY 的选项）：

- *esp_eth_config_t::mac*：由 MAC 生成器创建的实例（例如*esp_eth_mac_new_esp32()*）。
- *esp_eth_config_t::phy*：由 PHY 生成器创建的实例（例如*esp_eth_phy_new_ip101()*）。
- *esp_eth_config_t::check_link_period_ms*：以太网驱动程序会启用操作系统定时器来定期检查链接状态。该字段用于设置间隔时间，单位为毫秒。
- *esp_eth_config_t::stack_input*：在大多数的以太网物联网应用中，驱动器接收的以太网帧会被传递到上层（如 TCP/IP 栈）。经配置，该字段为负责处理传入帧的函数。可以在安装驱动程序后，通过函数*esp_eth_update_input_path()* 更新该字段。该字段支持在运行过程中进行更新。
- *esp_eth_config_t::on_lowlevel_init_done* 和*esp_eth_config_t::on_lowlevel_deinit_done*：这两个字段用于指定钩子函数，当去初始化或初始化低级别硬件时，会调用钩子函数。

ESP-IDF 在宏*ETH_DEFAULT_CONFIG* 中为安装驱动程序提供了一个默认配置。

```
esp_eth_config_t config = ETH_DEFAULT_CONFIG(mac, phy); // 应用默认驱动程序配置
esp_eth_handle_t eth_handle = NULL; // 驱动程序安装完毕后，将得到驱动程序的句柄
esp_eth_driver_install(&config, &eth_handle); // 安装驱动程序
```

以太网驱动程序包含事件驱动模型，该模型会向用户空间发送有用及重要的事件。安装以太网驱动程序之前，需要首先初始化事件循环。有关事件驱动编程的更多信息，请参考事件循环库。

```
/** 以太网事件的事件处理程序 */
static void eth_event_handler(void *arg, esp_event_base_t event_base,
                              int32_t event_id, void *event_data)
{
    uint8_t mac_addr[6] = {0};
    /* 可从事件数据中获得以太网驱动句柄 */
    esp_eth_handle_t eth_handle = *(esp_eth_handle_t *)event_data;

    switch (event_id) {
    case ETHERNET_EVENT_CONNECTED:
        esp_eth_ioctl(eth_handle, ETH_CMD_G_MAC_ADDR, mac_addr);
        ESP_LOGI(TAG, "Ethernet Link Up");
        ESP_LOGI(TAG, "Ethernet HW Addr %02x:%02x:%02x:%02x:%02x:%02x",
                 mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_
↪addr[4], mac_addr[5]);
        break;
    case ETHERNET_EVENT_DISCONNECTED:
        ESP_LOGI(TAG, "Ethernet Link Down");
        break;
    case ETHERNET_EVENT_START:
        ESP_LOGI(TAG, "Ethernet Started");
        break;
    case ETHERNET_EVENT_STOP:
        ESP_LOGI(TAG, "Ethernet Stopped");
```

(下页继续)

(续上页)

```
        break;
    default:
        break;
    }
}

esp_event_loop_create_default(); // 创建一个在后台运行的默认事件循环
esp_event_handler_register(ETH_EVENT, ESP_EVENT_ANY_ID, &eth_event_handler, NULL);␣
↪// 注册以太网事件处理程序（用于在发生 link up/down␣
↪等事件时，处理特定的用户相关内容）
```

启动以太网驱动程序 安装驱动程序后，可以立即启动以太网。

```
esp_eth_start(eth_handle); // 启动以太网驱动程序状态机
```

连接驱动程序至 TCP/IP 协议栈 现在，以太网驱动程序已经完成安装。但对应 OSI（开放式系统互连模型）来看，目前阶段仍然属于第二层（即数据链路层）。这意味着可以检测到 link up/down 事件，获得用户空间的 MAC 地址，但无法获得 IP 地址，当然也无法发送 HTTP 请求。ESP-IDF 中使用的 TCP/IP 协议栈是 LwIP，关于 LwIP 的更多信息，请参考*LwIP*。

要将以太网驱动程序连接至 TCP/IP 协议栈，需要以下三步：

1. 为以太网驱动程序创建网络接口
2. 将网络接口连接到以太网驱动程序
3. 注册 IP 事件处理程序

有关网络接口的更多信息，请参考*Network Interface*。

```
/** IP_EVENT_ETH_GOT_IP 的事件处理程序 */
static void got_ip_event_handler(void *arg, esp_event_base_t event_base,
                                 int32_t event_id, void *event_data)
{
    ip_event_got_ip_t *event = (ip_event_got_ip_t *) event_data;
    const esp_netif_ip_info_t *ip_info = &event->ip_info;

    ESP_LOGI(TAG, "Ethernet Got IP Address");
    ESP_LOGI(TAG, "~~~~~~~~~~~");
    ESP_LOGI(TAG, "ETHIP:" IPSTR, IP2STR(&ip_info->ip));
    ESP_LOGI(TAG, "ETHMASK:" IPSTR, IP2STR(&ip_info->netmask));
    ESP_LOGI(TAG, "ETHGW:" IPSTR, IP2STR(&ip_info->gw));
    ESP_LOGI(TAG, "~~~~~~~~~~~");
}

esp_netif_init()); // 初始化 TCP/IP 网络接口（在应用程序中应仅调用一次）
esp_netif_config_t cfg = ESP_NETIF_DEFAULT_ETH(); // 应用以太网的默认网络接口配置
esp_netif_t *eth_netif = esp_netif_new(&cfg); // 为以太网驱动程序创建网络接口

esp_netif_attach(eth_netif, esp_eth_new_netif_glue(eth_handle)); //␣
↪将以太网驱动程序连接至 TCP/IP 协议栈
esp_event_handler_register(IP_EVENT, IP_EVENT_ETH_GOT_IP, &got_ip_event_handler,␣
↪NULL); // 注册用户定义的 IP 事件处理程序
esp_eth_start(eth_handle); // 启动以太网驱动程序状态机
```

警告： 推荐在完成整个以太网驱动和网络接口的初始化后，再注册用户定义的以太网/IP 事件处理程序，也就是把注册事件处理程序作为启动以太网驱动程序的最后一步。这样可以确保以太网驱动程序或网络接口将首先执行以太网/IP 事件，从而保证在执行用户定义的处理程序时，系统处于预期状态。

以太网驱动程序的杂项控制 以下功能只支持在安装以太网驱动程序后调用。

- 关闭以太网驱动程序：*esp_eth_stop()*
- 更新以太网数据输入路径：*esp_eth_update_input_path()*
- 获取/设置以太网驱动程序杂项内容：*esp_eth_ioctl()*

```
/* 获取 MAC 地址 */
uint8_t mac_addr[6];
memset(mac_addr, 0, sizeof(mac_addr));
esp_eth_ioctl(eth_handle, ETH_CMD_G_MAC_ADDR, mac_addr);
ESP_LOGI(TAG, "Ethernet MAC Address: %02x:%02x:%02x:%02x:%02x:%02x",
         mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_
↪addr[5]);

/* 获取 PHY 地址 */
int phy_addr = -1;
esp_eth_ioctl(eth_handle, ETH_CMD_G_PHY_ADDR, &phy_addr);
ESP_LOGI(TAG, "Ethernet PHY Address: %d", phy_addr);
```

数据流量控制 受 RAM 大小限制，在网络拥堵时，MCU 上的以太网通常仅能处理有限数量的帧。发送站的数据传输速度可能快于对等端的接收能力。以太网数据流量控制机制允许接收节点向发送方发出信号，要求暂停传输，直到接收方跟上。这项功能是通过暂停帧实现的，该帧定义在 IEEE 802.3x 中。

暂停帧是一种特殊的以太网帧，用于携带暂停命令，其 EtherType 字段为 `0x8808`，控制操作码为 `0x0001`。只有配置为全双工操作的节点组可以发送暂停帧。当节点组希望暂停链路的另一端时，它会发送一个暂停帧到 48 位的保留组播地址 `01-80-C2-00-00-01`。暂停帧中也包括请求暂停的时间段，以两字节的整数形式发送，值的范围从 `0` 到 `65535`。

安装以太网驱动程序后，数据流量控制功能默认禁用，可以通过以下方式启用此功能：

```
bool flow_ctrl_enable = true;
esp_eth_ioctl(eth_handle, ETH_CMD_S_FLOW_CTRL, &flow_ctrl_enable);
```

需注意，暂停帧是在自动协商期间由 PHY 向对等端公布的。只有当链路两端都支持暂停帧时，以太网驱动程序才会发送暂停帧。

应用示例

- 以太网基本示例：ethernet/basic
- 以太网 iperf 示例：ethernet/iperf
- 以太网到 Wi-Fi AP “路由器”：network/eth2ap
- Wi-Fi station 到以太网 “网桥”：network/sta2eth
- 大多数协议示例也适用于以太网：protocols

进阶操作

自定义 PHY 驱动程序 市面上有多家 PHY 芯片制造商提供各种类型的芯片。ESP-IDF 现已支持数种 PHY 芯片，但是由于价格、功能、库存等原因，有时用户还是无法找到一款能满足其实际需求的芯片。

好在 IEEE 802.3 在其 22.2.4 管理功能部分对 EMAC 和 PHY 之间的管理接口进行了标准化。该部分定义了所谓的” MII 管理接口” 规范，用于控制 PHY 和收集 PHY 的状态，还定义了一组管理寄存器来控制芯片行为、链接属性、自动协商配置等。在 ESP-IDF 中，这项基本的管理功能是由 esp_eth/src/phy/esp_eth_phy_802_3.c 实现的，这也大大降低了创建新的自定义 PHY 芯片驱动的难度。

备注： 由于一些 PHY 芯片可能不符合 IEEE 802.3 第 22.2.4 节的规定，所以请首先查看 PHY 数据手册。不过，就算芯片不符合规定，依旧可以创建自定义 PHY 驱动程序，只是由于需要自行定义所有的 PHY 管理功能，这个过程将变得较为复杂。

ESP-IDF 以太网驱动程序所需的大部分 PHY 管理功能都已涵盖在 esp_eth/src/phy/esp_eth_phy_802_3.c 中。不过对于以下几项，可能仍需针对不同芯片开发具体的管理功能：

- 链接状态。此项总是由使用的具体芯片决定
- 芯片初始化。即使不存在严格的限制，也应进行自定义，以确保使用的是符合预期的芯片
- 芯片的具体功能配置

创建自定义 PHY 驱动程序的步骤：

1. 请根据 PHY 数据手册，定义针对供应商的特定注册表布局。示例请参见 esp_eth/src/phy/esp_eth_phy_ip101.c。
2. 准备衍生的 PHY 管理对象信息结构，该结构：
   - 必须至少包含 IEEE 802.3 *phy_802_3_t* 父对象
   - 可选择包含额外的变量，以支持非 IEEE 802.3 或定制功能。示例请参见 esp_eth/src/phy/esp_eth_phy_ksz80xx.c。
3. 定义针对芯片的特定管理回调功能。
4. 初始化 IEEE 802.3 父对象并重新分配针对芯片的特定管理回调功能。

实现新的自定义 PHY 驱动程序后，你可以通过 ESP-IDF 组件管理中心 将驱动分享给其他用户。

API 参考

Header File

- components/hal/include/hal/eth_types.h
- This header file can be included with:

```
#include "hal/eth_types.h"
```

Enumerations

enum **eth_data_interface_t**

Ethernet interface.

Values:

enumerator **EMAC_DATA_INTERFACE_RMII**

Reduced Media Independent Interface

enumerator **EMAC_DATA_INTERFACE_MII**

Media Independent Interface

enum **eth_link_t**

Ethernet link status.

Values:

enumerator **ETH_LINK_UP**

Ethernet link is up

enumerator **ETH_LINK_DOWN**

Ethernet link is down

enum **eth_speed_t**

Ethernet speed.

Values:

enumerator **ETH_SPEED_10M**

Ethernet speed is 10Mbps

enumerator **ETH_SPEED_100M**

Ethernet speed is 100Mbps

enumerator **ETH_SPEED_MAX**

Max speed mode (for checking purpose)

enum **eth_duplex_t**

Ethernet duplex mode.

Values:

enumerator **ETH_DUPLEX_HALF**

Ethernet is in half duplex

enumerator **ETH_DUPLEX_FULL**

Ethernet is in full duplex

enum **eth_checksum_t**

Ethernet Checksum.

Values:

enumerator **ETH_CHECKSUM_SW**

Ethernet checksum calculate by software

enumerator **ETH_CHECKSUM_HW**

Ethernet checksum calculate by hardware

enum **eth_mac_dma_burst_len_t**

Internal ethernet EMAC's DMA available burst sizes.

Values:

enumerator **ETH_DMA_BURST_LEN_32**

enumerator **ETH_DMA_BURST_LEN_16**

enumerator **ETH_DMA_BURST_LEN_8**

enumerator **ETH_DMA_BURST_LEN_4**

enumerator **ETH_DMA_BURST_LEN_2**

enumerator **ETH_DMA_BURST_LEN_1**

Header File

- components/esp_eth/include/esp_eth.h
- This header file can be included with:

```
#include "esp_eth.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Header File

- components/esp_eth/include/esp_eth_driver.h
- This header file can be included with:

```
#include "esp_eth_driver.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Functions

esp_err_t **esp_eth_driver_install** (const *esp_eth_config_t* *config, *esp_eth_handle_t* *out_hdl)

Install Ethernet driver.

参数

- **config** -- **[in]** configuration of the Ethernet driver
- **out_hdl** -- **[out]** handle of Ethernet driver

返回

- ESP_OK: install esp_eth driver successfully
- ESP_ERR_INVALID_ARG: install esp_eth driver failed because of some invalid argument
- ESP_ERR_NO_MEM: install esp_eth driver failed because there's no memory for driver
- ESP_FAIL: install esp_eth driver failed because some other error occurred

esp_err_t **esp_eth_driver_uninstall** (*esp_eth_handle_t* hdl)

Uninstall Ethernet driver.

备注: It's not recommended to uninstall Ethernet driver unless it won't get used any more in application code. To uninstall Ethernet driver, you have to make sure, all references to the driver are released. Ethernet driver can only be uninstalled successfully when reference counter equals to one.

参数 **hdl** -- **[in]** handle of Ethernet driver

返回

- ESP_OK: uninstall esp_eth driver successfully
- ESP_ERR_INVALID_ARG: uninstall esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: uninstall esp_eth driver failed because it has more than one reference

- ESP_FAIL: uninstall esp_eth driver failed because some other error occurred

esp_err_t **esp_eth_start**(*esp_eth_handle_t* hdl)

Start Ethernet driver **ONLY** in standalone mode (i.e. without TCP/IP stack)

备注: This API will start driver state machine and internal software timer (for checking link status).

参数 **hdl** -- **[in]** handle of Ethernet driver

返回

- ESP_OK: start esp_eth driver successfully
- ESP_ERR_INVALID_ARG: start esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: start esp_eth driver failed because driver has started already
- ESP_FAIL: start esp_eth driver failed because some other error occurred

esp_err_t **esp_eth_stop**(*esp_eth_handle_t* hdl)

Stop Ethernet driver.

备注: This function does the oppsite operation of `esp_eth_start`.

参数 **hdl** -- **[in]** handle of Ethernet driver

返回

- ESP_OK: stop esp_eth driver successfully
- ESP_ERR_INVALID_ARG: stop esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: stop esp_eth driver failed because driver has not started yet
- ESP_FAIL: stop esp_eth driver failed because some other error occurred

esp_err_t **esp_eth_update_input_path**(*esp_eth_handle_t* hdl, *esp_err_t* (*stack_input)(*esp_eth_handle_t* hdl, uint8_t *buffer, uint32_t length, void *priv), void *priv)

Update Ethernet data input path (i.e. specify where to pass the input buffer)

备注: After install driver, Ethernet still don't know where to deliver the input buffer. In fact, this API registers a callback function which get invoked when Ethernet received new packets.

参数

- **hdl** -- **[in]** handle of Ethernet driver
- **stack_input** -- **[in]** function pointer, which does the actual process on incoming packets
- **priv** -- **[in]** private resource, which gets passed to `stack_input` callback without any modification

返回

- ESP_OK: update input path successfully
- ESP_ERR_INVALID_ARG: update input path failed because of some invalid argument
- ESP_FAIL: update input path failed because some other error occurred

esp_err_t **esp_eth_transmit**(*esp_eth_handle_t* hdl, void *buf, size_t length)

General Transmit.

参数

- **hdl** -- **[in]** handle of Ethernet driver
- **buf** -- **[in]** buffer of the packet to transfer
- **length** -- **[in]** length of the buffer to transfer

返回

- ESP_OK: transmit frame buffer successfully
- ESP_ERR_INVALID_ARG: transmit frame buffer failed because of some invalid argument
- ESP_ERR_INVALID_STATE: invalid driver state (e.i. driver is not started)
- ESP_ERR_TIMEOUT: transmit frame buffer failed because HW was not get available in predefined period
- ESP_FAIL: transmit frame buffer failed because some other error occurred

esp_err_t **esp_eth_transmit_vargs** (*esp_eth_handle_t* hdl, uint32_t argc, ...)

Special Transmit with variable number of arguments.

参数

- **hdl** -- **[in]** handle of Ethernet driver
- **argc** -- **[in]** number variable arguments
- ... -- variable arguments

返回

- ESP_OK: transmit successful
- ESP_ERR_INVALID_STATE: invalid driver state (e.i. driver is not started)
- ESP_ERR_TIMEOUT: transmit frame buffer failed because HW was not get available in predefined period
- ESP_FAIL: transmit frame buffer failed because some other error occurred

esp_err_t **esp_eth_ioctl** (*esp_eth_handle_t* hdl, *esp_eth_io_cmd_t* cmd, void *data)

Misc IO function of Ethernet driver.

The following common IO control commands are supported:

- ETH_CMD_S_MAC_ADDR sets Ethernet interface MAC address. data argument is pointer to MAC address buffer with expected size of 6 bytes.
- ETH_CMD_G_MAC_ADDR gets Ethernet interface MAC address. data argument is pointer to a buffer to which MAC address is to be copied. The buffer size must be at least 6 bytes.
- ETH_CMD_S_PHY_ADDR sets PHY address in range of <0-31>. data argument is pointer to memory of uint32_t datatype from where the configuration option is read.
- ETH_CMD_G_PHY_ADDR gets PHY address. data argument is pointer to memory of uint32_t datatype to which the PHY address is to be stored.
- ETH_CMD_S_AUTONEGO enables or disables Ethernet link speed and duplex mode autonegotiation. data argument is pointer to memory of bool datatype from which the configuration option is read. Preconditions: Ethernet driver needs to be stopped.
- ETH_CMD_G_AUTONEGO gets current configuration of the Ethernet link speed and duplex mode autonegotiation. data argument is pointer to memory of bool datatype to which the current configuration is to be stored.
- ETH_CMD_S_SPEED sets the Ethernet link speed. data argument is pointer to memory of eth_speed_t datatype from which the configuration option is read. Preconditions: Ethernet driver needs to be stopped and auto-negotiation disabled.
- ETH_CMD_G_SPEED gets current Ethernet link speed. data argument is pointer to memory of eth_speed_t datatype to which the speed is to be stored.
- ETH_CMD_S_PROMISCUOUS sets/resets Ethernet interface promiscuous mode. data argument is pointer to memory of bool datatype from which the configuration option is read.
- ETH_CMD_S_FLOW_CTRL sets/resets Ethernet interface flow control. data argument is pointer to memory of bool datatype from which the configuration option is read.
- ETH_CMD_S_DUPLEX_MODE sets the Ethernet duplex mode. data argument is pointer to memory of eth_duplex_t datatype from which the configuration option is read. Preconditions: Ethernet driver needs to be stopped and auto-negotiation disabled.
- ETH_CMD_G_DUPLEX_MODE gets current Ethernet link duplex mode. data argument is pointer to memory of eth_duplex_t datatype to which the duplex mode is to be stored.
- ETH_CMD_S_PHY_LOOPBACK sets/resets PHY to/from loopback mode. data argument is pointer to memory of bool datatype from which the configuration option is read.

- Note that additional control commands may be available for specific MAC or PHY chips. Please consult specific MAC or PHY documentation or driver code.

参数

- **hdl** -- **[in]** handle of Ethernet driver
- **cmd** -- **[in]** IO control command
- **data** -- **[inout]** address of data for `set` command or address where to store the data when used with `get` command

返回

- ESP_OK: process io command successfully
- ESP_ERR_INVALID_ARG: process io command failed because of some invalid argument
- ESP_FAIL: process io command failed because some other error occurred
- ESP_ERR_NOT_SUPPORTED: requested feature is not supported

esp_err_t **esp_eth_get_phy_instance**(*esp_eth_handle_t* hdl, *esp_eth_phy_t* **phy)

Get PHY instance memory address.

参数

- **hdl** -- **[in]** handle of Ethernet driver
- **phy** -- **[out]** pointer to memory to store the instance

返回 esp_err_t

- ESP_OK: success
- ESP_ERR_INVALID_ARG: failed because of some invalid argument

esp_err_t **esp_eth_get_mac_instance**(*esp_eth_handle_t* hdl, *esp_eth_mac_t* **mac)

Get MAC instance memory address.

参数

- **hdl** -- **[in]** handle of Ethernet driver
- **mac** -- **[out]** pointer to memory to store the instance

返回 esp_err_t

- ESP_OK: success
- ESP_ERR_INVALID_ARG: failed because of some invalid argument

esp_err_t **esp_eth_increase_reference**(*esp_eth_handle_t* hdl)

Increase Ethernet driver reference.

备注: Ethernet driver handle can be obtained by os timer, netif, etc. It's dangerous when thread A is using Ethernet but thread B uninstall the driver. Using reference counter can prevent such risk, but care should be taken, when you obtain Ethernet driver, this API must be invoked so that the driver won't be uninstalled during your using time.

参数 **hdl** -- **[in]** handle of Ethernet driver

返回

- ESP_OK: increase reference successfully
- ESP_ERR_INVALID_ARG: increase reference failed because of some invalid argument

esp_err_t **esp_eth_decrease_reference**(*esp_eth_handle_t* hdl)

Decrease Ethernet driver reference.

参数 **hdl** -- **[in]** handle of Ethernet driver

返回

- ESP_OK: increase reference successfully
- ESP_ERR_INVALID_ARG: increase reference failed because of some invalid argument

Structures

struct **esp_eth_config_t**

Configuration of Ethernet driver.

Public Members

esp_eth_mac_t ***mac**

Ethernet MAC object.

esp_eth_phy_t ***phy**

Ethernet PHY object.

uint32_t **check_link_period_ms**

Period time of checking Ethernet link status.

esp_err_t (***stack_input**)(*esp_eth_handle_t* eth_handle, uint8_t *buffer, uint32_t length, void *priv)

Input frame buffer to user's stack.

Param eth_handle **[in]** handle of Ethernet driver
Param buffer **[in]** frame buffer that will get input to upper stack
Param length **[in]** length of the frame buffer
Return

- ESP_OK: input frame buffer to upper stack successfully
- ESP_FAIL: error occurred when inputting buffer to upper stack

esp_err_t (***on_lowlevel_init_done**)(*esp_eth_handle_t* eth_handle)

Callback function invoked when lowlevel initialization is finished.

Param eth_handle **[in]** handle of Ethernet driver
Return

- ESP_OK: process extra lowlevel initialization successfully
- ESP_FAIL: error occurred when processing extra lowlevel initialization

esp_err_t (***on_lowlevel_deinit_done**)(*esp_eth_handle_t* eth_handle)

Callback function invoked when lowlevel deinitialization is finished.

Param eth_handle **[in]** handle of Ethernet driver
Return

- ESP_OK: process extra lowlevel deinitialization successfully
- ESP_FAIL: error occurred when processing extra lowlevel deinitialization

esp_err_t (***read_phy_reg**)(*esp_eth_handle_t* eth_handle, uint32_t phy_addr, uint32_t phy_reg, uint32_t *reg_value)

Read PHY register.

备注: Usually the PHY register read/write function is provided by MAC (SMI interface), but if the PHY device is managed by other interface (e.g. I2C), then user needs to implement the corresponding read/write. Setting this to NULL means your PHY device is managed by MAC's SMI interface.

Param eth_handle **[in]** handle of Ethernet driver
Param phy_addr **[in]** PHY chip address (0~31)
Param phy_reg **[in]** PHY register index code

Param reg_value **[out]** PHY register value
Return
- ESP_OK: read PHY register successfully
- ESP_ERR_INVALID_ARG: read PHY register failed because of invalid argument
- ESP_ERR_TIMEOUT: read PHY register failed because of timeout
- ESP_FAIL: read PHY register failed because some other error occurred

esp_err_t (***write_phy_reg**)(*esp_eth_handle_t* eth_handle, uint32_t phy_addr, uint32_t phy_reg, uint32_t reg_value)

Write PHY register.

备注: Usually the PHY register read/write function is provided by MAC (SMI interface), but if the PHY device is managed by other interface (e.g. I2C), then user needs to implement the corresponding read/write. Setting this to NULL means your PHY device is managed by MAC's SMI interface.

Param eth_handle **[in]** handle of Ethernet driver
Param phy_addr **[in]** PHY chip address (0~31)
Param phy_reg **[in]** PHY register index code
Param reg_value **[in]** PHY register value
Return
- ESP_OK: write PHY register successfully
- ESP_ERR_INVALID_ARG: read PHY register failed because of invalid argument
- ESP_ERR_TIMEOUT: write PHY register failed because of timeout
- ESP_FAIL: write PHY register failed because some other error occurred

struct **esp_eth_phy_reg_rw_data_t**

Data structure to Read/Write PHY register via ioctl API.

Public Members

uint32_t **reg_addr**

PHY register address

uint32_t ***reg_value_p**

Pointer to a memory where the register value is read/written

Macros

ETH_DEFAULT_CONFIG(emac, ephy)

Default configuration for Ethernet driver.

Type Definitions

typedef void ***esp_eth_handle_t**

Handle of Ethernet driver.

Enumerations

enum **esp_eth_io_cmd_t**

Command list for ioctl API.

Values:

enumerator **ETH_CMD_G_MAC_ADDR**

Get MAC address

enumerator **ETH_CMD_S_MAC_ADDR**

Set MAC address

enumerator **ETH_CMD_G_PHY_ADDR**

Get PHY address

enumerator **ETH_CMD_S_PHY_ADDR**

Set PHY address

enumerator **ETH_CMD_G_AUTONEGO**

Get PHY Auto Negotiation

enumerator **ETH_CMD_S_AUTONEGO**

Set PHY Auto Negotiation

enumerator **ETH_CMD_G_SPEED**

Get Speed

enumerator **ETH_CMD_S_SPEED**

Set Speed

enumerator **ETH_CMD_S_PROMISCUOUS**

Set promiscuous mode

enumerator **ETH_CMD_S_FLOW_CTRL**

Set flow control

enumerator **ETH_CMD_G_DUPLEX_MODE**

Get Duplex mode

enumerator **ETH_CMD_S_DUPLEX_MODE**

Set Duplex mode

enumerator **ETH_CMD_S_PHY_LOOPBACK**

Set PHY loopback

enumerator **ETH_CMD_READ_PHY_REG**

Read PHY register

enumerator **ETH_CMD_WRITE_PHY_REG**

Write PHY register

enumerator **ETH_CMD_CUSTOM_MAC_CMDS**

enumerator **ETH_CMD_CUSTOM_PHY_CMDS**

Header File

- components/esp_eth/include/esp_eth_com.h
- This header file can be included with:

```
#include "esp_eth_com.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Structures

struct **esp_eth_mediator_s**

Ethernet mediator.

Public Members

esp_err_t (***phy_reg_read**)(*esp_eth_mediator_t* *eth, uint32_t phy_addr, uint32_t phy_reg, uint32_t *reg_value)

Read PHY register.

Param eth **[in]** mediator of Ethernet driver
Param phy_addr **[in]** PHY Chip address (0~31)
Param phy_reg **[in]** PHY register index code
Param reg_value **[out]** PHY register value
Return
- ESP_OK: read PHY register successfully
- ESP_FAIL: read PHY register failed because some error occurred

esp_err_t (***phy_reg_write**)(*esp_eth_mediator_t* *eth, uint32_t phy_addr, uint32_t phy_reg, uint32_t reg_value)

Write PHY register.

Param eth **[in]** mediator of Ethernet driver
Param phy_addr **[in]** PHY Chip address (0~31)
Param phy_reg **[in]** PHY register index code
Param reg_value **[in]** PHY register value
Return
- ESP_OK: write PHY register successfully
- ESP_FAIL: write PHY register failed because some error occurred

esp_err_t (***stack_input**)(*esp_eth_mediator_t* *eth, uint8_t *buffer, uint32_t length)

Deliver packet to upper stack.

Param eth **[in]** mediator of Ethernet driver
Param buffer **[in]** packet buffer
Param length **[in]** length of the packet
Return
- ESP_OK: deliver packet to upper stack successfully
- ESP_FAIL: deliver packet failed because some error occurred

esp_err_t (***on_state_changed**)(*esp_eth_mediator_t* *eth, *esp_eth_state_t* state, void *args)

Callback on Ethernet state changed.

Param eth **[in]** mediator of Ethernet driver
Param state **[in]** new state
Param args **[in]** optional argument for the new state
Return

- ESP_OK: process the new state successfully
- ESP_FAIL: process the new state failed because some error occurred

Macros

ETH_CMD_CUSTOM_MAC_CMDS_OFFSET

Offset for start of MAC custom ioctl commands.

ETH_CMD_CUSTOM_PHY_CMDS_OFFSET

Offset for start of PHY custom ioctl commands.

Type Definitions

typedef struct *esp_eth_mediator_s* **esp_eth_mediator_t**

Ethernet mediator.

Enumerations

enum **esp_eth_state_t**

Ethernet driver state.

Values:

enumerator **ETH_STATE_LLINIT**

Lowlevel init done

enumerator **ETH_STATE_DEINIT**

Deinit done

enumerator **ETH_STATE_LINK**

Link status changed

enumerator **ETH_STATE_SPEED**

Speed updated

enumerator **ETH_STATE_DUPLEX**

Duplex updated

enumerator **ETH_STATE_PAUSE**

Pause ability updated

enum **eth_event_t**

Ethernet event declarations.

Values:

enumerator **ETHERNET_EVENT_START**

Ethernet driver start

enumerator **ETHERNET_EVENT_STOP**

Ethernet driver stop

enumerator **ETHERNET_EVENT_CONNECTED**

Ethernet got a valid link

enumerator **ETHERNET_EVENT_DISCONNECTED**

Ethernet lost a valid link

Header File

- components/esp_eth/include/esp_eth_mac.h
- This header file can be included with:

```
#include "esp_eth_mac.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Structures

struct **esp_eth_mac_s**

Ethernet MAC.

Public Members

esp_err_t (***set_mediator**)(*esp_eth_mac_t* *mac, *esp_eth_mediator_t* *eth)

Set mediator for Ethernet MAC.

Param mac **[in]** Ethernet MAC instance
Param eth **[in]** Ethernet mediator
Return
- ESP_OK: set mediator for Ethernet MAC successfully
- ESP_ERR_INVALID_ARG: set mediator for Ethernet MAC failed because of invalid argument

esp_err_t (***init**)(*esp_eth_mac_t* *mac)

Initialize Ethernet MAC.

Param mac **[in]** Ethernet MAC instance
Return
- ESP_OK: initialize Ethernet MAC successfully
- ESP_ERR_TIMEOUT: initialize Ethernet MAC failed because of timeout
- ESP_FAIL: initialize Ethernet MAC failed because some other error occurred

esp_err_t (*`deinit`)(*esp_eth_mac_t* *mac)

Deinitialize Ethernet MAC.

Param mac **[in]** Ethernet MAC instance
Return
- ESP_OK: deinitialize Ethernet MAC successfully
- ESP_FAIL: deinitialize Ethernet MAC failed because some error occurred

esp_err_t (*`start`)(*esp_eth_mac_t* *mac)

Start Ethernet MAC.

Param mac **[in]** Ethernet MAC instance
Return
- ESP_OK: start Ethernet MAC successfully
- ESP_FAIL: start Ethernet MAC failed because some other error occurred

esp_err_t (*`stop`)(*esp_eth_mac_t* *mac)

Stop Ethernet MAC.

Param mac **[in]** Ethernet MAC instance
Return
- ESP_OK: stop Ethernet MAC successfully
- ESP_FAIL: stop Ethernet MAC failed because some error occurred

esp_err_t (*`transmit`)(*esp_eth_mac_t* *mac, uint8_t *buf, uint32_t length)

Transmit packet from Ethernet MAC.

备注: Returned error codes may differ for each specific MAC chip.

Param mac **[in]** Ethernet MAC instance
Param buf **[in]** packet buffer to transmit
Param length **[in]** length of packet
Return
- ESP_OK: transmit packet successfully
- ESP_ERR_INVALID_SIZE: number of actually sent bytes differs to expected
- ESP_FAIL: transmit packet failed because some other error occurred

esp_err_t (*`transmit_vargs`)(*esp_eth_mac_t* *mac, uint32_t argc, va_list args)

Transmit packet from Ethernet MAC constructed with special parameters at Layer2.

备注: Typical intended use case is to make possible to construct a frame from multiple higher layer buffers without a need of buffer reallocations. However, other use cases are not limited.

备注: Returned error codes may differ for each specific MAC chip.

Param mac **[in]** Ethernet MAC instance
Param argc **[in]** number variable arguments
Param args **[in]** variable arguments
Return
- ESP_OK: transmit packet successfully
- ESP_ERR_INVALID_SIZE: number of actually sent bytes differs to expected
- ESP_FAIL: transmit packet failed because some other error occurred

esp_err_t (*`receive`)(*esp_eth_mac_t* *mac, uint8_t *buf, uint32_t *length)

Receive packet from Ethernet MAC.

备注： Memory of buf is allocated in the Layer2, make sure it get free after process.

备注： Before this function got invoked, the value of ”length” should set by user, equals the size of buffer. After the function returned, the value of ”length” means the real length of received data.

Param mac **[in]** Ethernet MAC instance
Param buf **[out]** packet buffer which will preserve the received frame
Param length **[out]** length of the received packet
Return

- ESP_OK: receive packet successfully
- ESP_ERR_INVALID_ARG: receive packet failed because of invalid argument
- ESP_ERR_INVALID_SIZE: input buffer size is not enough to hold the incoming data. in this case, value of returned ”length” indicates the real size of incoming data.
- ESP_FAIL: receive packet failed because some other error occurred

esp_err_t (*`read_phy_reg`)(*esp_eth_mac_t* *mac, uint32_t phy_addr, uint32_t phy_reg, uint32_t *reg_value)

Read PHY register.

Param mac **[in]** Ethernet MAC instance
Param phy_addr **[in]** PHY chip address (0~31)
Param phy_reg **[in]** PHY register index code
Param reg_value **[out]** PHY register value
Return

- ESP_OK: read PHY register successfully
- ESP_ERR_INVALID_ARG: read PHY register failed because of invalid argument
- ESP_ERR_INVALID_STATE: read PHY register failed because of wrong state of MAC
- ESP_ERR_TIMEOUT: read PHY register failed because of timeout
- ESP_FAIL: read PHY register failed because some other error occurred

esp_err_t (*`write_phy_reg`)(*esp_eth_mac_t* *mac, uint32_t phy_addr, uint32_t phy_reg, uint32_t reg_value)

Write PHY register.

Param mac **[in]** Ethernet MAC instance
Param phy_addr **[in]** PHY chip address (0~31)
Param phy_reg **[in]** PHY register index code
Param reg_value **[in]** PHY register value
Return

- ESP_OK: write PHY register successfully
- ESP_ERR_INVALID_STATE: write PHY register failed because of wrong state of MAC
- ESP_ERR_TIMEOUT: write PHY register failed because of timeout
- ESP_FAIL: write PHY register failed because some other error occurred

esp_err_t (*`set_addr`)(*esp_eth_mac_t* *mac, uint8_t *addr)

Set MAC address.

Param mac **[in]** Ethernet MAC instance
Param addr **[in]** MAC address

Return
- ESP_OK: set MAC address successfully
- ESP_ERR_INVALID_ARG: set MAC address failed because of invalid argument
- ESP_FAIL: set MAC address failed because some other error occurred

esp_err_t (*`get_addr`)(*esp_eth_mac_t* *mac, uint8_t *addr)

Get MAC address.

Param mac **[in]** Ethernet MAC instance
Param addr **[out]** MAC address
Return
- ESP_OK: get MAC address successfully
- ESP_ERR_INVALID_ARG: get MAC address failed because of invalid argument
- ESP_FAIL: get MAC address failed because some other error occurred

esp_err_t (*`set_speed`)(*esp_eth_mac_t* *mac, *eth_speed_t* speed)

Set speed of MAC.

Param ma:c **[in]** Ethernet MAC instance
Param speed **[in]** MAC speed
Return
- ESP_OK: set MAC speed successfully
- ESP_ERR_INVALID_ARG: set MAC speed failed because of invalid argument
- ESP_FAIL: set MAC speed failed because some other error occurred

esp_err_t (*`set_duplex`)(*esp_eth_mac_t* *mac, *eth_duplex_t* duplex)

Set duplex mode of MAC.

Param mac **[in]** Ethernet MAC instance
Param duplex **[in]** MAC duplex
Return
- ESP_OK: set MAC duplex mode successfully
- ESP_ERR_INVALID_ARG: set MAC duplex failed because of invalid argument
- ESP_FAIL: set MAC duplex failed because some other error occurred

esp_err_t (*`set_link`)(*esp_eth_mac_t* *mac, *eth_link_t* link)

Set link status of MAC.

Param mac **[in]** Ethernet MAC instance
Param link **[in]** Link status
Return
- ESP_OK: set link status successfully
- ESP_ERR_INVALID_ARG: set link status failed because of invalid argument
- ESP_FAIL: set link status failed because some other error occurred

esp_err_t (*`set_promiscuous`)(*esp_eth_mac_t* *mac, bool enable)

Set promiscuous of MAC.

Param mac **[in]** Ethernet MAC instance
Param enable **[in]** set true to enable promiscuous mode; set false to disable promiscuous mode
Return
- ESP_OK: set promiscuous mode successfully
- ESP_FAIL: set promiscuous mode failed because some error occurred

esp_err_t (*`enable_flow_ctrl`)(*esp_eth_mac_t* *mac, bool enable)

Enable flow control on MAC layer or not.

Param mac **[in]** Ethernet MAC instance

Param enable **[in]** set true to enable flow control; set false to disable flow control
Return
- ESP_OK: set flow control successfully
- ESP_FAIL: set flow control failed because some error occurred

esp_err_t (***set_peer_pause_ability**)(*esp_eth_mac_t* *mac, uint32_t ability)

Set the PAUSE ability of peer node.

Param mac **[in]** Ethernet MAC instance
Param ability **[in]** zero indicates that pause function is supported by link partner; non-zero indicates that pause function is not supported by link partner
Return
- ESP_OK: set peer pause ability successfully
- ESP_FAIL: set peer pause ability failed because some error occurred

esp_err_t (***custom_ioctl**)(*esp_eth_mac_t* *mac, int cmd, void *data)

Custom IO function of MAC driver. This function is intended to extend common options of esp_eth_ioctl to cover specifics of MAC chip.

备注: This function may not be assigned when the MAC chip supports only most common set of configuration options.

Param mac **[in]** Ethernet MAC instance
Param cmd **[in]** IO control command
Param data **[inout]** address of data for `set` command or address where to store the data when used with `get` command
Return
- ESP_OK: process io command successfully
- ESP_ERR_INVALID_ARG: process io command failed because of some invalid argument
- ESP_FAIL: process io command failed because some other error occurred
- ESP_ERR_NOT_SUPPORTED: requested feature is not supported

esp_err_t (***del**)(*esp_eth_mac_t* *mac)

Free memory of Ethernet MAC.

Param mac **[in]** Ethernet MAC instance
Return
- ESP_OK: free Ethernet MAC instance successfully
- ESP_FAIL: free Ethernet MAC instance failed because some error occurred

struct **eth_mac_config_t**

Configuration of Ethernet MAC object.

Public Members

uint32_t **sw_reset_timeout_ms**

Software reset timeout value (Unit: ms)

uint32_t **rx_task_stack_size**

Stack size of the receive task

uint32_t **rx_task_prio**

Priority of the receive task

uint32_t **flags**

Flags that specify extra capability for mac driver

Macros

ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE

MAC driver can work when cache is disabled

ETH_MAC_FLAG_PIN_TO_CORE

Pin MAC task to the CPU core where driver installation happened

ETH_MAC_DEFAULT_CONFIG()

Default configuration for Ethernet MAC object.

Type Definitions

typedef struct *esp_eth_mac_s* **esp_eth_mac_t**

Ethernet MAC.

Header File

- components/esp_eth/include/esp_eth_mac_esp.h
- This header file can be included with:

```
#include "esp_eth_mac_esp.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Functions

esp_eth_mac_t ***esp_eth_mac_new_esp32**(const *eth_esp32_emac_config_t* *esp32_config, const *eth_mac_config_t* *config)

Create ESP32 Ethernet MAC instance.

参数

- **esp32_config** -- EMAC specific configuration
- **config** -- Ethernet MAC configuration

返回

- instance: create MAC instance successfully
- NULL: create MAC instance failed because some error occurred

Unions

union **eth_mac_clock_config_t**

#include <esp_eth_mac_esp.h> Ethernet MAC Clock Configuration.

Public Members

struct *eth_mac_clock_config_t*::[anonymous] **mii**

EMAC MII Clock Configuration

emac_rmii_clock_mode_t **clock_mode**

RMII Clock Mode Configuration

emac_rmii_clock_gpio_t **clock_gpio**

RMII Clock GPIO Configuration

struct *eth_mac_clock_config_t*::[anonymous] **rmii**

EMAC RMII Clock Configuration

union **eth_mac_dataif_gpio_config_t**

#include <esp_eth_mac_esp.h> Ethernet MAC MII/RMII data plane GPIO configuration.

Public Members

eth_mac_mii_gpio_config_t **mii**

EMAC MII Data GPIO Configuration

eth_mac_rmii_gpio_config_t **rmii**

EMAC RMII Data GPIO Configuration

Structures

struct **emac_esp_smi_gpio_config_t**

EMAC SMI GPIO configuration.

Public Members

int **mdc_num**

SMI MDC GPIO number, set to -1 could bypass the SMI GPIO configuration

int **mdio_num**

SMI MDIO GPIO number, set to -1 could bypass the SMI GPIO configuration

struct **eth_mac_mii_gpio_config_t**

EMAC MII data interface GPIO configuration.

Public Members

int **tx_clk_num**

TX_CLK GPIO number

int **tx_en_num**

TX_EN GPIO number

int **txd0_num**
TXD0 GPIO number

int **txd1_num**
TXD1 GPIO number

int **txd2_num**
TXD2 GPIO number

int **txd3_num**
TXD3 GPIO number

int **rx_clk_num**
RX_CLK GPIO number

int **rx_dv_num**
RX_DV GPIO number

int **rxd0_num**
RXD0 GPIO number

int **rxd1_num**
RXD1 GPIO number

int **rxd2_num**
RXD2 GPIO number

int **rxd3_num**
RXD3 GPIO number

int **col_in_num**
COL_IN GPIO number

int **crs_in_num**
CRS_IN GPIO number

int **tx_er_num**
TX_ER GPIO number

int **rx_er_num**
RX_ER GPIO number

struct **eth_mac_rmii_gpio_config_t**
EMAC RMII data interface GPIO configuration.

Public Members

int **tx_en_num**

TX_EN GPIO number

int **txd0_num**

TXD0 GPIO number

int **txd1_num**

TXD1 GPIO number

int **crs_dv_num**

CRS_DV GPIO number

int **rxd0_num**

RXD0 GPIO number

int **rxd1_num**

RXD1 GPIO number

struct **eth_esp32_emac_config_t**

EMAC specific configuration.

Public Members

emac_esp_smi_gpio_config_t **smi_gpio**

SMI GPIO numbers

int **smi_mdc_gpio_num**

SMI MDC GPIO number, set to -1 could bypass the SMI GPIO configuration

int **smi_mdio_gpio_num**

SMI MDIO GPIO number, set to -1 could bypass the SMI GPIO configuration

eth_data_interface_t **interface**

EMAC Data interface to PHY (MII/RMII)

eth_mac_clock_config_t **clock_config**

EMAC Interface clock configuration

eth_mac_dma_burst_len_t **dma_burst_len**

EMAC DMA burst length for both Tx and Rx

int **intr_priority**

EMAC interrupt priority, if set to 0 or a negative value, the driver will try to allocate an interrupt with a default priority

eth_mac_dataif_gpio_config_t **emac_dataif_gpio**

EMAC MII/RMII data plane GPIO configuration

eth_mac_clock_config_t **clock_config_out_in**

EMAC input clock configuration for internally generated output clock (when output clock is looped back externally)

Macros

ETH_ESP32_EMAC_DEFAULT_CONFIG()

Default ESP32's EMAC specific configuration.

Type Definitions

typedef int **emac_rmii_clock_gpio_t**

RMII Clock GPIO number.

Enumerations

enum **emac_rmii_clock_mode_t**

RMII Clock Mode Options.

Values:

enumerator **EMAC_CLK_DEFAULT**

Default values configured using Kconfig are going to be used when "Default" selected.

> **警告:** Deprecated option. Clock configuration using Kconfig is limitedly supported only for ESP32 SoC via `ETH_ESP32_EMAC_DEFAULT_CONFIG` and is going to be reevaluated in the next major release. Clock mode and clock GPIO number is supposed to be defined in `EMAC specific configuration` structure from user's code.

enumerator **EMAC_CLK_EXT_IN**

Input RMII Clock from external. EMAC Clock GPIO number needs to be configured when this option is selected.

> **备注:** MAC will get RMII clock from outside. Note that ESP32 only supports GPIO0 to input the RMII clock.

enumerator **EMAC_CLK_OUT**

Output RMII Clock from internal (A/M)PLL Clock. EMAC Clock GPIO number needs to be configured when this option is selected.

enum **eth_mac_esp_io_cmd_t**

List of ESP EMAC specific commands for ioctl API.

Values:

enumerator **ETH_MAC_ESP_CMD_SET_TDES0_CFG_BITS**

Set Transmit Descriptor Word 0 control bit mask (debug option)

enumerator **ETH_MAC_ESP_CMD_CLEAR_TDES0_CFG_BITS**

Clear Transmit Descriptor Word 0 control bit mask (debug option)

enumerator **ETH_MAC_ESP_CMD_PTP_ENABLE**

Enable IEEE1588 Time stamping

Header File

- components/esp_eth/include/esp_eth_mac_spi.h
- This header file can be included with:

```
#include "esp_eth_mac_spi.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Structures

struct **eth_spi_custom_driver_config_t**

Custom SPI Driver Configuration. This structure declares configuration and callback functions to access Ethernet SPI module via user's custom SPI driver.

Public Members

void ***config**

Custom driver specific configuration data used by *init()* function.

备注: Type and its content is fully under user's control

void *(***init**)(const void *spi_config)

Custom driver SPI Initialization.

备注: return type and its content is fully under user's control

Param spi_config **[in]** Custom driver specific configuration
Return

- spi_ctx: when initialization is successful, a pointer to context structure holding all variables needed for subsequent SPI access operations (e.g. SPI bus identification, mutexes, etc.)
- NULL: driver initialization failed

esp_err_t (***deinit**)(void *spi_ctx)

Custom driver De-initialization.

Param spi_ctx **[in]** a pointer to driver specific context structure
Return

- ESP_OK: driver de-initialization was successful
- ESP_FAIL: driver de-initialization failed
- any other failure codes are allowed to be used to provide failure isolation

esp_err_t (***read**)(void *spi_ctx, uint32_t cmd, uint32_t addr, void *data, uint32_t data_len)

Custom driver SPI read.

备注： The read function is responsible to construct command, address and data fields of the SPI frame in format expected by particular SPI Ethernet module

---

Param spi_ctx **[in]** a pointer to driver specific context structure
Param cmd **[in]** command
Param addr **[in]** register address
Param data **[out]** read data
Param data_len **[in]** read data length in bytes
Return

- ESP_OK: read was successful
- ESP_FAIL: read failed
- any other failure codes are allowed to be used to provide failure isolation

esp_err_t (***write**)(void *spi_ctx, uint32_t cmd, uint32_t addr, const void *data, uint32_t data_len)

Custom driver SPI write.

备注： The write function is responsible to construct command, address and data fields of the SPI frame in format expected by particular SPI Ethernet module

---

Param spi_ctx **[in]** a pointer to driver specific context structure
Param cmd **[in]** command
Param addr **[in]** register address
Param data **[in]** data to write
Param data_len **[in]** length of data to write in bytes
Return

- ESP_OK: write was successful
- ESP_FAIL: write failed
- any other failure codes are allowed to be used to provide failure isolation

Macros

ETH_DEFAULT_SPI

Default configuration of the custom SPI driver. Internal ESP-IDF SPI Master driver is used by default.

Header File

- components/esp_eth/include/esp_eth_phy.h
- This header file can be included with:

```
#include "esp_eth_phy.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Functions

esp_eth_phy_t ***esp_eth_phy_new_ip101*** (const *eth_phy_config_t* *config)

Create a PHY instance of IP101.

参数 **config** -- **[in]** configuration of PHY

返回

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

esp_eth_phy_t ***esp_eth_phy_new_rtl8201*** (const *eth_phy_config_t* *config)

Create a PHY instance of RTL8201.

参数 **config** -- **[in]** configuration of PHY

返回

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

esp_eth_phy_t ***esp_eth_phy_new_lan87xx*** (const *eth_phy_config_t* *config)

Create a PHY instance of LAN87xx.

参数 **config** -- **[in]** configuration of PHY

返回

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

esp_eth_phy_t ***esp_eth_phy_new_dp83848*** (const *eth_phy_config_t* *config)

Create a PHY instance of DP83848.

参数 **config** -- **[in]** configuration of PHY

返回

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

esp_eth_phy_t ***esp_eth_phy_new_ksz80xx*** (const *eth_phy_config_t* *config)

Create a PHY instance of KSZ80xx.

The phy model from the KSZ80xx series is detected automatically. If the driver is unable to detect a supported model, `NULL` is returned.

Currently, the following models are supported: KSZ8001, KSZ8021, KSZ8031, KSZ8041, KSZ8051, KSZ8061, KSZ8081, KSZ8091

参数 **config** -- **[in]** configuration of PHY

返回

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

Structures

struct ***esp_eth_phy_s***

Ethernet PHY.

Public Members

esp_err_t (***set_mediator**)(*esp_eth_phy_t* *phy, *esp_eth_mediator_t* *mediator)

Set mediator for PHY.

Param phy **[in]** Ethernet PHY instance
Param mediator **[in]** mediator of Ethernet driver
Return

- ESP_OK: set mediator for Ethernet PHY instance successfully

- ESP_ERR_INVALID_ARG: set mediator for Ethernet PHY instance failed because of some invalid arguments

esp_err_t (*`reset`)(*esp_eth_phy_t* *phy)

Software Reset Ethernet PHY.

Param phy **[in]** Ethernet PHY instance
Return
- ESP_OK: reset Ethernet PHY successfully
- ESP_FAIL: reset Ethernet PHY failed because some error occurred

esp_err_t (*`reset_hw`)(*esp_eth_phy_t* *phy)

Hardware Reset Ethernet PHY.

备注： Hardware reset is mostly done by pull down and up PHY's nRST pin

Param phy **[in]** Ethernet PHY instance
Return
- ESP_OK: reset Ethernet PHY successfully
- ESP_FAIL: reset Ethernet PHY failed because some error occurred

esp_err_t (*`init`)(*esp_eth_phy_t* *phy)

Initialize Ethernet PHY.

Param phy **[in]** Ethernet PHY instance
Return
- ESP_OK: initialize Ethernet PHY successfully
- ESP_FAIL: initialize Ethernet PHY failed because some error occurred

esp_err_t (*`deinit`)(*esp_eth_phy_t* *phy)

Deinitialize Ethernet PHY.

Param phy **[in]** Ethernet PHY instance
Return
- ESP_OK: deinitialize Ethernet PHY successfully
- ESP_FAIL: deinitialize Ethernet PHY failed because some error occurred

esp_err_t (*`autonego_ctrl`)(*esp_eth_phy_t* *phy, *eth_phy_autoneg_cmd_t* cmd, bool *autonego_en_stat)

Configure auto negotiation.

Param phy **[in]** Ethernet PHY instance
Param cmd **[in]** Configuration command, it is possible to Enable (restart), Disable or get current status of PHY auto negotiation
Param autonego_en_stat **[out]** Address where to store current status of auto negotiation configuration
Return
- ESP_OK: restart auto negotiation successfully
- ESP_FAIL: restart auto negotiation failed because some error occurred
- ESP_ERR_INVALID_ARG: invalid command

esp_err_t (*`get_link`)(*esp_eth_phy_t* *phy)

Get Ethernet PHY link status.

Param phy **[in]** Ethernet PHY instance
Return

- ESP_OK: get Ethernet PHY link status successfully
- ESP_FAIL: get Ethernet PHY link status failed because some error occurred

esp_err_t (*`set_link`)(*esp_eth_phy_t* *phy, *eth_link_t* link)

Set Ethernet PHY link status.

Param phy **[in]** Ethernet PHY instance
Param link **[in]** new link status
Return
- ESP_OK: set Ethernet PHY link status successfully
- ESP_FAIL: set Ethernet PHY link status failed because some error occurred

esp_err_t (*`pwrctl`)(*esp_eth_phy_t* *phy, bool enable)

Power control of Ethernet PHY.

Param phy **[in]** Ethernet PHY instance
Param enable **[in]** set true to power on Ethernet PHY; ser false to power off Ethernet PHY
Return
- ESP_OK: control Ethernet PHY power successfully
- ESP_FAIL: control Ethernet PHY power failed because some error occurred

esp_err_t (*`set_addr`)(*esp_eth_phy_t* *phy, uint32_t addr)

Set PHY chip address.

Param phy **[in]** Ethernet PHY instance
Param addr **[in]** PHY chip address
Return
- ESP_OK: set Ethernet PHY address successfully
- ESP_FAIL: set Ethernet PHY address failed because some error occurred

esp_err_t (*`get_addr`)(*esp_eth_phy_t* *phy, uint32_t *addr)

Get PHY chip address.

Param phy **[in]** Ethernet PHY instance
Param addr **[out]** PHY chip address
Return
- ESP_OK: get Ethernet PHY address successfully
- ESP_ERR_INVALID_ARG: get Ethernet PHY address failed because of invalid argument

esp_err_t (*`advertise_pause_ability`)(*esp_eth_phy_t* *phy, uint32_t ability)

Advertise pause function supported by MAC layer.

Param phy **[in]** Ethernet PHY instance
Param addr **[out]** Pause ability
Return
- ESP_OK: Advertise pause ability successfully
- ESP_ERR_INVALID_ARG: Advertise pause ability failed because of invalid argument

esp_err_t (*`loopback`)(*esp_eth_phy_t* *phy, bool enable)

Sets the PHY to loopback mode.

Param phy **[in]** Ethernet PHY instance
Param enable **[in]** enables or disables PHY loopback
Return
- ESP_OK: PHY instance loopback mode has been configured successfully
- ESP_FAIL: PHY instance loopback configuration failed because some error occurred

esp_err_t (*`set_speed`)(*esp_eth_phy_t* *phy, *eth_speed_t* speed)

Sets PHY speed mode.

备注： Autonegotiation feature needs to be disabled prior to calling this function for the new setting to be applied

---

Param phy [in] Ethernet PHY instance
Param speed [in] Speed mode to be set
Return
- ESP_OK: PHY instance speed mode has been configured successfully
- ESP_FAIL: PHY instance speed mode configuration failed because some error occurred

esp_err_t (*`set_duplex`)(*esp_eth_phy_t* *phy, *eth_duplex_t* duplex)

Sets PHY duplex mode.

备注： Autonegotiation feature needs to be disabled prior to calling this function for the new setting to be applied

---

Param phy [in] Ethernet PHY instance
Param duplex [in] Duplex mode to be set
Return
- ESP_OK: PHY instance duplex mode has been configured successfully
- ESP_FAIL: PHY instance duplex mode configuration failed because some error occurred

esp_err_t (*`custom_ioctl`)(*esp_eth_phy_t* *phy, int cmd, void *data)

Custom IO function of PHY driver. This function is intended to extend common options of esp_eth_ioctl to cover specifics of PHY chip.

备注： This function may not be assigned when the PHY chip supports only most common set of configuration options.

Param phy [in] Ethernet PHY instance
Param cmd [in] IO control command
Param data [inout] address of data for `set` command or address where to store the data when used with `get` command
Return
- ESP_OK: process io command successfully
- ESP_ERR_INVALID_ARG: process io command failed because of some invalid argument
- ESP_FAIL: process io command failed because some other error occurred
- ESP_ERR_NOT_SUPPORTED: requested feature is not supported

esp_err_t (*`del`)(*esp_eth_phy_t* *phy)

Free memory of Ethernet PHY instance.

Param phy [in] Ethernet PHY instance
Return
- ESP_OK: free PHY instance successfully
- ESP_FAIL: free PHY instance failed because some error occurred

struct **eth_phy_config_t**

Ethernet PHY configuration.

Public Members

int32_t **phy_addr**

PHY address, set -1 to enable PHY address detection at initialization stage

uint32_t **reset_timeout_ms**

Reset timeout value (Unit: ms)

uint32_t **autonego_timeout_ms**

Auto-negotiation timeout value (Unit: ms)

int **reset_gpio_num**

Reset GPIO number, -1 means no hardware reset

Macros

ESP_ETH_PHY_ADDR_AUTO

ETH_PHY_DEFAULT_CONFIG( )

Default configuration for Ethernet PHY object.

Type Definitions

typedef struct *esp_eth_phy_s* **esp_eth_phy_t**

Ethernet PHY.

Enumerations

enum **eth_phy_autoneg_cmd_t**

Auto-negotiation control commands.

Values:

enumerator **ESP_ETH_PHY_AUTONEGO_RESTART**

enumerator **ESP_ETH_PHY_AUTONEGO_EN**

enumerator **ESP_ETH_PHY_AUTONEGO_DIS**

enumerator **ESP_ETH_PHY_AUTONEGO_G_STAT**

Header File

- components/esp_eth/include/esp_eth_phy_802_3.h
- This header file can be included with:

```
#include "esp_eth_phy_802_3.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Functions

esp_err_t **esp_eth_phy_802_3_set_mediator**(*phy_802_3_t* *phy_802_3, *esp_eth_mediator_t* *eth)

Set Ethernet mediator.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **eth** -- Ethernet mediator pointer

返回

- ESP_OK: Ethermet mediator set successfully
- ESP_ERR_INVALID_ARG: if `eth` is `NULL`

esp_err_t **esp_eth_phy_802_3_reset**(*phy_802_3_t* *phy_802_3)

Reset PHY.

参数 **phy_802_3** -- IEEE 802.3 PHY object infostructure

返回

- ESP_OK: Ethernet PHY reset successfully
- ESP_FAIL: reset Ethernet PHY failed because some error occurred

esp_err_t **esp_eth_phy_802_3_autonego_ctrl**(*phy_802_3_t* *phy_802_3, *eth_phy_autoneg_cmd_t* cmd, bool *autonego_en_stat)

Control autonegotiation mode of Ethernet PHY.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **cmd** -- autonegotiation command enumeration
- **autonego_en_stat** -- **[out]** autonegotiation enabled flag

返回

- ESP_OK: Ethernet PHY autonegotiation configured successfully
- ESP_FAIL: Ethernet PHY autonegotiation configuration fail because some error occurred
- ESP_ERR_INVALID_ARG: invalid value of `cmd`

esp_err_t **esp_eth_phy_802_3_pwrctl**(*phy_802_3_t* *phy_802_3, bool enable)

Power control of Ethernet PHY.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **enable** -- set true to power ON Ethernet PHY; set false to power OFF Ethernet PHY

返回

- ESP_OK: Ethernet PHY power down mode set successfully
- ESP_FAIL: Ethernet PHY power up or power down failed because some error occurred

esp_err_t **esp_eth_phy_802_3_set_addr**(*phy_802_3_t* *phy_802_3, uint32_t addr)

Set Ethernet PHY address.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **addr** -- new PHY address

返回

- ESP_OK: Ethernet PHY address set

esp_err_t **esp_eth_phy_802_3_get_addr**(*phy_802_3_t* *phy_802_3, uint32_t *addr)

Get Ethernet PHY address.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **addr** -- **[out]** Ethernet PHY address

返回

- ESP_OK: Ethernet PHY address read successfully
- ESP_ERR_INVALID_ARG: `addr` pointer is `NULL`

esp_err_t **esp_eth_phy_802_3_advertise_pause_ability**(*phy_802_3_t* *phy_802_3, uint32_t ability)

Advertise pause function ability.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **ability** -- enable or disable pause ability

返回

- ESP_OK: pause ability set successfully
- ESP_FAIL: Advertise pause function ability failed because some error occurred

esp_err_t **esp_eth_phy_802_3_loopback**(*phy_802_3_t* *phy_802_3, bool enable)

Set Ethernet PHY loopback mode.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **enable** -- set true to enable loopback; set false to disable loopback

返回

- ESP_OK: Ethernet PHY loopback mode set successfully
- ESP_FAIL: Ethernet PHY loopback configuration failed because some error occurred

esp_err_t **esp_eth_phy_802_3_set_speed**(*phy_802_3_t* *phy_802_3, *eth_speed_t* speed)

Set Ethernet PHY speed.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **speed** -- new speed of Ethernet PHY link

返回

- ESP_OK: Ethernet PHY speed set successfully
- ESP_FAIL: Set Ethernet PHY speed failed because some error occurred

esp_err_t **esp_eth_phy_802_3_set_duplex**(*phy_802_3_t* *phy_802_3, *eth_duplex_t* duplex)

Set Ethernet PHY duplex mode.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **duplex** -- new duplex mode for Ethernet PHY link

返回

- ESP_OK: Ethernet PHY duplex mode set successfully
- ESP_ERR_INVALID_STATE: unable to set duplex mode to Half if loopback is enabled
- ESP_FAIL: Set Ethernet PHY duplex mode failed because some error occurred

esp_err_t **esp_eth_phy_802_3_set_link**(*phy_802_3_t* *phy_802_3, *eth_link_t* link)

Set Ethernet PHY link status.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **link** -- new link status

返回

- ESP_OK: Ethernet PHY link set successfully

esp_err_t **esp_eth_phy_802_3_init** (*phy_802_3_t* *phy_802_3)

Initialize Ethernet PHY.

参数 **phy_802_3** -- IEEE 802.3 PHY object infostructure

返回

- ESP_OK: Ethernet PHY initialized successfully

esp_err_t **esp_eth_phy_802_3_deinit** (*phy_802_3_t* *phy_802_3)

Power off Eternet PHY.

参数 **phy_802_3** -- IEEE 802.3 PHY object infostructure

返回

- ESP_OK: Ethernet PHY powered off successfully

esp_err_t **esp_eth_phy_802_3_del** (*phy_802_3_t* *phy_802_3)

Delete Ethernet PHY infostructure.

参数 **phy_802_3** -- IEEE 802.3 PHY object infostructure

返回

- ESP_OK: Ethrnet PHY infostructure deleted

esp_err_t **esp_eth_phy_802_3_reset_hw** (*phy_802_3_t* *phy_802_3, uint32_t reset_assert_us)

Performs hardware reset with specific reset pin assertion time.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **reset_assert_us** -- Hardware reset pin assertion time

返回

- ESP_OK: reset Ethernet PHY successfully

esp_err_t **esp_eth_phy_802_3_detect_phy_addr** (*esp_eth_mediator_t* *eth, int *detected_addr)

Detect PHY address.

参数

- **eth** -- Mediator of Ethernet driver
- **detected_addr** -- **[out]** a valid address after detection

返回

- ESP_OK: detect phy address successfully
- ESP_ERR_INVALID_ARG: invalid parameter
- ESP_ERR_NOT_FOUND: can't detect any PHY device
- ESP_FAIL: detect phy address failed because some error occurred

esp_err_t **esp_eth_phy_802_3_basic_phy_init** (*phy_802_3_t* *phy_802_3)

Performs basic PHY chip initialization.

备注: It should be called as the first function in PHY specific driver instance

---

参数 **phy_802_3** -- IEEE 802.3 PHY object infostructure

返回

- ESP_OK: initialized Ethernet PHY successfully
- ESP_FAIL: initialization of Ethernet PHY failed because some error occurred
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NOT_FOUND: PHY device not detected
- ESP_ERR_TIMEOUT: MII Management read/write operation timeout
- ESP_ERR_INVALID_STATE: PHY is in invalid state to perform requested operation

esp_err_t **esp_eth_phy_802_3_basic_phy_deinit** (*phy_802_3_t* *phy_802_3)

Performs basic PHY chip de-initialization.

备注： It should be called as the last function in PHY specific driver instance

参数 **phy_802_3** -- IEEE 802.3 PHY object infostructure

返回

- ESP_OK: de-initialized Ethernet PHY successfully
- ESP_FAIL: de-initialization of Ethernet PHY failed because some error occurred
- ESP_ERR_TIMEOUT: MII Management read/write operation timeout
- ESP_ERR_INVALID_STATE: PHY is in invalid state to perform requested operation

esp_err_t **esp_eth_phy_802_3_read_oui**(*phy_802_3_t* *phy_802_3, uint32_t *oui)

Reads raw content of OUI field.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **oui** -- **[out]** OUI value

返回

- ESP_OK: OUI field read successfully
- ESP_FAIL: OUI field read failed because some error occurred
- ESP_ERR_INVALID_ARG: invalid `oui` argument
- ESP_ERR_TIMEOUT: MII Management read/write operation timeout
- ESP_ERR_INVALID_STATE: PHY is in invalid state to perform requested operation

esp_err_t **esp_eth_phy_802_3_read_manufac_info**(*phy_802_3_t* *phy_802_3, uint8_t *model, uint8_t *rev)

Reads manufacturer's model and revision number.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **model** -- **[out]** Manufacturer's model number (can be NULL when not required)
- **rev** -- **[out]** Manufacturer's revision number (can be NULL when not required)

返回

- ESP_OK: Manufacturer's info read successfully
- ESP_FAIL: Manufacturer's info read failed because some error occurred
- ESP_ERR_TIMEOUT: MII Management read/write operation timeout
- ESP_ERR_INVALID_STATE: PHY is in invalid state to perform requested operation

esp_err_t **esp_eth_phy_802_3_get_mmd_addr**(*phy_802_3_t* *phy_802_3, uint8_t devaddr, uint16_t *mmd_addr)

Reads MDIO device's internal address register.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **devaddr** -- Address of MDIO device
- **mmd_addr** -- **[out]** Current address stored in device's register

返回

- ESP_OK: Address register read successfully
- ESP_FAIL: Address register read failed because of some error occurred
- ESP_ERR_INVALID_ARG: Device address provided is out of range (hardware limits device address to 5 bits)

esp_err_t **esp_eth_phy_802_3_set_mmd_addr**(*phy_802_3_t* *phy_802_3, uint8_t devaddr, uint16_t mmd_addr)

Write to DIO device's internal address register.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **devaddr** -- Address of MDIO device
- **mmd_addr** -- **[out]** New value of MDIO device's address register value

返回

- ESP_OK: Address register written to successfully
- ESP_FAIL: Address register write failed because of some error occurred
- ESP_ERR_INVALID_ARG: Device address provided is out of range (hardware limits device address to 5 bits)

esp_err_t **esp_eth_phy_802_3_read_mmd_data**(*phy_802_3_t* *phy_802_3, uint8_t devaddr, *esp_eth_phy_802_3_mmd_func_t* function, uint32_t *data)

Read data of MDIO device's memory at address register.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **devaddr** -- Address of MDIO device
- **function** -- MMD function
- **data** -- **[out]** Data read from the device's memory

返回

- ESP_OK: Memory read successfully
- ESP_FAIL: Memory read failed because of some error occurred
- ESP_ERR_INVALID_ARG: Device address provided is out of range (hardware limits device address to 5 bits) or MMD access function is invalid

esp_err_t **esp_eth_phy_802_3_write_mmd_data**(*phy_802_3_t* *phy_802_3, uint8_t devaddr, *esp_eth_phy_802_3_mmd_func_t* function, uint32_t data)

Write data to MDIO device's memory at address register.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **devaddr** -- Address of MDIO device
- **function** -- MMD function
- **data** -- **[out]** Data to write to the device's memory

返回

- ESP_OK: Memory written successfully
- ESP_FAIL: Memory write failed because of some error occurred
- ESP_ERR_INVALID_ARG: Device address provided is out of range (hardware limits device address to 5 bits) or MMD access function is invalid

esp_err_t **esp_eth_phy_802_3_read_mmd_register**(*phy_802_3_t* *phy_802_3, uint8_t devaddr, uint16_t mmd_addr, uint32_t *data)

Set MMD address to mmd_addr with function MMD_FUNC_NOINCR and read contents to *data.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **devaddr** -- Address of MDIO device
- **mmd_addr** -- Address of MDIO device register
- **data** -- **[out]** Data read from the device's memory

返回

- ESP_OK: Memory read successfully
- ESP_FAIL: Memory read failed because of some error occurred
- ESP_ERR_INVALID_ARG: Device address provided is out of range (hardware limits device address to 5 bits)

esp_err_t **esp_eth_phy_802_3_write_mmd_register**(*phy_802_3_t* *phy_802_3, uint8_t devaddr, uint16_t mmd_addr, uint32_t data)

Set MMD address to mmd_addr with function MMD_FUNC_NOINCR and write data.

参数

- **phy_802_3** -- IEEE 802.3 PHY object infostructure
- **devaddr** -- Address of MDIO device
- **mmd_addr** -- Address of MDIO device register

- **data** -- **[out]** Data to write to the device's memory

返回

- ESP_OK: Memory written to successfully
- ESP_FAIL: Memory write failed because of some error occurred
- ESP_ERR_INVALID_ARG: Device address provided is out of range (hardware limits device address to 5 bits)

inline *phy_802_3_t* ***esp_eth_phy_into_phy_802_3**(*esp_eth_phy_t* *phy)

Returns address to parent IEEE 802.3 PHY object infostructure.

参数 **phy** -- Ethernet PHY instance

返回 phy_802_3_t*

- address to parent IEEE 802.3 PHY object infostructure

esp_err_t **esp_eth_phy_802_3_obj_config_init**(*phy_802_3_t* *phy_802_3, const *eth_phy_config_t* *config)

Initializes configuration of parent IEEE 802.3 PHY object infostructure.

参数

- **phy_802_3** -- Address to IEEE 802.3 PHY object infostructure
- **config** -- Configuration of the IEEE 802.3 PHY object

返回

- ESP_OK: configuration initialized successfully
- ESP_ERR_INVALID_ARG: invalid `config` argument

Structures

struct **phy_802_3_t**

IEEE 802.3 PHY object infostructure.

Public Members

esp_eth_phy_t **parent**

Parent Ethernet PHY instance

esp_eth_mediator_t ***eth**

Mediator of Ethernet driver

int **addr**

PHY address

uint32_t **reset_timeout_ms**

Reset timeout value (Unit: ms)

uint32_t **autonego_timeout_ms**

Auto-negotiation timeout value (Unit: ms)

eth_link_t **link_status**

Current Link status

int **reset_gpio_num**

Reset GPIO number, -1 means no hardware reset

Enumerations

enum **esp_eth_phy_802_3_mmd_func_t**

IEEE 802.3 MMD modes enumeration.

Values:

enumerator **MMD_FUNC_ADDRESS**

enumerator **MMD_FUNC_DATA_NOINCR**

enumerator **MMD_FUNC_DATA_RWINCR**

enumerator **MMD_FUNC_DATA_WINCR**

Header File

- components/esp_eth/include/esp_eth_netif_glue.h
- This header file can be included with:

```
#include "esp_eth_netif_glue.h"
```

- This header file is a part of the API provided by the `esp_eth` component. To declare that your component depends on `esp_eth`, add the following to your CMakeLists.txt:

```
REQUIRES esp_eth
```

or

```
PRIV_REQUIRES esp_eth
```

Functions

esp_eth_netif_glue_handle_t **esp_eth_new_netif_glue**(*esp_eth_handle_t* eth_hdl)

Create a netif glue for Ethernet driver.

备注： netif glue is used to attach io driver to TCP/IP netif

参数 **eth_hdl** -- Ethernet driver handle
返回 glue object, which inherits esp_netif_driver_base_t

esp_err_t **esp_eth_del_netif_glue**(*esp_eth_netif_glue_handle_t* eth_netif_glue)

Delete netif glue of Ethernet driver.

参数 **eth_netif_glue** -- netif glue
返回 -ESP_OK: delete netif glue successfully

Type Definitions

typedef struct esp_eth_netif_glue_t ***esp_eth_netif_glue_handle_t**

Handle of netif glue - an intermediate layer between netif and Ethernet driver.

本部分的以太网 API 示例代码存放在 ESP-IDF 示例项目的 ethernet 目录下。

2.4.2 Thread

Thread

概述 Thread 是一个基于 IP 的网状网络协议，它基于 802.15.4 物理层和 MAC 层。

应用示例 ESP-IDF 示例目录 openthread 包含以下应用程序：

- OpenThread 交互 shell：openthread/ot_cli
- 边界路由器 (Thread Border Router)：openthread/ot_br
- Thread 无线电协处理器 (Thread Radio Co-Processor)：openthread/ot_rcp

API 参考 应使用 OpenThread API 操作 Thread 网络。请参考 OpenThread API 文档。

ESP-IDF 提供额外的 API，用于启动和管理 OpenThread 实现执行网络接口绑定和边界路由功能。

Header File

- components/openthread/include/esp_openthread.h
- This header file can be included with:

```
#include "esp_openthread.h"
```

- This header file is a part of the API provided by the `openthread` component. To declare that your component depends on `openthread`, add the following to your CMakeLists.txt:

```
REQUIRES openthread
```

or

```
PRIV_REQUIRES openthread
```

Functions

esp_err_t **esp_openthread_init**(const *esp_openthread_platform_config_t* *init_config)

Initializes the full OpenThread stack.

备注： The OpenThread instance will also be initialized in this function.

参数 **init_config** -- **[in]** The initialization configuration.

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if allocation has failed
- ESP_ERR_INVALID_ARG if radio or host connection mode not supported
- ESP_ERR_INVALID_STATE if already initialized

esp_err_t **esp_openthread_auto_start**(otOperationalDatasetTlvs *datasetTlvs)

Starts the Thread protocol operation and attaches to a Thread network.

参数 **datasetTlvs** -- **[in]** The operational dataset (TLV encoded), if it's NULL, the function will generate the dataset based on the configurations from kconfig.

返回

- ESP_OK on success
- ESP_FAIL on failures

esp_err_t **esp_openthread_launch_mainloop**(void)

Launches the OpenThread main loop.

备注: This function will not return unless error happens when running the OpenThread stack.

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if allocation has failed
- ESP_FAIL on other failures

esp_err_t **esp_openthread_deinit**(void)

This function performs OpenThread stack and platform driver deinitialization.

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if not initialized

otInstance ***esp_openthread_get_instance**(void)

This function acquires the underlying OpenThread instance.

备注: This function can be called on other tasks without lock.

返回 The OpenThread instance pointer

Header File

- components/openthread/include/esp_openthread_types.h
- This header file can be included with:

```
#include "esp_openthread_types.h"
```

- This header file is a part of the API provided by the `openthread` component. To declare that your component depends on `openthread`, add the following to your CMakeLists.txt:

```
REQUIRES openthread
```

or

```
PRIV_REQUIRES openthread
```

Structures

struct **esp_openthread_role_changed_event_t**

OpenThread role changed event data.

Public Members

otDeviceRole **previous_role**

Previous Thread role

otDeviceRole **current_role**

Current Thread role

struct **esp_openthread_dataset_changed_event_t**

OpenThread dataset changed event data.

Public Members

esp_openthread_dataset_type_t **type**

Dataset type

otOperationalDataset **new_dataset**

New dataset

struct **esp_openthread_mainloop_context_t**

This structure represents a context for a select() based mainloop.

Public Members

fd_set **read_fds**

The read file descriptors

fd_set **write_fds**

The write file descriptors

fd_set **error_fds**

The error file descriptors

int **max_fd**

The max file descriptor

struct timeval **timeout**

The timeout

struct **esp_openthread_uart_config_t**

The uart port config for OpenThread.

Public Members

uart_port_t **port**

UART port number

uart_config_t **uart_config**

UART configuration, see *uart_config_t* docs

gpio_num_t **rx_pin**

UART RX pin

gpio_num_t **tx_pin**

UART TX pin

struct **esp_openthread_spi_host_config_t**

The spi port config for OpenThread.

Public Members

spi_host_device_t **host_device**

SPI host device

spi_dma_chan_t **dma_channel**

DMA channel

spi_bus_config_t **spi_interface**

SPI bus

spi_device_interface_config_t **spi_device**

SPI peripheral device

gpio_num_t **intr_pin**

SPI interrupt pin

struct **esp_openthread_spi_slave_config_t**

The spi slave config for OpenThread.

Public Members

spi_host_device_t **host_device**

SPI host device

spi_bus_config_t **bus_config**

SPI bus config

spi_slave_interface_config_t **slave_config**

SPI slave config

gpio_num_t **intr_pin**

SPI interrupt pin

struct **esp_openthread_radio_config_t**

The OpenThread radio configuration.

Public Members

esp_openthread_radio_mode_t **radio_mode**

The radio mode

esp_openthread_uart_config_t **radio_uart_config**

The uart configuration to RCP

esp_openthread_spi_host_config_t **radio_spi_config**

The spi configuration to RCP

struct **esp_openthread_host_connection_config_t**

The OpenThread host connection configuration.

Public Members

esp_openthread_host_connection_mode_t **host_connection_mode**

The host connection mode

esp_openthread_uart_config_t **host_uart_config**

The uart configuration to host

usb_serial_jtag_driver_config_t **host_usb_config**

The usb configuration to host

esp_openthread_spi_slave_config_t **spi_slave_config**

The spi configuration to host

struct **esp_openthread_port_config_t**

The OpenThread port specific configuration.

Public Members

const char ***storage_partition_name**

The partition for storing OpenThread dataset

uint8_t **netif_queue_size**

The packet queue size for the network interface

uint8_t **task_queue_size**

The task queue size

struct **esp_openthread_platform_config_t**

The OpenThread platform configuration.

Public Members

esp_openthread_radio_config_t **radio_config**

The radio configuration

esp_openthread_host_connection_config_t **host_config**

The host connection configuration

esp_openthread_port_config_t **port_config**

The port configuration

Type Definitions

typedef void (***esp_openthread_rcp_failure_handler**)(void)

typedef void (***esp_openthread_compatibility_error_callback**)(void)

Enumerations

enum **esp_openthread_event_t**

OpenThread event declarations.

Values:

enumerator **OPENTHREAD_EVENT_START**

OpenThread stack start

enumerator **OPENTHREAD_EVENT_STOP**

OpenThread stack stop

enumerator **OPENTHREAD_EVENT_DETACHED**

OpenThread detached

enumerator **OPENTHREAD_EVENT_ATTACHED**

OpenThread attached

enumerator **OPENTHREAD_EVENT_ROLE_CHANGED**

OpenThread role changed

enumerator **OPENTHREAD_EVENT_IF_UP**

OpenThread network interface up

enumerator **OPENTHREAD_EVENT_IF_DOWN**

OpenThread network interface down

enumerator **OPENTHREAD_EVENT_GOT_IP6**

OpenThread stack added IPv6 address

enumerator **OPENTHREAD_EVENT_LOST_IP6**

OpenThread stack removed IPv6 address

enumerator **OPENTHREAD_EVENT_MULTICAST_GROUP_JOIN**

OpenThread stack joined IPv6 multicast group

enumerator **OPENTHREAD_EVENT_MULTICAST_GROUP_LEAVE**

OpenThread stack left IPv6 multicast group

enumerator **OPENTHREAD_EVENT_TREL_ADD_IP6**

OpenThread stack added TREL IPv6 address

enumerator **OPENTHREAD_EVENT_TREL_REMOVE_IP6**

OpenThread stack removed TREL IPv6 address

enumerator **OPENTHREAD_EVENT_TREL_MULTICAST_GROUP_JOIN**

OpenThread stack joined TREL IPv6 multicast group

enumerator **OPENTHREAD_EVENT_SET_DNS_SERVER**

OpenThread stack set DNS server >

enumerator **OPENTHREAD_EVENT_PUBLISH_MESHCOP_E**

OpenThread stack start to publish meshcop-e service >

enumerator **OPENTHREAD_EVENT_REMOVE_MESHCOP_E**

OpenThread stack start to remove meshcop-e service >

enumerator **OPENTHREAD_EVENT_DATASET_CHANGED**

OpenThread dataset changed >

enum **esp_openthread_dataset_type_t**

OpenThread dataset type.

Values:

enumerator **OPENTHREAD_ACTIVE_DATASET**

Active dataset

enumerator **OPENTHREAD_PENDING_DATASET**

Pending dataset

enum **esp_openthread_radio_mode_t**

The radio mode of OpenThread.

Values:

enumerator **RADIO_MODE_NATIVE**

Use the native 15.4 radio

enumerator **RADIO_MODE_UART_RCP**

UART connection to a 15.4 capable radio co-processor (RCP)

enumerator **RADIO_MODE_SPI_RCP**

SPI connection to a 15.4 capable radio co-processor (RCP)

enumerator **RADIO_MODE_TREL**

Use the Thread Radio Encapsulation Link (TREL)

enumerator **RADIO_MODE_MAX**

Using for parameter check

enum **esp_openthread_host_connection_mode_t**

How OpenThread connects to the host.

Values:

enumerator **HOST_CONNECTION_MODE_NONE**

Disable host connection

enumerator **HOST_CONNECTION_MODE_CLI_UART**

CLI UART connection to the host

enumerator **HOST_CONNECTION_MODE_CLI_USB**

CLI USB connection to the host

enumerator **HOST_CONNECTION_MODE_RCP_UART**

RCP UART connection to the host

enumerator **HOST_CONNECTION_MODE_RCP_SPI**

RCP SPI connection to the host

enumerator **HOST_CONNECTION_MODE_MAX**

Using for parameter check

Header File

- components/openthread/include/esp_openthread_lock.h
- This header file can be included with:

```
#include "esp_openthread_lock.h"
```

- This header file is a part of the API provided by the `openthread` component. To declare that your component depends on `openthread`, add the following to your CMakeLists.txt:

```
REQUIRES openthread
```

or

```
PRIV_REQUIRES openthread
```

Functions

esp_err_t **esp_openthread_lock_init**(void)

This function initializes the OpenThread API lock.

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if allocation has failed
- ESP_ERR_INVALID_STATE if already initialized

void **esp_openthread_lock_deinit**(void)

This function deinitializes the OpenThread API lock.

bool **esp_openthread_lock_acquire**(TickType_t block_ticks)

This function acquires the OpenThread API lock.

备注: Every Openthread APIs that takes an otInstance argument MUST be protected with this API lock except that the call site is in Openthread callbacks.

参数 **block_ticks** -- **[in]** The maximum number of RTOS ticks to wait for the lock.

返回

- True on lock acquired
- False on failing to acquire the lock with the timeout.

void **esp_openthread_lock_release**(void)

This function releases the OpenThread API lock.

bool **esp_openthread_task_switching_lock_acquire**(TickType_t block_ticks)

This function acquires the OpenThread API task switching lock.

备注: In OpenThread API context, it waits for some actions to be done in other tasks (like lwip), after task switching, it needs to call OpenThread API again. Normally it's not allowed, since the previous OpenThread API lock is not released yet. This task_switching lock allows the OpenThread API can be called in this case.

备注: Please use esp_openthread_lock_acquire() for normal cases.

参数 **block_ticks** -- **[in]** The maximum number of RTOS ticks to wait for the lock.

返回

- True on lock acquired
- False on failing to acquire the lock with the timeout.

void **esp_openthread_task_switching_lock_release**(void)

This function releases the OpenThread API task switching lock.

备注: This API must be called after `esp_openthread_task_switching_lock_acquire` or `esp_openthread_lock_acquire` and will cause a crash if the current task is not the task switching lock holder. This error could be caused by calling OpenThread APIs without locking OpenThread stack.

Header File

- components/openthread/include/esp_openthread_netif_glue.h
- This header file can be included with:

```
#include "esp_openthread_netif_glue.h"
```

- This header file is a part of the API provided by the `openthread` component. To declare that your component depends on `openthread`, add the following to your CMakeLists.txt:

```
REQUIRES openthread
```

or

```
PRIV_REQUIRES openthread
```

Functions

void ***esp_openthread_netif_glue_init**(const *esp_openthread_platform_config_t* *config)

This function initializes the OpenThread network interface glue.

参数 **config** -- **[in]** The platform configuration.

返回

- glue pointer on success
- NULL on failure

void **esp_openthread_netif_glue_deinit**(void)

This function deinitializes the OpenThread network interface glue.

esp_netif_t ***esp_openthread_get_netif**(void)

This function acquires the OpenThread netif.

返回 The OpenThread netif or NULL if not initialzied.

void **esp_openthread_register_meshcop_e_handler**(*esp_event_handler_t* handler, bool for_publish)

This function register a handler for meshcop-e service publish event and remove event.

参数

- **handler** -- **[in]** The handler.
- **for_publish** -- **[in]** The usage of handler, true for publish event and false for remove event.

Macros

ESP_NETIF_INHERENT_DEFAULT_OPENTHREAD()

Default configuration reference of OpenThread esp-netif.

ESP_NETIF_DEFAULT_OPENTHREAD()

Header File

- components/openthread/include/esp_openthread_border_router.h
- This header file can be included with:

```
#include "esp_openthread_border_router.h"
```

- This header file is a part of the API provided by the `openthread` component. To declare that your component depends on `openthread`, add the following to your CMakeLists.txt:

```
REQUIRES openthread
```

or

```
PRIV_REQUIRES openthread
```

Functions

void **esp_openthread_set_backbone_netif**(*esp_netif_t* *backbone_netif)

Sets the backbone interface used for border routing.

备注: This function must be called before esp_openthread_init

参数 **backbone_netif** -- **[in]** The backbone network interface (WiFi or ethernet)

esp_err_t **esp_openthread_border_router_init**(void)

Initializes the border router features of OpenThread.

备注: Calling this function will make the device behave as an OpenThread border router. Kconfig option CONFIG_OPENTHREAD_BORDER_ROUTER is required.

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if feature not supported
- ESP_ERR_INVALID_STATE if already initialized
- ESP_FIAL on other failures

esp_err_t **esp_openthread_border_router_deinit**(void)

Deinitializes the border router features of OpenThread.

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if not initialized
- ESP_FIAL on other failures

esp_netif_t ***esp_openthread_get_backbone_netif**(void)

Gets the backbone interface of OpenThread border router.

返回 The backbone interface or NULL if border router not initialized.

void **esp_openthread_register_rcp_failure_handler**(*esp_openthread_rcp_failure_handler* handler)

Registers the callback for RCP failure.

void **esp_openthread_set_compatibility_error_callback**(*esp_openthread_compatibility_error_callback* callback)

Registers the callback for spinel compatibility error.

备注: This function must be called before esp_openthread_init.

参数 **callback** -- **[in]** The callback.

esp_err_t **esp_openthread_rcp_deinit**(void)

Deinitializes the connection to RCP.

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if fail to deinitialize RCP

esp_err_t **esp_openthread_rcp_init**(void)

Initializes the connection to RCP.

返回

- ESP_OK on success
- ESP_FAIL if fail to initialize RCP

esp_err_t **esp_openthread_set_meshcop_instance_name**(const char *instance_name)

Sets the meshcop(e) instance name.

备注: This function can only be called before `esp_openthread_border_router_init`. If `instance_name` is NULL, then the service will use the hostname as instance name.

参数 **instance_name** -- **[in]** The instance name, can be NULL.

返回

- ESP_OK on success
- ESP_FAIL if fail to initialize RCP

Thread 是一种基于 IPv6 的物联网网状网络技术。

本部分的 Thread API 示例代码存放在 ESP-IDF 示例项目的 openthread 目录下。

2.4.3 ESP-NETIF

ESP-NETIF

ESP-NETIF 库的作用主要体现在以下两方面：

- 在 TCP/IP 协议栈之上为应用程序提供抽象层，允许应用程序在 IP 栈间选择。
- 在底层 TCP/IP 协议栈 API 非线程安全的情况下，也能提供线程安全的 API。

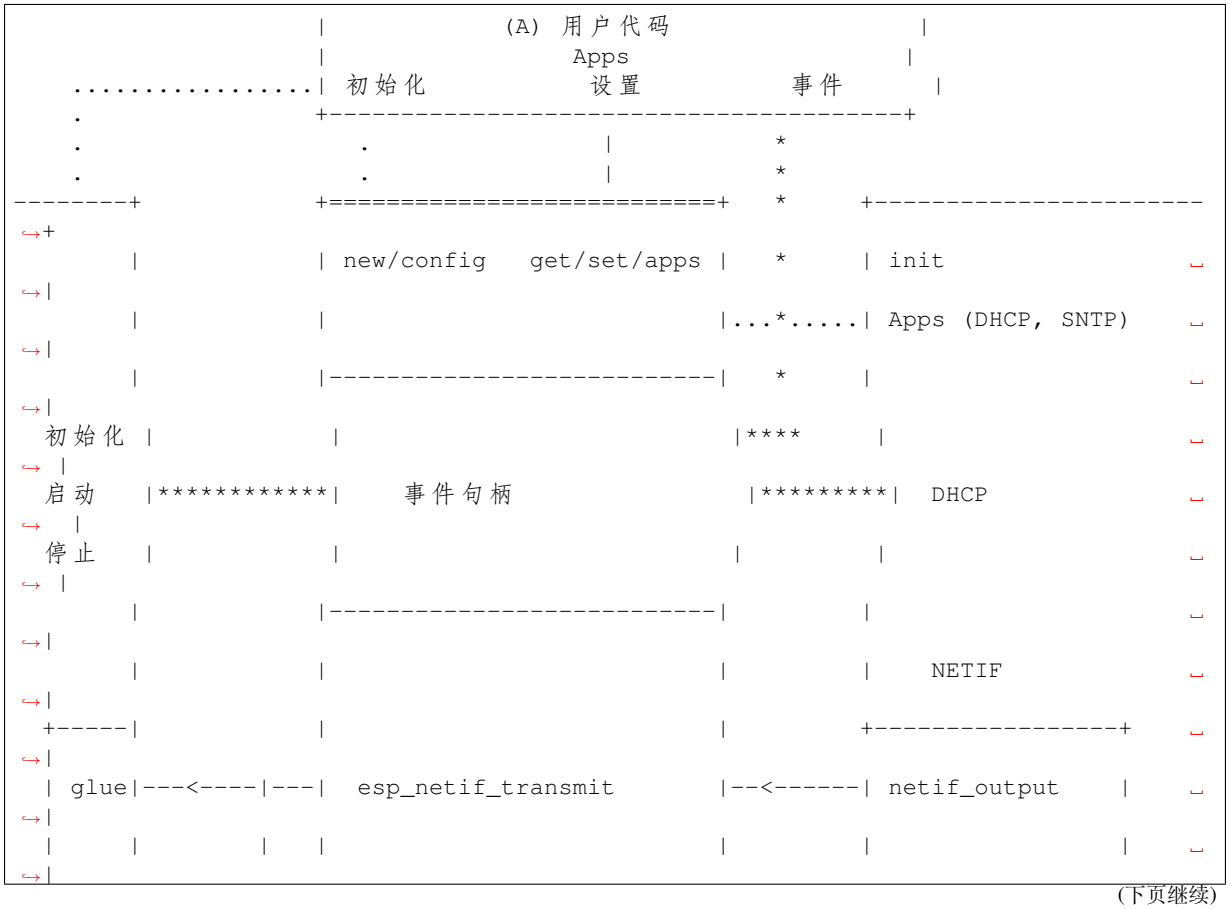
ESP-IDF 目前只为 lwIP TCP/IP 协议栈实现了 ESP-NETIF。然而，该适配器本身不依赖特定 TCP/IP 实现，因而支持不同实现方式。

ESP-IDF 支持实现了 BSD API 的自定义 TCP/IP 协议栈。有关不使用 lwIP 时构建 ESP-IDF 的详细信息，请参阅 components/esp_netif_stack/README.md。

部分 ESP-NETIF 的 API 函数可以被应用程序直接调用，如：获取或设置接口 IP 地址、配置 DHCP 等。其他 ESP-NETIF 的 API 函数则供网络驱动层在 ESP-IDF 内部调用。

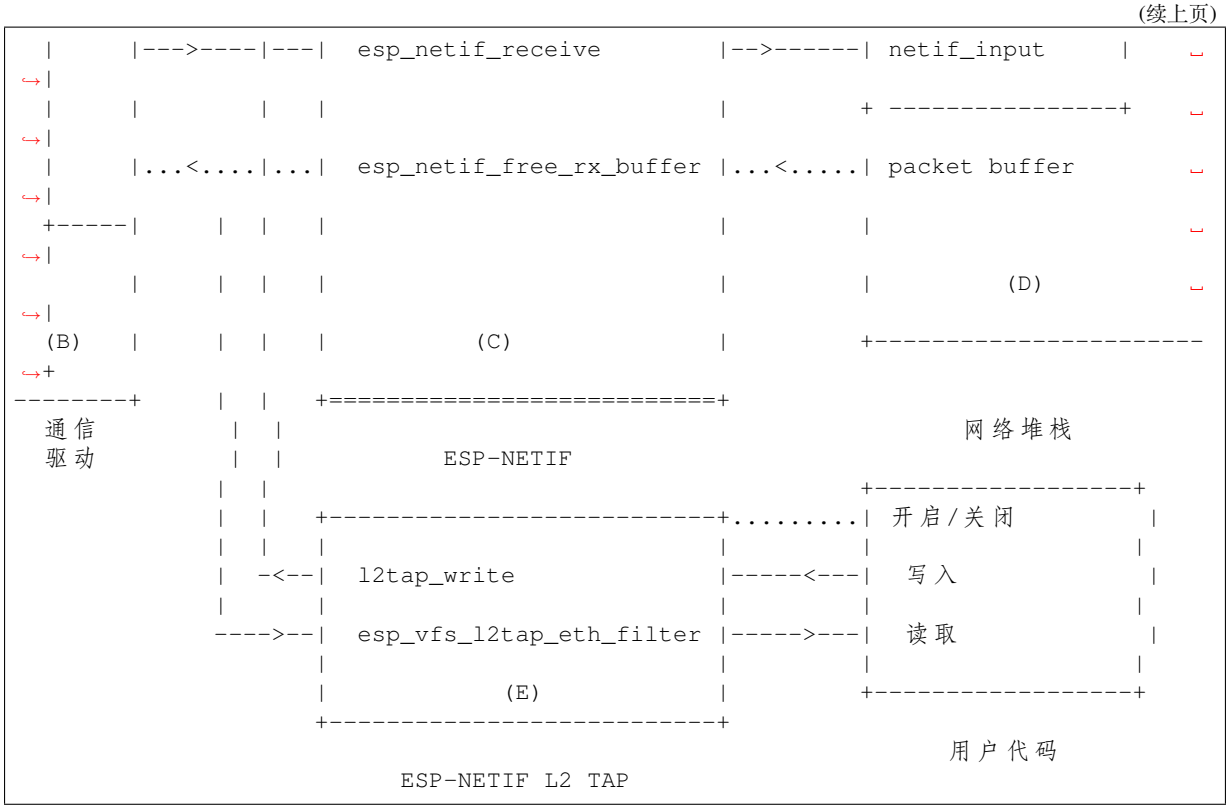
应用程序通常无需直接调用 ESP-NETIF 的 API，它们会由默认网络事件句柄调用。

ESP-NETIF 架构



(下页继续)

(续上页)



图中不同线段对应的数据流和事件流

- `.....` 从用户代码到 ESP-NETIF 和通信驱动的初始化线。
- `--<---->--` 数据包在通信媒介与 TCP/IP 协议栈之间往返。
- `********` 聚集在 ESP-NETIF 中的事件传递到驱动程序、用户代码和网络堆栈中。
- `|` 用户设置及运行时间配置。

ESP-NETIF 交互

A) 用户代码样板 通过使用 ESP-NETIF API 抽象，应用程序与用于通信介质的特定 IO 驱动程序以及配置的 TCP/IP 网络栈之间的交互可以概括如下：

A) 初始化代码

1) 初始化 IO 驱动
2) 创建新的 ESP-NETIF 实例，并完成以下配置：

- ESP-NETIF 的特定选项（flags、行为、名称）
- 网络栈堆选项（netif init 和 input 函数，非公开信息）
- IO 驱动的特定选项（发送、释放 rx 缓冲区功能、IO 驱动句柄）

3) 将 IO 驱动句柄附加到上述步骤所创建的 ESP-NETIF 实例
4) 配置事件句柄

- 对 IO 驱动定义的公共接口使用默认句柄；或为定制的行为或新接口定义特定的句柄
- 为应用程序相关事件（如 IP 丢失或获取）注册句柄

B) 通过 ESP-NETIF API 与网络接口交互

1) 获取、设置 TCP/IP 相关参数（如 DHCP，IP 等）
2) 接收 IP 事件（连接或断连）
3) 控制应用程序生命周期（启用或禁用接口）

B) 通信驱动、IO 驱动和媒介驱动 对于 ESP-NETIF，通信驱动具有以下两个重要作用：

1) 事件句柄：定义与 ESP-NETIF 交互的行为模式（如：连接以太网 -> 开启 netif）
2) 胶合 IO 层：调整输入或输出函数以使用 ESP-NETIF 的传输、接收，并清空接收缓冲区

- 给适当的 ESP-NETIF 对象安装 driver_transmit，以便将网络堆栈中传出的数据包传输给 IO 驱动
- 调用函数`esp_netif_receive()`以便将传入的数据传输给网络堆栈

C) ESP-NETIF ESP-NETIF 是 IO 驱动和网络堆栈间的媒介，用于连通两者之间的数据包传输路径。它提供了一组接口，用于在运行时将驱动程序附加到 ESP-NETIF 对象并在编译期间配置网络堆栈。此外，ESP-NETIF 还提供了一组 API，用于控制网络接口的生命周期及其 TCP/IP 属性。ESP-NETIF 的公共接口大体上可以分为以下六组：

1) 初始化 API（用于创建并配置 ESP-NETIF 实例）
2) 输入或输出 API（用于在 IO 驱动和网络堆栈间传输数据）
3) 事件或行为 API

- 管理网络接口生命周期
- ESP-NETIF 为设计事件句柄提供了构建模块

4) 基本网络接口属性设置器和获取器 API
5) 网络堆栈抽象 API：实现用户与 TCP/IP 堆栈交互

- 启用或禁用接口
- DHCP 服务器和客户端 API
- DNS API
- *SNTP API*

6) 驱动转换工具 API

D) 网络堆栈 网络堆栈与应用程序代码在公共接口方面无公开交互，需通过 ESP-NETIF API 实现完全抽象。

E) ESP-NETIF L2 TAP 接口 ESP-NETIF L2 TAP 接口是 ESP-IDF 访问用户应用程序中的数据链路层（OSI/ISO 中的 L2）以进行帧接收和传输的机制。在嵌入式开发中，它通常用于实现非 IP 相关协议，如 PTP 和 Wake on LAN 等。请注意，目前 ESP-NETIF L2 TAP 接口仅支持以太网 (IEEE 802.3)。

使用 VFS 的文件描述符访问 ESP-NETIF L2 TAP 接口，VFS 文件描述符会提供类似文件的接口（调用 `open()`、`read()`、`write()` 等函数访问），详情请参阅虚拟文件系统组件。

ESP-NETIF 只提供一个 L2 TAP 接口设备（路径名），但由于 ESP-NETIF L2 TAP 接口也可作为第二层基础设施的通用入口点，因此可以同时打开多个不同配置的文件描述符。特定文件描述符的具体配置很关键，它可以配置为仅允许访问由 `if_key`（如 *ETH_DEF*）标识的特定网络接口，并根据帧类型（如 IEEE 802.3 中的以太网类型）过滤特定帧。由于 ESP-NETIF L2 TAP 需要与 IP 堆栈同时存在，因此不应将 IP 相关流量（IP、ARP 等）直接传递给用户应用程序，此时则需要通过配置过滤特定帧实现这一点。在未过滤的情况下，即使该选项仍可配置，也不建议在标准用例中使用。过滤的另一优势在于，过滤后，用户应用程序只能访问它感兴趣的帧类型，其余的流量会传递到其他 L2 TAP 文件描述符或 IP 堆栈。

ESP-NETIF L2 TAP 接口使用手册

初始化 要使用 ESP-NETIF L2 TAP 接口，需要首先通过 Kconfig 配置*CONFIG_ESP_NETIF_L2_TAP* 启用接口，随后通过`esp_vfs_l2tap_intf_register()`注册。请在完成上述步骤后再使用 VFS 函数。

open() ESP-NETIF L2 TAP 注册完成后，可使用路径名"/dev/net/tap" 访问。同一路径名最多可以被打开*CONFIG_ESP_NETIF_L2_TAP_MAX_FDS* 次，多个具有不同配置的文件描述符可以访问数据链路层的各个帧。

ESP-NETIF L2 TAP 可以使用 O_NONBLOCK 文件状态标志打开，确保 read() 不会阻塞。请注意，在当前实现中，当访问网络接口时，由于网络接口被多个 ESP-NETIF L2 TAP 文件描述符和 IP 栈共享，且缺乏列队机制，因此 write() 可能会受阻塞。使用 fcntl() 检索和修改文件状态标志。

成功时，open() 返回新的文件描述符（非负整数）。出错时，返回 -1，并设置 errno 以标识错误。

ioctl() 由于新打开的 ESP-NETIF L2 TAP 文件描述符尚未绑定任意网络接口或配置任意帧类型过滤器，使用前，用户需通过以下选项完成配置：

- L2TAP_S_INTF_DEVICE - 将文件描述符绑定到特定网络接口的选项，该网络接口由其 if_key 标识。ESP-NETIF 网络接口的 if_key 作为第三个参数传输给 ioctl()。ESP-IDF 中，默认网络接口 if_key 的使用存放在 esp_netif/include/esp_netif_defaults.h 头文件中。
- L2TAP_S_DEVICE_DRV_HNDL - 将文件描述符绑定到特定网络接口的另一选项。此时，网络接口直接由其 IO 驱动句柄（例如以太网中的*esp_eth_handle_t*）标识。IO 驱动句柄将作为第三个参数传输给 ioctl()。
- L2TAP_S_RCV_FILTER - 设置过滤器，将特定类型的帧传递到文件描述符。在以太网中，帧是基于长度和以太网类型字段过滤的。如果过滤器值设置为小于或等于 0x05DC，则将以太网类型字段视作 IEEE802.3 长度字段，并将该字段中所有值在 <0, 0x05DC> 区间内的帧传递到文件描述符中。随后，由用户应用程序执行 IEEE802.2 逻辑链路控制 (LLC) 的解析。如果过滤器值设置为大于 0x05DC，则将以太网类型字段视为代表协议标识，仅将与设置值相等的帧传递到文件描述符中。

上述配置选项都支持通过对应获取器选项读取当前配置。

警告： 首先调用 L2TAP_S_INTF_DEVICE 或 L2TAP_S_DEVICE_DRV_HNDL 将文件描述符绑定到特定网络接口，随后方可调用 L2TAP_S_RCV_FILTER 选项。

备注： 当前不支持识别 VLAN 标记帧。如果用户需要处理 VLAN 标记帧，应将过滤器设置为等于 VLAN 标记（即 0x8100 或 0x88A8），并在用户应用程序中处理 VLAN 标记帧。

备注： 当用户应用程序不需要使用 IP 栈时，L2TAP_S_DEVICE_DRV_HNDL 将非常适用，也无需初始化 ESP-NETIF。但在此情况下，网络接口无法通过 if_key 来识别，需要通过 IO 驱动程序句柄直接标识网络接口。

成功时，ioctl() 返回 0。出错时，返回 -1，并设置 errno 以指示错误类型:
* EBADF - 文件描述符无效。
* EACCES - 此状态下无法改变选项（例如文件描述符尚未绑定到网络接口）。
* EINVAL - 配置参数无效。同一网络接口上的其他文件描述符已经使用了以太网类型过滤器。
* ENODEV - 此文件描述符尝试分配给的网络接口不存在。
* ENOSYS - 不支持该操作，传递的配置选项不存在。

fcntl() fcntl() 配置已开启的 ESP-NETIF L2 TAP 文件描述符属性。

以下命令调控与文件描述符相关的状态标志：

- F_GETFD - 函数返回文件描述符标志，忽略第三个参数。
- F_SETFD - 将文件描述符标志设置为第三个参数的指定值。返回零。

成功时，ioctl() 返回 0。出错时，返回 -1，并设置 errno 以指示错误类型。
* EBADF - 文件描述符无效。
* ENOSYS - 不支持该命令。

read() 已开启并完成配置的 ESP-NETIF L2 TAP 文件描述符可通过 `read()` 获取入站帧。读取可以是阻塞或非阻塞的，具体取决于 `O_NONBLOCK` 文件状态标志的实际状态。当文件状态标志设置为阻塞时，读取程序将等待，直到接收到帧，并将上下文切换到其他任务。当文件状态标志设置为非阻塞时，立即返回读取程序。在此情况下，如果已经帧已经入队，则返回一帧，否则函数指示队列为空。与文件描述符关联的队列帧数量受*CONFIG_ESP_NETIF_L2_TAP_RX_QUEUE_SIZE* Kconfig 选项限制。一旦队列里帧的数量达到配置的阈值，新到达的帧将被丢弃，直到队列有足够的空间接受传入的流量（队尾丢弃队列管理）。

成功时，`read()` 函数返回读取的字节数。当目标缓冲区的大小为 0 时，函数返回 0。出错时，函数返回 -1，并设置 `errno` 以指示错误类型。
* EBADF - 文件描述符无效。
* EAGAIN - 文件描述符标记为非阻塞 (`O_NONBLOCK`)，但读取受阻塞。

write() 通过已开启并完成配置的 ESP-NETIF L2 TAP 文件描述符可以将原始数据链路层帧发送到网络接口，用户应用程序负责构建除物理接口设备自动添加的字段外的整个帧。在以太网链路中，用户应用程序需要构建以下字段：源或目的 MAC 地址、以太网类型、实际协议头和用户数据，字段长度如下表：

目的 MAC	源 MAC	类型/长度	负载（协议头/数据）
6 B	6 B	2 B	0-1486 B

换句话说，ESP-NETIF L2 TAP 接口不会对数据帧进行额外处理，只会检查数据帧的以太网类型是否与文件描述符配置的过滤器相同。如果以太网类型不同，则会返回错误，并且不发送数据帧。需要注意的是，由于网络接口是由多个 ESP-NETIF L2 TAP 文件描述符和 IP 栈共享的资源，且当前缺乏列队机制，当前实现中的 `write()` 在进入网络接口时可能会受阻塞，。

成功时，`write()` 返回已写入的字节数。如果输入缓冲区的大小为 0，则返回 0。出错时，则返回 -1，并设置 `errno` 以指示错误类型。
* EBADF - 文件描述符无效。
* EBADMSG - 帧的以太网类型与文件描述符配置的过滤器不同。
* EIO - 网络接口不可用或正忙。

close() 已开启的 ESP-NETIF L2 TAP 文件描述符可以通过 `close()` 函数关闭，释放其分配到的资源。ESP-NETIF L2 TAP 实现的 `close()` 函数可能会受阻塞，但它是线程安全的，可以从与实际使用文件描述符的任务不同的任务中调用。如果出现一个任务在 I/O 操作中被阻塞、另一个任务试图关闭文件描述符的情况，则第一个任务会解除阻塞，其读取程序以错误结束。

成功时，`close()` 返回 0。出错时，则返回 -1，并设置 `errno` 以指示错误类型。
* EBADF - 文件描述符无效。

select() `select()` 函数按标准方法使用，启用*CONFIG_VFS_SUPPORT_SELECT* 即可使用该函数。

SNTP API SNTP 的简要介绍、初始化代码和基本模式请参阅*系统时间* 的*SNTP 时间同步* 小节。

本节介绍了使用 SNTP 服务特定用例的详细信息，包括静态配置的服务器、使用 DHCP 提供的服务器或两者兼备的情况，操作流程如下：

1) 调用*esp_netif_sntp_init()* 初始化服务并完成配置。此操作只能执行一次（除非已调用*esp_netif_sntp_deinit()* 销毁 SNTP 服务）。

2) 调用*esp_netif_sntp_start()* 启动服务。如果在前一步中已经默认启动了服务，则不需要此步骤。如果需使用通过 DHCP 获取的 NTP 服务器，推荐在完成连接后显式启动该服务。注意，应在连接前启用通过 DHCP 获取的 NTP 服务器选项，并在连接后再启用 SNTP 服务。
3) 需要时，可调用*esp_netif_sntp_sync_wait()* 等待系统时间同步。
4) 调用*esp_netif_sntp_deinit()* 停止并销毁服务。

使用静态定义服务器的基本模式 连接到网络后，使用默认配置初始化该模块。注意，在配置结构体中可提供多个 NTP 服务器：

```
esp_sntp_config_t config = ESP_NETIF_SNTP_DEFAULT_CONFIG_MULTIPLE(2,
                           ESP_SNTP_SERVER_LIST("time.windows.com", "pool.ntp.org"␣
↪) );
esp_netif_sntp_init(&config);
```

备注： 要配置多个 SNTP 服务器，需要更新 lwIP 配置，请参阅*CONFIG_LWIP_SNTP_MAX_SERVERS*。

使用 DHCP 获取的 SNTP 服务器 首先，激活 lwIP 配置选项，相关配置请参阅*CONFIG_LWIP_DHCP_GET_NTP_SRV*。其次，在使用 DHCP 选项、且不使用 NTP 服务器的情况下初始化 SNTP 模块，代码如下：

```
esp_sntp_config_t config = ESP_NETIF_SNTP_DEFAULT_CONFIG_MULTIPLE(0, {} );
config.start = false;                       // 启动 SNTP 服务
config.server_from_dhcp = true;             // 接收来自 DHCP 服务器的 NTP␣
↪服务提供方案
esp_netif_sntp_init(&config);
```

连接成功后，可通过以下代码启动服务器：

```
esp_netif_sntp_start();
```

备注： 也可选择在初始化期间启动服务（即默认 `config.start=true`）。注意，此时连接尚未完成，可能导致初始 SNTP 请求失败，并增加后续各次请求之间的延迟时间。

同时使用静态和动态服务器 同时使用静态和动态服务器的流程与使用 DHCP 获取的 SNTP 服务器基本相同。配置时，用户应确保在通过 DHCP 获取 NTP 服务器时刷新静态服务器配置。底层 lwIP 代码会在接受 DHCP 提供的信息时清理其余的 NTP 服务器列表。因此，ESP-NETIF SNTP 模块会保存静态配置的服务器，并在获取 DHCP 租约后对其重新配置。

典型配置依次如下，提供特定 `IP_EVENT` 更新配置，并提供第一个服务器的索引完成重新配置（例如，设置 `config.index_of_first_server=1` 会将 DHCP 提供的服务器保留在索引 0，而将静态配置的服务器保留在索引 1）。

```
esp_sntp_config_t config = ESP_NETIF_SNTP_DEFAULT_CONFIG("pool.ntp.org");
config.start = false;                       // 启动 SNTP 服务（连接成功后）
config.server_from_dhcp = true;             // 接收来自 DHCP 服务器的 NTP␣
↪服务提供方案
config.renew_servers_after_new_IP = true;   // 让 esp-netif 在接收到 DHCP␣
↪租约后更新配置的 SNTP 服务器
config.index_of_first_server = 1;           // 服务器 1 开始更新，保留从 DHCP␣
↪获取的服务器 0 的设置
config.ip_event_to_renew = IP_EVENT_STA_GOT_IP;  // 基于 IP 事件刷新配置
```

随后，调用*esp_netif_sntp_start()* 启用服务。

ESP-NETIF 编程手册 请参阅示例部分，了解默认接口的基本初始化：

- 以太网 ethernet/basic/main/ethernet_example_main.c
- L2 TAP protocols/l2tap/main/l2tap_main.c

更多示例请参阅*ESP-NETIF 自定义 I/O 驱动程序*。

API 参考

Header File

- components/esp_netif/include/esp_netif.h
- This header file can be included with:

```
#include "esp_netif.h"
```

- This header file is a part of the API provided by the `esp_netif` component. To declare that your component depends on `esp_netif`, add the following to your CMakeLists.txt:

```
REQUIRES esp_netif
```

or

```
PRIV_REQUIRES esp_netif
```

Functions

esp_err_t **esp_netif_init**(void)

Initialize the underlying TCP/IP stack.

备注： This function should be called exactly once from application code, when the application starts up.

返回

- ESP_OK on success
- ESP_FAIL if initializing failed

esp_err_t **esp_netif_deinit**(void)

Deinitialize the esp-netif component (and the underlying TCP/IP stack)

```
Note: Deinitialization is not supported yet
```

返回

- ESP_ERR_INVALID_STATE if esp_netif not initialized
- ESP_ERR_NOT_SUPPORTED otherwise

esp_netif_t ***esp_netif_new**(const *esp_netif_config_t* *esp_netif_config)

Creates an instance of new esp-netif object based on provided config.

参数 **esp_netif_config** -- **[in]** pointer esp-netif configuration

返回

- pointer to esp-netif object on success
- NULL otherwise

void **esp_netif_destroy**(*esp_netif_t* *esp_netif)

Destroys the esp_netif object.

参数 **esp_netif** -- **[in]** pointer to the object to be deleted

esp_err_t **esp_netif_set_driver_config**(*esp_netif_t* *esp_netif, const *esp_netif_driver_ifconfig_t* *driver_config)

Configures driver related options of esp_netif object.

参数

- **esp_netif** -- **[inout]** pointer to the object to be configured
- **driver_config** -- **[in]** pointer esp-netif io driver related configuration

返回

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS if invalid parameters provided

esp_err_t **esp_netif_attach**(*esp_netif_t* *esp_netif, *esp_netif_iodriver_handle* driver_handle)

Attaches esp_netif instance to the io driver handle.

Calling this function enables connecting specific esp_netif object with already initialized io driver to update esp_netif object with driver specific configuration (i.e. calls post_attach callback, which typically sets io driver callbacks to esp_netif instance and starts the driver)

参数

- **esp_netif** -- **[inout]** pointer to esp_netif object to be attached
- **driver_handle** -- **[in]** pointer to the driver handle

返回

- ESP_OK on success
- ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED if driver's pot_attach callback failed

esp_err_t **esp_netif_receive**(*esp_netif_t* *esp_netif, void *buffer, size_t len, void *eb)

Passes the raw packets from communication media to the appropriate TCP/IP stack.

This function is called from the configured (peripheral) driver layer. The data are then forwarded as frames to the TCP/IP stack.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **buffer** -- **[in]** Received data
- **len** -- **[in]** Length of the data frame
- **eb** -- **[in]** Pointer to internal buffer (used in Wi-Fi driver)

返回

- ESP_OK

void **esp_netif_action_start**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver start event Creates network interface, if AUTOUP enabled turns the interface on, if DHCPS enabled starts dhcp server.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

void **esp_netif_action_stop**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver stop event.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

void **esp_netif_action_connected**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver connected event.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

void **esp_netif_action_disconnected**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver disconnected event.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

void **esp_netif_action_got_ip**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon network got IP event.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

void **esp_netif_action_join_ip6_multicast_group**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IPv6 multicast group join.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

void **esp_netif_action_leave_ip6_multicast_group**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IPv6 multicast group leave.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

void **esp_netif_action_add_ip6_address**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IPv6 address added by the underlying stack.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

void **esp_netif_action_remove_ip6_address**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IPv6 address removed by the underlying stack.

备注: This API can be directly used as event handler

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **base** -- The base type of the event
- **event_id** -- The specific ID of the event
- **data** -- Optional data associated with the event

esp_err_t **esp_netif_set_default_netif**(*esp_netif_t* *esp_netif)

Manual configuration of the default netif.

This API overrides the automatic configuration of the default interface based on the route_prio If the selected netif is set default using this API, no other interface could be set-default disregarding its route_prio number (unless the selected netif gets destroyed)

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 ESP_OK on success

esp_netif_t ***esp_netif_get_default_netif**(void)

Getter function of the default netif.

This API returns the selected default netif.

返回 Handle to esp-netif instance of the default netif.

esp_err_t **esp_netif_join_ip6_multicast_group**(*esp_netif_t* *esp_netif, const *esp_ip6_addr_t* *addr)

Cause the TCP/IP stack to join a IPv6 multicast group.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **addr** -- **[in]** The multicast group to join

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_MLD6_FAILED
- ESP_ERR_NO_MEM

esp_err_t **esp_netif_leave_ip6_multicast_group**(*esp_netif_t* *esp_netif, const *esp_ip6_addr_t* *addr)

Cause the TCP/IP stack to leave a IPv6 multicast group.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **addr** -- **[in]** The multicast group to leave

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_MLD6_FAILED
- ESP_ERR_NO_MEM

esp_err_t **esp_netif_set_mac**(*esp_netif_t* *esp_netif, uint8_t mac[])

Set the mac address for the interface instance.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **mac** -- **[in]** Desired mac address for the related network interface

返回

- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_NOT_SUPPORTED - mac not supported on this interface

esp_err_t **esp_netif_get_mac**(*esp_netif_t* *esp_netif, uint8_t mac[])

Get the mac address for the interface instance.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **mac** -- **[out]** Resultant mac address for the related network interface

返回

- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_NOT_SUPPORTED - mac not supported on this interface

esp_err_t **esp_netif_set_hostname**(*esp_netif_t* *esp_netif, const char *hostname)

Set the hostname of an interface.

The configured hostname overrides the default configuration value CONFIG_LWIP_LOCAL_HOSTNAME. Please note that when the hostname is altered after interface started/connected the changes would only be reflected once the interface restarts/reconnects

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **hostname** -- **[in]** New hostname for the interface. Maximum length 32 bytes.

返回

- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_ESP_NETIF_INVALID_PARAMS - parameter error

esp_err_t **esp_netif_get_hostname**(*esp_netif_t* *esp_netif, const char **hostname)

Get interface hostname.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **hostname** -- **[out]** Returns a pointer to the hostname. May be NULL if no hostname is set. If set non-NULL, pointer remains valid (and string may change if the hostname changes).

返回

- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_ESP_NETIF_INVALID_PARAMS - parameter error

bool **esp_netif_is_netif_up**(*esp_netif_t* *esp_netif)

Test if supplied interface is up or down.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance

返回

- true - Interface is up
- false - Interface is down

esp_err_t **esp_netif_get_ip_info**(*esp_netif_t* *esp_netif, *esp_netif_ip_info_t* *ip_info)

Get interface's IP address information.

If the interface is up, IP information is read directly from the TCP/IP stack. If the interface is down, IP information is read from a copy kept in the ESP-NETIF instance

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **ip_info** -- **[out]** If successful, IP information will be returned in this argument.

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

esp_err_t **esp_netif_get_old_ip_info**(*esp_netif_t* *esp_netif, *esp_netif_ip_info_t* *ip_info)

Get interface's old IP information.

Returns an "old" IP address previously stored for the interface when the valid IP changed.

If the IP lost timer has expired (meaning the interface was down for longer than the configured interval) then the old IP information will be zero.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **ip_info** -- **[out]** If successful, IP information will be returned in this argument.

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

esp_err_t **esp_netif_set_ip_info**(*esp_netif_t* *esp_netif, const *esp_netif_ip_info_t* *ip_info)

Set interface's IP address information.

This function is mainly used to set a static IP on an interface.

If the interface is up, the new IP information is set directly in the TCP/IP stack.

The copy of IP information kept in the ESP-NETIF instance is also updated (this copy is returned if the IP is queried while the interface is still down.)

备注： DHCP client/server must be stopped (if enabled for this interface) before setting new IP information.

备 注: Calling this interface for may generate a SYSTEM_EVENT_STA_GOT_IP or SYSTEM_EVENT_ETH_GOT_IP event.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **ip_info** -- **[in]** IP information to set on the specified interface

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED If DHCP server or client is still running

esp_err_t **esp_netif_set_old_ip_info**(*esp_netif_t* *esp_netif, const *esp_netif_ip_info_t* *ip_info)

Set interface old IP information.

This function is called from the DHCP client (if enabled), before a new IP is set. It is also called from the default handlers for the SYSTEM_EVENT_STA_CONNECTED and SYSTEM_EVENT_ETH_CONNECTED events.

Calling this function stores the previously configured IP, which can be used to determine if the IP changes in the future.

If the interface is disconnected or down for too long, the "IP lost timer" will expire (after the configured interval) and set the old IP information to zero.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **ip_info** -- **[in]** Store the old IP information for the specified interface

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

int **esp_netif_get_netif_impl_index**(*esp_netif_t* *esp_netif)

Get net interface index from network stack implementation.

备注: This index could be used in `setsockopt()` to bind socket with multicast interface

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 implementation specific index of interface represented with supplied esp_netif

esp_err_t **esp_netif_get_netif_impl_name**(*esp_netif_t* *esp_netif, char *name)

Get net interface name from network stack implementation.

备注: This name could be used in `setsockopt()` to bind socket with appropriate interface

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **name** -- **[out]** Interface name as specified in underlying TCP/IP stack. Note that the actual name will be copied to the specified buffer, which must be allocated to hold maximum interface name size (6 characters for lwIP)

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

esp_err_t **esp_netif_napt_enable**(*esp_netif_t* *esp_netif)

Enable NAPT on an interface.

备注: Enable operation can be performed only on one interface at a time. NAPT cannot be enabled on multiple interfaces according to this implementation.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance

返回

- ESP_OK
- ESP_FAIL
- ESP_ERR_NOT_SUPPORTED

esp_err_t **esp_netif_napt_disable**(*esp_netif_t* *esp_netif)

Disable NAPT on an interface.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance

返回

- ESP_OK
- ESP_FAIL
- ESP_ERR_NOT_SUPPORTED

esp_err_t **esp_netif_dhcps_option**(*esp_netif_t* *esp_netif, *esp_netif_dhcp_option_mode_t* opt_op, *esp_netif_dhcp_option_id_t* opt_id, void *opt_val, uint32_t opt_len)

Set or Get DHCP server option.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **opt_op** -- **[in]** ESP_NETIF_OP_SET to set an option, ESP_NETIF_OP_GET to get an option.
- **opt_id** -- **[in]** Option index to get or set, must be one of the supported enum values.
- **opt_val** -- **[inout]** Pointer to the option parameter.
- **opt_len** -- **[in]** Length of the option parameter.

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

esp_err_t **esp_netif_dhcpc_option**(*esp_netif_t* *esp_netif, *esp_netif_dhcp_option_mode_t* opt_op, *esp_netif_dhcp_option_id_t* opt_id, void *opt_val, uint32_t opt_len)

Set or Get DHCP client option.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **opt_op** -- **[in]** ESP_NETIF_OP_SET to set an option, ESP_NETIF_OP_GET to get an option.
- **opt_id** -- **[in]** Option index to get or set, must be one of the supported enum values.
- **opt_val** -- **[inout]** Pointer to the option parameter.
- **opt_len** -- **[in]** Length of the option parameter.

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

esp_err_t **esp_netif_dhcpc_start**(*esp_netif_t* *esp_netif)

Start DHCP client (only if enabled in interface object)

备 注: The default event handlers for the SYSTEM_EVENT_STA_CONNECTED and SYSTEM_EVENT_ETH_CONNECTED events call this function.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回
- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED
- ESP_ERR_ESP_NETIF_DHCPC_START_FAILED

esp_err_t **esp_netif_dhcpc_stop**(*esp_netif_t* *esp_netif)

Stop DHCP client (only if enabled in interface object)

备注: Calling action_netif_stop() will also stop the DHCP Client if it is running.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回
- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_IF_NOT_READY

esp_err_t **esp_netif_dhcpc_get_status**(*esp_netif_t* *esp_netif, *esp_netif_dhcp_status_t* *status)

Get DHCP client status.

参数
- **esp_netif** -- **[in]** Handle to esp-netif instance
- **status** -- **[out]** If successful, the status of DHCP client will be returned in this argument.

返回
- ESP_OK

esp_err_t **esp_netif_dhcps_get_status**(*esp_netif_t* *esp_netif, *esp_netif_dhcp_status_t* *status)

Get DHCP Server status.

参数
- **esp_netif** -- **[in]** Handle to esp-netif instance
- **status** -- **[out]** If successful, the status of the DHCP server will be returned in this argument.

返回
- ESP_OK

esp_err_t **esp_netif_dhcps_start**(*esp_netif_t* *esp_netif)

Start DHCP server (only if enabled in interface object)

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回
- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

esp_err_t **esp_netif_dhcps_stop**(*esp_netif_t* *esp_netif)

Stop DHCP server (only if enabled in interface object)

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回
- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_IF_NOT_READY

esp_err_t **esp_netif_dhcps_get_clients_by_mac**(*esp_netif_t* *esp_netif, int num, *esp_netif_pair_mac_ip_t* *mac_ip_pair)

Populate IP addresses of clients connected to DHCP server listed by their MAC addresses.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **num** -- **[in]** Number of clients with specified MAC addresses in the array of pairs
- **mac_ip_pair** -- **[inout]** Array of pairs of MAC and IP addresses (MAC are inputs, IP outputs)

返回

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS on invalid params
- ESP_ERR_NOT_SUPPORTED if DHCP server not enabled

esp_err_t **esp_netif_set_dns_info**(*esp_netif_t* *esp_netif, *esp_netif_dns_type_t* type, *esp_netif_dns_info_t* *dns)

Set DNS Server information.

This function behaves differently if DHCP server or client is enabled

If DHCP client is enabled, main and backup DNS servers will be updated automatically from the DHCP lease if the relevant DHCP options are set. Fallback DNS Server is never updated from the DHCP lease and is designed to be set via this API. If DHCP client is disabled, all DNS server types can be set via this API only.

Note that LWIP stores DNS server information globally, not per interface, so the first parameter is unused in the default LWIP configuration. If CONFIG_ESP_NETIF_SET_DNS_PER_DEFAULT_NETIF=1 this API sets internal DNS server information per netif. It's also possible to set the global DNS server info by supplying esp_netif=NULL

If DHCP server is enabled, the Main DNS Server setting is used by the DHCP server to provide a DNS Server option to DHCP clients (Wi-Fi stations).

- The default Main DNS server is typically the IP of the DHCP server itself.
- This function can override it by setting server type ESP_NETIF_DNS_MAIN.
- Other DNS Server types are not supported for the DHCP server.
- To propagate the DNS info to client, please stop the DHCP server before using this API.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **type** -- **[in]** Type of DNS Server to set: ESP_NETIF_DNS_MAIN, ESP_NETIF_DNS_BACKUP, ESP_NETIF_DNS_FALLBACK
- **dns** -- **[in]** DNS Server address to set

返回

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS invalid params

esp_err_t **esp_netif_get_dns_info**(*esp_netif_t* *esp_netif, *esp_netif_dns_type_t* type, *esp_netif_dns_info_t* *dns)

Get DNS Server information.

Return the currently configured DNS Server address for the specified interface and Server type.

This may be result of a previous call to *esp_netif_set_dns_info()*. If the interface's DHCP client is enabled, the Main or Backup DNS Server may be set by the current DHCP lease.

Note that LWIP stores DNS server information globally, not per interface, so the first parameter is unused in the default LWIP configuration. If CONFIG_ESP_NETIF_SET_DNS_PER_DEFAULT_NETIF=1 this API returns internally saved DNS server information per netif. It's also possible to ask for the global DNS server info by supplying esp_netif=NULL

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **type** -- **[in]** Type of DNS Server to get: ESP_NETIF_DNS_MAIN, ESP_NETIF_DNS_BACKUP, ESP_NETIF_DNS_FALLBACK
- **dns** -- **[out]** DNS Server result is written here on success

返回

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS invalid params

esp_err_t **esp_netif_create_ip6_linklocal** (*esp_netif_t* *esp_netif)

Create interface link-local IPv6 address.

Cause the TCP/IP stack to create a link-local IPv6 address for the specified interface.

This function also registers a callback for the specified interface, so that if the link-local address becomes verified as the preferred address then a SYSTEM_EVENT_GOT_IP6 event will be sent.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

esp_err_t **esp_netif_get_ip6_linklocal** (*esp_netif_t* *esp_netif, *esp_ip6_addr_t* *if_ip6)

Get interface link-local IPv6 address.

If the specified interface is up and a preferred link-local IPv6 address has been created for the interface, return a copy of it.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **if_ip6** -- **[out]** IPv6 information will be returned in this argument if successful.

返回

- ESP_OK
- ESP_FAIL If interface is down, does not have a link-local IPv6 address, or the link-local IPv6 address is not a preferred address.

esp_err_t **esp_netif_get_ip6_global** (*esp_netif_t* *esp_netif, *esp_ip6_addr_t* *if_ip6)

Get interface global IPv6 address.

If the specified interface is up and a preferred global IPv6 address has been created for the interface, return a copy of it.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **if_ip6** -- **[out]** IPv6 information will be returned in this argument if successful.

返回

- ESP_OK
- ESP_FAIL If interface is down, does not have a global IPv6 address, or the global IPv6 address is not a preferred address.

int **esp_netif_get_all_ip6** (*esp_netif_t* *esp_netif, *esp_ip6_addr_t* if_ip6[])

Get all IPv6 addresses of the specified interface.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **if_ip6** -- **[out]** Array of IPv6 addresses will be copied to the argument

返回 number of returned IPv6 addresses

int **esp_netif_get_all_preferred_ip6** (*esp_netif_t* *esp_netif, *esp_ip6_addr_t* if_ip6[])

Get all preferred IPv6 addresses of the specified interface.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **if_ip6** -- **[out]** Array of IPv6 addresses will be copied to the argument

返回 number of returned IPv6 addresses

esp_err_t **esp_netif_add_ip6_address**(*esp_netif_t* *esp_netif, const *esp_ip6_addr_t* addr, bool preferred)

Cause the TCP/IP stack to add an IPv6 address to the interface.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **addr** -- **[in]** The address to be added
- **preferred** -- **[in]** The preferred status of the address

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_IP6_ADDR_FAILED
- ESP_ERR_NO_MEM

esp_err_t **esp_netif_remove_ip6_address**(*esp_netif_t* *esp_netif, const *esp_ip6_addr_t* *addr)

Cause the TCP/IP stack to remove an IPv6 address from the interface.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **addr** -- **[in]** The address to be removed

返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_IP6_ADDR_FAILED
- ESP_ERR_NO_MEM

void **esp_netif_set_ip4_addr**(*esp_ip4_addr_t* *addr, uint8_t a, uint8_t b, uint8_t c, uint8_t d)

Sets IPv4 address to the specified octets.

参数

- **addr** -- **[out]** IP address to be set
- **a** -- the first octet (127 for IP 127.0.0.1)
- **b** --
- **c** --
- **d** --

char ***esp_ip4addr_ntoa**(const *esp_ip4_addr_t* *addr, char *buf, int buflen)

Converts numeric IP address into decimal dotted ASCII representation.

参数

- **addr** -- ip address in network order to convert
- **buf** -- target buffer where the string is stored
- **buflen** -- length of buf

返回 either pointer to buf which now holds the ASCII representation of addr or NULL if buf was too small

uint32_t **esp_ip4addr_aton**(const char *addr)

Ascii internet address interpretation routine The value returned is in network order.

参数 **addr** -- IP address in ascii representation (e.g. ”127.0.0.1”)

返回 ip address in network order

esp_err_t **esp_netif_str_to_ip4**(const char *src, *esp_ip4_addr_t* *dst)

Converts Ascii internet IPv4 address into esp_ip4_addr_t.

参数

- **src** -- **[in]** IPv4 address in ascii representation (e.g. ”127.0.0.1”)
- **dst** -- **[out]** Address of the target esp_ip4_addr_t structure to receive converted address

返回

- ESP_OK on success
- ESP_FAIL if conversion failed

- ESP_ERR_INVALID_ARG if invalid parameter is passed into

esp_err_t **esp_netif_str_to_ip6**(const char *src, *esp_ip6_addr_t* *dst)

Converts Ascii internet IPv6 address into esp_ip4_addr_t Zeros in the IP address can be stripped or completely omitted: "2001:db8:85a3:0:0:0:2:1" or "2001:db8::2:1")

参数

- **src** -- **[in]** IPv6 address in ascii representation (e.g. ""2001:0db8:85a3:0000:0000:0000:0002:0001")
- **dst** -- **[out]** Address of the target esp_ip6_addr_t structure to receive converted address

返回

- ESP_OK on success
- ESP_FAIL if conversion failed
- ESP_ERR_INVALID_ARG if invalid parameter is passed into

esp_netif_iodriver_handle **esp_netif_get_io_driver**(*esp_netif_t* *esp_netif)

Gets media driver handle for this esp-netif instance.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 opaque pointer of related IO driver

esp_netif_t ***esp_netif_get_handle_from_ifkey**(const char *if_key)

Searches over a list of created objects to find an instance with supplied if key.

参数 **if_key** -- Textual description of network interface
返回 Handle to esp-netif instance

esp_netif_flags_t **esp_netif_get_flags**(*esp_netif_t* *esp_netif)

Returns configured flags for this interface.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 Configuration flags

const char ***esp_netif_get_ifkey**(*esp_netif_t* *esp_netif)

Returns configured interface key for this esp-netif instance.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 Textual description of related interface

const char ***esp_netif_get_desc**(*esp_netif_t* *esp_netif)

Returns configured interface type for this esp-netif instance.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 Enumerated type of this interface, such as station, AP, ethernet

int **esp_netif_get_route_prio**(*esp_netif_t* *esp_netif)

Returns configured routing priority number.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 Integer representing the instance's route-prio, or -1 if invalid parameters

int32_t **esp_netif_get_event_id**(*esp_netif_t* *esp_netif, *esp_netif_ip_event_type_t* event_type)

Returns configured event for this esp-netif instance and supplied event type.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **event_type** -- (either get or lost IP)

返回 specific event id which is configured to be raised if the interface lost or acquired IP address
-1 if supplied event_type is not known

esp_netif_t ***esp_netif_next**(*esp_netif_t* *esp_netif)

Iterates over list of interfaces. Returns first netif if NULL given as parameter.

备注： This API doesn't lock the list, nor the TCPIP context, as this it's usually required to get atomic access between iteration steps rather that within a single iteration. Therefore it is recommended to iterate over the interfaces inside *esp_netif_tcpip_exec()*

备注： This API is deprecated. Please use *esp_netif_next_unsafe()* directly if all the system interfaces are under your control and you can safely iterate over them. Otherwise, iterate over interfaces using *esp_netif_tcpip_exec()*, or use *esp_netif_find_if()* to search in the list of netifs with defined predicate.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 First netif from the list if supplied parameter is NULL, next one otherwise

esp_netif_t ***esp_netif_next_unsafe**(*esp_netif_t* *esp_netif)

Iterates over list of interfaces without list locking. Returns first netif if NULL given as parameter.

Used for bulk search loops within TCPIP context, e.g. using *esp_netif_tcpip_exec()*, or if we're sure that the iteration is safe from our application perspective (e.g. no interface is removed between iterations)

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 First netif from the list if supplied parameter is NULL, next one otherwise

esp_netif_t ***esp_netif_find_if**(*esp_netif_find_predicate_t* fn, void *ctx)

Return a netif pointer for the first interface that meets criteria defined by the callback.

参数
- **fn** -- Predicate function returning true for the desired interface
- **ctx** -- Context pointer passed to the predicate, typically a descriptor to compare with

返回 valid netif pointer if found, NULL if not

size_t **esp_netif_get_nr_of_ifs**(void)

Returns number of registered esp_netif objects.

返回 Number of esp_netifs

void **esp_netif_netstack_buf_ref**(void *netstack_buf)

increase the reference counter of net stack buffer

参数 **netstack_buf** -- **[in]** the net stack buffer

void **esp_netif_netstack_buf_free**(void *netstack_buf)

free the netstack buffer

参数 **netstack_buf** -- **[in]** the net stack buffer

esp_err_t **esp_netif_tcpip_exec**(*esp_netif_callback_fn* fn, void *ctx)

Utility to execute the supplied callback in TCP/IP context.

参数
- **fn** -- Pointer to the callback
- **ctx** -- Parameter to the callback

返回 The error code (esp_err_t) returned by the callback

Type Definitions

typedef bool (***esp_netif_find_predicate_t**)(*esp_netif_t* *netif, void *ctx)

Predicate callback for *esp_netif_find_if()* used to find interface which meets defined criteria.

typedef *esp_err_t* (***esp_netif_callback_fn**)(void *ctx)

TCPIP thread safe callback used with *esp_netif_tcpip_exec()*

Header File

- components/esp_netif/include/esp_netif_sntp.h
- This header file can be included with:

```
#include "esp_netif_sntp.h"
```

- This header file is a part of the API provided by the `esp_netif` component. To declare that your component depends on `esp_netif`, add the following to your CMakeLists.txt:

```
REQUIRES esp_netif
```

or

```
PRIV_REQUIRES esp_netif
```

Functions

esp_err_t **esp_netif_sntp_init** (const *esp_sntp_config_t* *config)

Initialize SNTP with supplied config struct.

参数 **config** -- Config struct
返回 ESP_OK on success

esp_err_t **esp_netif_sntp_start** (void)

Start SNTP service if it wasn't started during init (config.start = false) or restart it if already started.

返回 ESP_OK on success

void **esp_netif_sntp_deinit** (void)

Deinitialize esp_netif SNTP module.

esp_err_t **esp_netif_sntp_sync_wait** (TickType_t tout)

Wait for time sync event.

参数 **tout** -- Specified timeout in RTOS ticks
返回 ESP_TIMEOUT if sync event didn't came withing the timeout ESP_ERR_NOT_FINISHED if the sync event came, but we're in smooth update mode and still in progress (SNTP_SYNC_STATUS_IN_PROGRESS) ESP_OK if time sync'ed

esp_err_t **esp_netif_sntp_reachability** (unsigned int index, unsigned int *reachability)

Returns SNTP server's reachability shift register as described in RFC 5905.

参数

- **index** -- Index of the SERVER
- **reachability** -- reachability shift register

返回 ESP_OK on success, ESP_ERR_INVALID_STATE if SNTP not initialized ESP_ERR_INVALID_ARG if invalid arguments

Structures

struct **esp_sntp_config**

SNTP configuration struct.

Public Members

bool **smooth_sync**

set to true if smooth sync required

bool **server_from_dhcp**

set to true to request NTP server config from DHCP

bool **wait_for_sync**

if true, we create a semaphore to signal time sync event

bool **start**

set to true to automatically start the SNTP service

esp_sntp_time_cb_t **sync_cb**

optionally sets callback function on time sync event

bool **renew_servers_after_new_IP**

this is used to refresh server list if NTP provided by DHCP (which cleans other pre-configured servers)

ip_event_t **ip_event_to_renew**

set the IP event id on which we refresh server list (if renew_servers_after_new_IP=true)

size_t **index_of_first_server**

refresh server list after this server (if renew_servers_after_new_IP=true)

size_t **num_of_servers**

number of preconfigured NTP servers

const char ***servers**[1]

list of servers

Macros

ESP_SNTP_SERVER_LIST(...)

Utility macro for providing multiple servers in parentheses.

ESP_NETIF_SNTP_DEFAULT_CONFIG_MULTIPLE(servers_in_list, list_of_servers)

Default configuration to init SNTP with multiple servers.

参数

- **servers_in_list** -- Number of servers in the list
- **list_of_servers** -- List of servers (use *ESP_SNTP_SERVER_LIST(...)*)

ESP_NETIF_SNTP_DEFAULT_CONFIG(server)

Default configuration with a single server.

Type Definitions

typedef void (***esp_sntp_time_cb_t**)(struct timeval *tv)

Time sync notification function.

typedef struct *esp_sntp_config* **esp_sntp_config_t**

SNTP configuration struct.

Header File

- components/esp_netif/include/esp_netif_types.h
- This header file can be included with:

```
#include "esp_netif_types.h"
```

- This header file is a part of the API provided by the `esp_netif` component. To declare that your component depends on `esp_netif`, add the following to your CMakeLists.txt:

```
REQUIRES esp_netif
```

or

```
PRIV_REQUIRES esp_netif
```

Structures

struct **esp_netif_dns_info_t**

DNS server info.

Public Members

esp_ip_addr_t **ip**

IPV4 address of DNS server

struct **esp_netif_ip_info_t**

Event structure for IP_EVENT_STA_GOT_IP, IP_EVENT_ETH_GOT_IP events

Public Members

esp_ip4_addr_t **ip**

Interface IPV4 address

esp_ip4_addr_t **netmask**

Interface IPV4 netmask

esp_ip4_addr_t **gw**

Interface IPV4 gateway address

struct **esp_netif_ip6_info_t**

IPV6 IP address information.

Public Members

esp_ip6_addr_t **ip**

Interface IPV6 address

struct **ip_event_got_ip_t**

Event structure for IP_EVENT_GOT_IP event.

Public Members

esp_netif_t ***esp_netif**

Pointer to corresponding esp-netif object

esp_netif_ip_info_t **ip_info**

IP address, netmask, gatway IP address

bool **ip_changed**

Whether the assigned IP has changed or not

struct **ip_event_got_ip6_t**

Event structure for IP_EVENT_GOT_IP6 event

Public Members

esp_netif_t ***esp_netif**

Pointer to corresponding esp-netif object

esp_netif_ip6_info_t **ip6_info**

IPv6 address of the interface

int **ip_index**

IPv6 address index

struct **ip_event_add_ip6_t**

Event structure for ADD_IP6 event

Public Members

esp_ip6_addr_t **addr**

The address to be added to the interface

bool **preferred**

The default preference of the address

struct **ip_event_ap_staipassigned_t**

Event structure for IP_EVENT_AP_STAIPASSIGNED event

Public Members

esp_netif_t ***esp_netif**

Pointer to the associated netif handle

esp_ip4_addr_t **ip**

IP address which was assigned to the station

uint8_t **mac**[6]

MAC address of the connected client

struct **bridgeif_config**

LwIP bridge configuration

Public Members

uint16_t **max_fdb_dyn_entries**

maximum number of entries in dynamic forwarding database

uint16_t **max_fdb_sta_entries**

maximum number of entries in static forwarding database

uint8_t **max_ports**

maximum number of ports the bridge can consist of

struct **esp_netif_inherent_config**

ESP-netif inherent config parameters.

Public Members

esp_netif_flags_t **flags**

flags that define esp-netif behavior

uint8_t **mac**[6]

initial mac address for this interface

const *esp_netif_ip_info_t* ***ip_info**

initial ip address for this interface

uint32_t **get_ip_event**

event id to be raised when interface gets an IP

uint32_t **lost_ip_event**

event id to be raised when interface losts its IP

const char ***if_key**

string identifier of the interface

const char ***if_desc**

textual description of the interface

int **route_prio**

numeric priority of this interface to become a default routing if (if other netifs are up). A higher value of route_prio indicates a higher priority

bridgeif_config_t ***bridge_info***
LwIP bridge configuration

struct **esp_netif_driver_base_s**
ESP-netif driver base handle.

Public Members

esp_err_t (***post_attach**)(*esp_netif_t* *netif, *esp_netif_iodriver_handle* h)
post attach function pointer

esp_netif_t ***netif**
netif handle

struct **esp_netif_driver_ifconfig**
Specific IO driver configuration.

Public Members

esp_netif_iodriver_handle **handle**
io-driver handle

esp_err_t (***transmit**)(void *h, void *buffer, size_t len)
transmit function pointer

esp_err_t (***transmit_wrap**)(void *h, void *buffer, size_t len, void *netstack_buffer)
transmit wrap function pointer

void (***driver_free_rx_buffer**)(void *h, void *buffer)
free rx buffer function pointer

struct **esp_netif_config**
Generic esp_netif configuration.

Public Members

const *esp_netif_inherent_config_t* ***base**
base config

const *esp_netif_driver_ifconfig_t* ***driver**
driver config

const *esp_netif_netstack_config_t* ***stack**
stack config

struct **esp_netif_pair_mac_ip_t**
DHCP client's addr info (pair of MAC and IP address)

Public Members

uint8_t **mac**[6]

Clients MAC address

esp_ip4_addr_t **ip**

Clients IP address

Macros

ESP_ERR_ESP_NETIF_BASE

Definition of ESP-NETIF based errors.

ESP_ERR_ESP_NETIF_INVALID_PARAMS

ESP_ERR_ESP_NETIF_IF_NOT_READY

ESP_ERR_ESP_NETIF_DHCPC_START_FAILED

ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED

ESP_ERR_ESP_NETIF_NO_MEM

ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED

ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED

ESP_ERR_ESP_NETIF_INIT_FAILED

ESP_ERR_ESP_NETIF_DNS_NOT_CONFIGURED

ESP_ERR_ESP_NETIF_MLD6_FAILED

ESP_ERR_ESP_NETIF_IP6_ADDR_FAILED

ESP_ERR_ESP_NETIF_DHCPS_START_FAILED

ESP_NETIF_BR_FLOOD

Definition of ESP-NETIF bridge controll.

ESP_NETIF_BR_DROP

ESP_NETIF_BR_FDW_CPU

Type Definitions

typedef struct esp_netif_obj **esp_netif_t**

typedef enum *esp_netif_flags* **esp_netif_flags_t**

typedef enum *esp_netif_ip_event_type* **esp_netif_ip_event_type_t**

typedef struct *bridgeif_config* **bridgeif_config_t**

LwIP bridge configuration

typedef struct *esp_netif_inherent_config* **esp_netif_inherent_config_t**

ESP-netif inherent config parameters.

typedef struct *esp_netif_config* **esp_netif_config_t**

typedef void ***esp_netif_iodriver_handle**

IO driver handle type.

typedef struct *esp_netif_driver_base_s* **esp_netif_driver_base_t**

ESP-netif driver base handle.

typedef struct *esp_netif_driver_ifconfig* **esp_netif_driver_ifconfig_t**

typedef struct esp_netif_netstack_config **esp_netif_netstack_config_t**

Specific L3 network stack configuration.

typedef *esp_err_t* (***esp_netif_receive_t**)(*esp_netif_t* *esp_netif, void *buffer, size_t len, void *eb)

ESP-NETIF Receive function type.

Enumerations

enum **esp_netif_dns_type_t**

Type of DNS server.

Values:

enumerator **ESP_NETIF_DNS_MAIN**

DNS main server address

enumerator **ESP_NETIF_DNS_BACKUP**

DNS backup server address (Wi-Fi STA and Ethernet only)

enumerator **ESP_NETIF_DNS_FALLBACK**

DNS fallback server address (Wi-Fi STA and Ethernet only)

enumerator **ESP_NETIF_DNS_MAX**

enum **esp_netif_dhcp_status_t**

Status of DHCP client or DHCP server.

Values:

enumerator **ESP_NETIF_DHCP_INIT**

DHCP client/server is in initial state (not yet started)

enumerator **ESP_NETIF_DHCP_STARTED**

DHCP client/server has been started

enumerator **ESP_NETIF_DHCP_STOPPED**

DHCP client/server has been stopped

enumerator **ESP_NETIF_DHCP_STATUS_MAX**

enum **esp_netif_dhcp_option_mode_t**

Mode for DHCP client or DHCP server option functions.

Values:

enumerator **ESP_NETIF_OP_START**

enumerator **ESP_NETIF_OP_SET**

Set option

enumerator **ESP_NETIF_OP_GET**

Get option

enumerator **ESP_NETIF_OP_MAX**

enum **esp_netif_dhcp_option_id_t**

Supported options for DHCP client or DHCP server.

Values:

enumerator **ESP_NETIF_SUBNET_MASK**

Network mask

enumerator **ESP_NETIF_DOMAIN_NAME_SERVER**

Domain name server

enumerator **ESP_NETIF_ROUTER_SOLICITATION_ADDRESS**

Solicitation router address

enumerator **ESP_NETIF_REQUESTED_IP_ADDRESS**

Request specific IP address

enumerator **ESP_NETIF_IP_ADDRESS_LEASE_TIME**

Request IP address lease time

enumerator **ESP_NETIF_IP_REQUEST_RETRY_TIME**

Request IP address retry counter

enumerator **ESP_NETIF_VENDOR_CLASS_IDENTIFIER**

Vendor Class Identifier of a DHCP client

enumerator **ESP_NETIF_VENDOR_SPECIFIC_INFO**

Vendor Specific Information of a DHCP server

enum **ip_event_t**

IP event declarations

Values:

enumerator **IP_EVENT_STA_GOT_IP**

station got IP from connected AP

enumerator **IP_EVENT_STA_LOST_IP**

station lost IP and the IP is reset to 0

enumerator **IP_EVENT_AP_STAIPASSIGNED**

soft-AP assign an IP to a connected station

enumerator **IP_EVENT_GOT_IP6**

station or ap or ethernet interface v6IP addr is preferred

enumerator **IP_EVENT_ETH_GOT_IP**

ethernet got IP from connected AP

enumerator **IP_EVENT_ETH_LOST_IP**

ethernet lost IP and the IP is reset to 0

enumerator **IP_EVENT_PPP_GOT_IP**

PPP interface got IP

enumerator **IP_EVENT_PPP_LOST_IP**

PPP interface lost IP

enum **esp_netif_flags**

Values:

enumerator **ESP_NETIF_DHCP_CLIENT**

enumerator **ESP_NETIF_DHCP_SERVER**

enumerator **ESP_NETIF_FLAG_AUTOUP**

enumerator **ESP_NETIF_FLAG_GARP**

enumerator **ESP_NETIF_FLAG_EVENT_IP_MODIFIED**

enumerator **ESP_NETIF_FLAG_IS_PPP**

enumerator **ESP_NETIF_FLAG_IS_BRIDGE**

enumerator **ESP_NETIF_FLAG_MLDV6_REPORT**

enum **esp_netif_ip_event_type**

Values:

enumerator **ESP_NETIF_IP_EVENT_GOT_IP**

enumerator **ESP_NETIF_IP_EVENT_LOST_IP**

Header File

- components/esp_netif/include/esp_netif_ip_addr.h
- This header file can be included with:

```
#include "esp_netif_ip_addr.h"
```

- This header file is a part of the API provided by the `esp_netif` component. To declare that your component depends on `esp_netif`, add the following to your CMakeLists.txt:

```
REQUIRES esp_netif
```

or

```
PRIV_REQUIRES esp_netif
```

Functions

esp_ip6_addr_type_t **esp_netif_ip6_get_addr_type**(*esp_ip6_addr_t* *ip6_addr)

Get the IPv6 address type.

参数 **ip6_addr** -- **[in]** IPv6 type

返回 IPv6 type in form of enum esp_ip6_addr_type_t

static inline void **esp_netif_ip_addr_copy**(*esp_ip_addr_t* *dest, const *esp_ip_addr_t* *src)

Copy IP addresses.

参数

- **dest** -- **[out]** destination IP
- **src** -- **[in]** source IP

Structures

struct **esp_ip6_addr**

IPv6 address.

Public Members

uint32_t **addr**[4]

IPv6 address

uint8_t **zone**

zone ID

struct **esp_ip4_addr**

IPv4 address.

Public Members

uint32_t **addr**

IPv4 address

struct **_ip_addr**

IP address.

Public Members

esp_ip6_addr_t **ip6**

IPv6 address type

esp_ip4_addr_t **ip4**

IPv4 address type

union *_ip_addr*::[anonymous] **u_addr**

IP address union

uint8_t **type**

ipaddress type

Macros

esp_netif_htonl(x)

esp_netif_ip4_makeu32(a, b, c, d)

ESP_IP6_ADDR_BLOCK1(ip6addr)

ESP_IP6_ADDR_BLOCK2(ip6addr)

ESP_IP6_ADDR_BLOCK3(ip6addr)

ESP_IP6_ADDR_BLOCK4(ip6addr)

ESP_IP6_ADDR_BLOCK5(ip6addr)

ESP_IP6_ADDR_BLOCK6(ip6addr)

ESP_IP6_ADDR_BLOCK7(ip6addr)

ESP_IP6_ADDR_BLOCK8(ip6addr)

IPSTR

esp_ip4_addr_get_byte(ipaddr, idx)

esp_ip4_addr1(ipaddr)

esp_ip4_addr2(ipaddr)

esp_ip4_addr3(ipaddr)

esp_ip4_addr4(ipaddr)

esp_ip4_addr1_16(ipaddr)

esp_ip4_addr2_16(ipaddr)

esp_ip4_addr3_16(ipaddr)

esp_ip4_addr4_16(ipaddr)

IP2STR(ipaddr)

IPV6STR

IPV62STR(ipaddr)

ESP_IPADDR_TYPE_V4

ESP_IPADDR_TYPE_V6

ESP_IPADDR_TYPE_ANY

ESP_IP4TOUINT32(a, b, c, d)

ESP_IP4TOADDR(a, b, c, d)

ESP_IP4ADDR_INIT(a, b, c, d)

ESP_IP6ADDR_INIT(a, b, c, d)

IP4ADDR_STRLEN_MAX

ESP_IP_IS_ANY(addr)

Type Definitions

typedef struct *esp_ip4_addr* **esp_ip4_addr_t**

typedef struct *esp_ip6_addr* **esp_ip6_addr_t**

typedef struct *_ip_addr* **esp_ip_addr_t**

IP address.

Enumerations

enum **esp_ip6_addr_type_t**

Values:

enumerator **ESP_IP6_ADDR_IS_UNKNOWN**

enumerator **ESP_IP6_ADDR_IS_GLOBAL**

enumerator **ESP_IP6_ADDR_IS_LINK_LOCAL**

enumerator **ESP_IP6_ADDR_IS_SITE_LOCAL**

enumerator **ESP_IP6_ADDR_IS_UNIQUE_LOCAL**

enumerator **ESP_IP6_ADDR_IS_IPV4_MAPPED_IPV6**

Header File

- components/esp_netif/include/esp_vfs_l2tap.h
- This header file can be included with:

```
#include "esp_vfs_l2tap.h"
```

- This header file is a part of the API provided by the `esp_netif` component. To declare that your component depends on `esp_netif`, add the following to your CMakeLists.txt:

```
REQUIRES esp_netif
```

or

```
PRIV_REQUIRES esp_netif
```

Functions

esp_err_t **esp_vfs_l2tap_intf_register**(*l2tap_vfs_config_t* *config)

Add L2 TAP virtual filesystem driver.

This function must be called prior usage of ESP-NETIF L2 TAP Interface

参数 **config** -- L2 TAP virtual filesystem driver configuration. Default base path /dev/net/tap is used when this paramenter is NULL.

返回 esp_err_t

- ESP_OK on success

esp_err_t **esp_vfs_l2tap_intf_unregister**(const char *base_path)

Removes L2 TAP virtual filesystem driver.

参数 **base_path** -- Base path to the L2 TAP virtual filesystem driver. Default path /dev/net/tap is used when this paramenter is NULL.

返回 esp_err_t

- ESP_OK on success

esp_err_t **esp_vfs_l2tap_eth_filter**(*l2tap_iodriver_handle* driver_handle, void *buff, size_t *size)

Filters received Ethernet L2 frames into L2 TAP infrastructure.

参数

- **driver_handle** -- handle of driver at which the frame was received
- **buff** -- received L2 frame
- **size** -- input length of the L2 frame which is set to 0 when frame is filtered into L2 TAP

返回 esp_err_t

- ESP_OK is always returned

Structures

struct **l2tap_vfs_config_t**

L2Tap VFS config parameters.

Public Members

const char ***base_path**

vfs base path

Macros

L2TAP_VFS_DEFAULT_PATH

L2TAP_VFS_CONFIG_DEFAULT()

Type Definitions

typedef void ***l2tap_iodriver_handle**

Enumerations

enum **l2tap_ioctl_opt_t**

Values:

enumerator **L2TAP_S_RCV_FILTER**

enumerator **L2TAP_G_RCV_FILTER**

enumerator **L2TAP_S_INTF_DEVICE**

enumerator **L2TAP_G_INTF_DEVICE**

enumerator **L2TAP_S_DEVICE_DRV_HNDL**

enumerator **L2TAP_G_DEVICE_DRV_HNDL**

2.4.4 IP 网络层协议

ESP-NETIF 自定义 I/O 驱动程序

本节概述了如何配置具有 ESP-NETIF 连接功能的新 I/O 驱动程序。

通常情况下，I/O 驱动程序须注册为 ESP-NETIF 驱动程序。因此，它依赖于 ESP-NETIF 组件，并负责提供数据路径函数、后附回调函数，并在多数情况下用于设置默认事件处理程序，根据驱动程序的生命周期转换来定义网络接口操作。

数据包 Input/Output 根据*ESP-NETIF 架构* 章节提供的图表可以看出，须定义以下三个数据路径函数 API 以连接 ESP-NETIF：

- *esp_netif_transmit()*
- *esp_netif_free_rx_buffer()*
- *esp_netif_receive()*

前两个函数可以传输和释放 RX 缓冲区，用作回调。它们由 ESP-NETIF（及其底层 TCP/IP 堆栈）调用，并由 I/O 驱动实现。

另一方面，接收函数由 I/O 驱动程序调用，因此驱动的代码只需在接收到新数据时调用*esp_netif_receive()* 函数。

后附回调 网络接口初始化的最后一步是调用以下 API，将 ESP-NETIF 实例附加到 I/O 驱动程序上：

```
esp_err_t esp_netif_attach(esp_netif_t *esp_netif, esp_netif_iodriver_handle␣
↪driver_handle);
```

假设 esp_netif_iodriver_handle 是指向驱动程序对象的指针，该对象是从 struct esp_netif_driver_base_s 衍生的结构体，那么 I/O 驱动结构体的第一个成员必须是此基础结构，并指向：

- 后附函数回调
- 相关的 ESP-NETIF 实例

因此，I/O 驱动程序须创建以下结构体的实例：

```
typedef struct my_netif_driver_s {
        esp_netif_driver_base_t base;           /*!< 保留基本结构体作为 esp-netif␣
↪驱动 */
        driver_impl             *h;             /*!< 驱动实现 */
    } my_netif_driver_t;
```

此实例中包含 my_netif_driver_t::base.post_attach 的真实值和实际的驱动处理程序 my_netif_driver_t::h。

从初始化代码调用*esp_netif_attach()* 时，将执行 I/O 驱动程序代码的后附回调，以在 ESP-NETIF 和 I/O 驱动程序实例之间相互注册回调。通常，后附回调中也会启动驱动程序。以下为一个简单的后附回调示例：

```
static esp_err_t my_post_attach_start(esp_netif_t * esp_netif, void * args)
{
    my_netif_driver_t *driver = args;
    const esp_netif_driver_ifconfig_t driver_ifconfig = {
            .driver_free_rx_buffer = my_free_rx_buf,
            .transmit = my_transmit,
            .handle = driver->driver_impl
    };
    driver->base.netif = esp_netif;
    ESP_ERROR_CHECK(esp_netif_set_driver_config(esp_netif, &driver_ifconfig));
    my_driver_start(driver->driver_impl);
    return ESP_OK;
}
```

默认处理程序 I/O 驱动程序通常还会根据 I/O 驱动程序的状态转换，为相关网络接口的生命周期行为提供默认定义，例如 *driver start -> network start* 等。

以下是此类默认处理程序的一个示例：

```
esp_err_t my_driver_netif_set_default_handlers(my_netif_driver_t *driver, esp_
↪netif_t * esp_netif)
{
```

(下页继续)

(续上页)

```
    driver_set_event_handler(driver->driver_impl, esp_netif_action_start, MY_DRV_
↪EVENT_START, esp_netif);
    driver_set_event_handler(driver->driver_impl, esp_netif_action_stop, MY_DRV_
↪EVENT_STOP, esp_netif);
    return ESP_OK;
}
```

网络堆栈连接 用于传输和释放 RX 缓冲区的数据路径函数（在 I/O 驱动中定义）由 ESP-NETIF 的 TCP/IP 堆栈连接层调用。

注意，ESP-IDF 为最常见的网络接口（如 Wi-Fi station 或以太网）提供了几种网络堆栈配置。这些配置定义在 esp_netif/include/esp_netif_defaults.h 中，能够满足大多数网络驱动程序的需求。在少数情况下，一些专家用户可能希望自定义基于 lwIP 的接口层，这需要额外设置 lwIP 依赖。

以下参考 API 概述了这些网络堆栈和 ESP-NETIF 的交互：

Header File

- components/esp_netif/include/esp_netif_net_stack.h
- This header file can be included with:

```
#include "esp_netif_net_stack.h"
```

- This header file is a part of the API provided by the `esp_netif` component. To declare that your component depends on `esp_netif`, add the following to your CMakeLists.txt:

```
REQUIRES esp_netif
```

or

```
PRIV_REQUIRES esp_netif
```

Functions

esp_netif_t ***esp_netif_get_handle_from_netif_impl*** (void *dev)

Returns esp-netif handle.

参数 **dev** -- **[in]** opaque ptr to network interface of specific TCP/IP stack
返回 handle to related esp-netif instance

void ***esp_netif_get_netif_impl*** (*esp_netif_t* *esp_netif)

Returns network stack specific implementation handle.

参数 **esp_netif** -- **[in]** Handle to esp-netif instance
返回 handle to related network stack netif handle

esp_err_t **esp_netif_set_link_speed** (*esp_netif_t* *esp_netif, uint32_t speed)

Set link-speed for the specified network interface.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **speed** -- **[in]** Link speed in bit/s

返回 ESP_OK on success

esp_err_t **esp_netif_transmit** (*esp_netif_t* *esp_netif, void *data, size_t len)

Outputs packets from the TCP/IP stack to the media to be transmitted.

This function gets called from network stack to output packets to IO driver.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **data** -- **[in]** Data to be transmitted

- **len** -- **[in]** Length of the data frame

返回 ESP_OK on success, an error passed from the I/O driver otherwise

esp_err_t **esp_netif_transmit_wrap**(*esp_netif_t* *esp_netif, void *data, size_t len, void *netstack_buf)

Outputs packets from the TCP/IP stack to the media to be transmitted.

This function gets called from network stack to output packets to IO driver.

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **data** -- **[in]** Data to be transmitted
- **len** -- **[in]** Length of the data frame
- **netstack_buf** -- **[in]** net stack buffer

返回 ESP_OK on success, an error passed from the I/O driver otherwise

void **esp_netif_free_rx_buffer**(void *esp_netif, void *buffer)

Free the rx buffer allocated by the media driver.

This function gets called from network stack when the rx buffer to be freed in IO driver context, i.e. to deallocate a buffer owned by io driver (when data packets were passed to higher levels to avoid copying)

参数

- **esp_netif** -- **[in]** Handle to esp-netif instance
- **buffer** -- **[in]** Rx buffer pointer

TCP/IP 套接字 API 的示例代码存放在 ESP-IDF 示例项目的 protocols/sockets 目录下。

2.4.5 应用层协议

应用层网络协议（IP 网络层协议之上）的相关文档存放在应用层协议 目录下。

2.5 外设 API

警告： This document is not updated for ESP32P4 yet, so some of the content may not be correct.

This warning was automatically inserted due to the source file being in the *add_warnings_pages* list.

2.5.1 模数转换器 (ADC) 单次转换模式驱动

简介

模数转换器集成于芯片，支持测量特定模拟 IO 管脚的模拟信号。

ESP32-P4 有两个 ADC 单元，可以在以下场景使用：

- 生成 ADC 单次转换结果
- 生成连续 ADC 转换结果

本指南介绍了 ADC 单次转换模式。

功能概述

下文将分节概述安装和运行 ADC 的基本步骤：

- *资源分配* - 介绍获取 ADC 句柄所需设置的参数，以及如何在 ADC 完成工作后回收资源。
- *配置 ADC 单元实例* - 介绍配置 ADC 单元所需设置的参数，用于获取 ADC 转换的原始结果。
- *读取转换结果* - 介绍如何获取 ADC 转换的原始结果。
- *硬件限制* - 介绍与 ADC 相关的硬件限制。
- *电源管理* - 介绍电源管理的相关内容。
- *IRAM 安全* - 介绍在禁用 cache 时，如何读取 ADC 转换的原始结果。
- *线程安全* - 介绍由驱动程序认证为线程安全的 API。
- *Kconfig 选项* - 介绍支持的 Kconfig 选项，不同选项对驱动程序的操作会产生不同影响。

资源分配 ADC 单次转换模式驱动基于 ESP32-P4 SAR ADC 模块实现，不同的 ESP 芯片可能拥有不同数量的独立 ADC。对于单次转换模式驱动而言，ADC 实例以*adc_oneshot_unit_handle_t* 表示。

请设置所需的初始配置结构体*adc_oneshot_unit_init_cfg_t* 安装 ADC 实例，具体如下：

- *adc_oneshot_unit_init_cfg_t::unit_id* 选择 ADC。请参阅 技术规格书，了解对应 ADC 的专用模拟 IO 管脚。
- *adc_oneshot_unit_init_cfg_t::clk_src* 选择 ADC 的时钟源。设置为 0 时，驱动程序将使用默认时钟源，详情请参阅*adc_oneshot_clk_src_t*。
- *adc_oneshot_unit_init_cfg_t::ulp_mode* 设置是否支持 ADC 在 ULP 模式下工作。

完成 ADC 初始配置后，使用已设置的初始配置结构体*adc_oneshot_unit_init_cfg_t* 调用*adc_oneshot_new_unit()*。如果分配成功，该函数将返回 ADC 单元实例句柄。

该函数可能因参数无效、内存不足等原因返回错误代码。比如，当要分配的 ADC 实例已经注册时，该函数会返回*ESP_ERR_NOT_FOUND* 错误。可用 ADC 数量可通过*SOC_ADC_PERIPH_NUM* 查看。

如果不再需要先前创建的 ADC 单元实例，请调用*adc_oneshot_del_unit()* 回收该实例，相关的硬件和软件资源也会回收。

在普通单次转换模式下创建 ADC 单元实例句柄

```
adc_oneshot_unit_handle_t adc1_handle;
adc_oneshot_unit_init_cfg_t init_config1 = {
    .unit_id = ADC_UNIT_1,
    .ulp_mode = ADC_ULP_MODE_DISABLE,
};
ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config1, &adc1_handle));
```

回收 ADC 单元实例

```
ESP_ERROR_CHECK(adc_oneshot_del_unit(adc1_handle));
```

配置 ADC 单元实例 创建 ADC 单元实例后，请设置*adc_oneshot_chan_cfg_t* 配置 ADC IO 以测量模拟信号，具体如下：

- *adc_oneshot_chan_cfg_t::atten*，ADC 衰减。请参阅 技术参考手册 > 片上传感器与模拟信号处理。
- *adc_oneshot_chan_cfg_t::bitwidth*，原始转换结果的位宽。

备注： ADC IO 及其对应的 ADC 通道编号，请参阅 技术规格书。

此外，可以使用*adc_continuous_io_to_channel()* 和*adc_continuous_channel_to_io()* 了解 ADC 通道和 ADC IO。

为使以上设置生效，请使用上述配置结构体调用*adc_oneshot_config_channel()*，并指定要配置的 ADC 通道。函数*adc_oneshot_config_channel()* 支持多次调用，以配置不同的 ADC 通道。驱动程序将在内部保存每个通道的配置。

配置两个 ADC 通道

```
adc_oneshot_chan_cfg_t config = {
    .bitwidth = ADC_BITWIDTH_DEFAULT,
    .atten = ADC_ATTEN_DB_12,
};
ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, EXAMPLE_ADC1_CHAN0, &
↪config));
ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle, EXAMPLE_ADC1_CHAN1, &
↪config));
```

读取转换结果 完成上述配置后，ADC 即可测量来自配置好的 ADC 通道的模拟信号。调用*adc_oneshot_read()* 可以获取 ADC 通道的原始转换结果。

- *adc_oneshot_read()* 可安全使用。ADC 由其他驱动程序/外设共享，请参阅*硬件限制*。函数*adc_oneshot_read()* 使用互斥锁，避免与其他函数同时使用硬件，因此该函数不应在 ISR 上下文中使用。当 ADC 由其他驱动程序/外设占用时，该函数可能出错，并返回*ESP_ERR_TIMEOUT* 错误。此时，ADC 原始结果无效。

该函数可能因参数无效而调用失败。

通过该函数获取的 ADC 转换结果为原始数据。可以使用以下公式，根据 ADC 原始结果计算电压：

```
Vout = Dout * Vmax / Dmax        (1)
```

其中：

Vout	**数字输出结果，代表电压。**
Dout	ADC 原始数字读取结果。
Vmax	可测量的最大模拟输入电压，与 ADC 衰减相关，请参考 技术参考手册 > 片上传感器与模拟信号处理。
Dmax	输出 ADC 原始数字读取结果的最大值，即 2^ 位宽，位宽即之前配置的*adc_oneshot_chan_cfg_t::bitwidth*。

若需进一步校准，将 ADC 原始结果转换为以 mV 为单位的电压数据，请参考校准文档*模数转换器 (ADC) 校准驱动程序*。

读取原始结果

```
ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, EXAMPLE_ADC1_CHAN0, &adc_raw[0][0]));
ESP_LOGI(TAG, "ADC%d Channel[%d] Raw Data: %d", ADC_UNIT_1 + 1, EXAMPLE_ADC1_CHAN0,
↪ adc_raw[0][0]);

ESP_ERROR_CHECK(adc_oneshot_read(adc1_handle, EXAMPLE_ADC1_CHAN1, &adc_raw[0][1]));
ESP_LOGI(TAG, "ADC%d Channel[%d] Raw Data: %d", ADC_UNIT_1 + 1, EXAMPLE_ADC1_CHAN1,
↪ adc_raw[0][1]);
```

硬件限制

- 随机数生成器 (RNG) 以 ADC 为输入源。使用 ADC 单次转换模式驱动从 RNG 生成随机数时，随机性会减弱。
- 一个 ADC 单元每次只能在一种操作模式下运行，可以是连续模式或单次模式。`adc_oneshot_start()` 提供了保护措施。

电源管理 启用电源管理，即启用*CONFIG_PM_ENABLE* 时，系统在空闲状态下可能会调整系统时钟频率。然而，ADC 单次转换模式驱动以轮询例程运行，`adc_oneshot_read()` 会不断检查 CPU 是否完成读取，直到函数返回。在此期间，ADC 单次转换模式驱动程序所在的任务不会受阻塞。因此，在读取时时钟频率保持稳定。

IRAM 安全 flash 写入/擦除、OTA 等原因都可能导致 cache 禁用，此时，默认不应运行任何 ADC 单次转换模式驱动 API。如果在禁用 cache 时执行了 ADC 单次转换模式驱动 API，可能会出现类似 `Illegal Instruction` 或 `Load/Store Prohibited` 的错误。

线程安全

- `adc_oneshot_new_unit()`
- `adc_oneshot_config_channel()`
- `adc_oneshot_read()`

上述函数均为线程安全，使用时，可以直接从不同的 RTOS 任务中调用以上函数，无需额外锁保护。

- `adc_oneshot_del_unit()` 非线程安全。此外，与上文中线程安全的函数一起调用该函数时，可能导致线程安全函数的调用出错。

Kconfig 选项

- *CONFIG_ADC_ONESHOT_CTRL_FUNC_IN_IRAM* 决定了放置 ADC 快速读取函数的位置，即 IRAM 或 flash 中，详情请参阅*IRAM 安全*。

应用示例

- ADC 单次转换模式示例：peripherals/adc/oneshot_read。

API 参考

Header File

- components/hal/include/hal/adc_types.h
- This header file can be included with:

```
#include "hal/adc_types.h"
```

Structures

struct **adc_digi_pattern_config_t**

ADC digital controller pattern configuration.

Public Members

uint8_t **atten**

Attenuation of this ADC channel.

uint8_t **channel**

ADC channel.

uint8_t **unit**

ADC unit.

uint8_t **bit_width**

ADC output bit width.

struct **adc_digi_output_data_t**

ADC digital controller (DMA mode) output data format. Used to analyze the acquired ADC (DMA) data.

Public Members

uint32_t **data**

ADC real output data info. Resolution: 12 bit.

uint32_t **reserved12**

Reserved12.

uint32_t **channel**

ADC channel index info. If (channel < ADC_CHANNEL_MAX), The data is valid. If (channel > ADC_CHANNEL_MAX), The data is invalid.

uint32_t **unit**

ADC unit index info. 0: ADC1; 1: ADC2.

uint32_t **reserved17_31**

Reserved17.

struct *adc_digi_output_data_t*::[anonymous]::[anonymous] **type2**

When the configured output format is 12bit.

uint32_t **val**

Raw data value

Type Definitions

typedef *soc_periph_adc_digi_clk_src_t* **adc_oneshot_clk_src_t**

Clock source type of oneshot mode which uses digital controller.

typedef *soc_periph_adc_digi_clk_src_t* **adc_continuous_clk_src_t**

Clock source type of continuous mode which uses digital controller.

Enumerations

enum **adc_unit_t**

ADC unit.

Values:

enumerator **ADC_UNIT_1**

SAR ADC 1.

enumerator **ADC_UNIT_2**

SAR ADC 2.

enum **adc_channel_t**

ADC channels.

Values:

enumerator **ADC_CHANNEL_0**

ADC channel.

enumerator **ADC_CHANNEL_1**

ADC channel.

enumerator **ADC_CHANNEL_2**

ADC channel.

enumerator **ADC_CHANNEL_3**

ADC channel.

enumerator **ADC_CHANNEL_4**

ADC channel.

enumerator **ADC_CHANNEL_5**

ADC channel.

enumerator **ADC_CHANNEL_6**

ADC channel.

enumerator **ADC_CHANNEL_7**

ADC channel.

enumerator **ADC_CHANNEL_8**

ADC channel.

enumerator **ADC_CHANNEL_9**

ADC channel.

enum **adc_atten_t**

ADC attenuation parameter. Different parameters determine the range of the ADC.

Values:

enumerator **ADC_ATTEN_DB_0**

No input attenuation, ADC can measure up to approx.

enumerator **ADC_ATTEN_DB_2_5**

The input voltage of ADC will be attenuated extending the range of measurement by about 2.5 dB.

enumerator **ADC_ATTEN_DB_6**

The input voltage of ADC will be attenuated extending the range of measurement by about 6 dB.

enumerator **ADC_ATTEN_DB_12**

The input voltage of ADC will be attenuated extending the range of measurement by about 12 dB.

enumerator **ADC_ATTEN_DB_11**

This is deprecated, it behaves the same as `ADC_ATTEN_DB_12`

enum **adc_bitwidth_t**

Values:

enumerator **ADC_BITWIDTH_DEFAULT**

Default ADC output bits, max supported width will be selected.

enumerator **ADC_BITWIDTH_9**

ADC output width is 9Bit.

enumerator **ADC_BITWIDTH_10**

ADC output width is 10Bit.

enumerator **ADC_BITWIDTH_11**

ADC output width is 11Bit.

enumerator **ADC_BITWIDTH_12**

ADC output width is 12Bit.

enumerator **ADC_BITWIDTH_13**

ADC output width is 13Bit.

enum **adc_ulp_mode_t**

Values:

enumerator **ADC_ULP_MODE_DISABLE**

ADC ULP mode is disabled.

enumerator **ADC_ULP_MODE_FSM**

ADC is controlled by ULP FSM.

enumerator **ADC_ULP_MODE_RISCV**

ADC is controlled by ULP RISCV.

enum **adc_digi_convert_mode_t**

ADC digital controller (DMA mode) work mode.

Values:

enumerator **ADC_CONV_SINGLE_UNIT_1**

Only use ADC1 for conversion.

enumerator **ADC_CONV_SINGLE_UNIT_2**

Only use ADC2 for conversion.

enumerator **ADC_CONV_BOTH_UNIT**

Use Both ADC1 and ADC2 for conversion simultaneously.

enumerator **ADC_CONV_ALTER_UNIT**

Use both ADC1 and ADC2 for conversion by turn. e.g. ADC1 -> ADC2 -> ADC1 -> ADC2 .....

enum **adc_digi_output_format_t**

ADC digital controller (DMA mode) output data format option.

Values:

enumerator **ADC_DIGI_OUTPUT_FORMAT_TYPE1**

See `adc_digi_output_data_t.type1`

enumerator **ADC_DIGI_OUTPUT_FORMAT_TYPE2**

See *adc_digi_output_data_t.type2*

enum **adc_digi_iir_filter_t**

ADC IIR Filter ID.

Values:

enumerator **ADC_DIGI_IIR_FILTER_0**

Filter 0.

enumerator **ADC_DIGI_IIR_FILTER_1**

Filter 1.

enum **adc_digi_iir_filter_coeff_t**

IIR Filter Coefficient.

Values:

enumerator **ADC_DIGI_IIR_FILTER_COEFF_2**

The filter coefficient is 2.

enumerator **ADC_DIGI_IIR_FILTER_COEFF_4**

The filter coefficient is 4.

enumerator **ADC_DIGI_IIR_FILTER_COEFF_8**

The filter coefficient is 8.

enumerator **ADC_DIGI_IIR_FILTER_COEFF_16**

The filter coefficient is 16.

enumerator **ADC_DIGI_IIR_FILTER_COEFF_64**

The filter coefficient is 64.

enum **adc_monitor_id_t**

ADC monitor (continuous mode) ID.

Values:

enumerator **ADC_MONITOR_0**

The monitor index 0.

enumerator **ADC_MONITOR_1**

The monitor index 1.

enum **adc_monitor_mode_t**

Monitor config/event mode type.

Values:

enumerator **ADC_MONITOR_MODE_HIGH**

ADC raw_result > threshold value, monitor interrupt will be generated.

enumerator **ADC_MONITOR_MODE_LOW**

ADC raw_result < threshold value, monitor interrupt will be generated.

Header File

- components/esp_adc/include/esp_adc/adc_oneshot.h
- This header file can be included with:

```
#include "esp_adc/adc_oneshot.h"
```

- This header file is a part of the API provided by the `esp_adc` component. To declare that your component depends on `esp_adc`, add the following to your CMakeLists.txt:

```
REQUIRES esp_adc
```

or

```
PRIV_REQUIRES esp_adc
```

Functions

esp_err_t **adc_oneshot_new_unit**(const *adc_oneshot_unit_init_cfg_t* *init_config, *adc_oneshot_unit_handle_t* *ret_unit)

Create a handle to a specific ADC unit.

备注： This API is thread-safe. For more details, see ADC programming guide

---

参数

- **init_config** -- **[in]** Driver initial configurations
- **ret_unit** -- **[out]** ADC unit handle

返回

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_NO_MEM: No memory
- ESP_ERR_NOT_FOUND: The ADC peripheral to be claimed is already in use
- ESP_FAIL: Clock source isn't initialised correctly

esp_err_t **adc_oneshot_config_channel**(*adc_oneshot_unit_handle_t* handle, *adc_channel_t* channel, const *adc_oneshot_chan_cfg_t* *config)

Set ADC oneshot mode required configurations.

备注： This API is thread-safe. For more details, see ADC programming guide

参数
- **handle** -- **[in]** ADC handle
- **channel** -- **[in]** ADC channel to be configured
- **config** -- **[in]** ADC configurations

返回
- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments

esp_err_t **adc_oneshot_read**(*adc_oneshot_unit_handle_t* handle, *adc_channel_t* chan, int *out_raw)

Get one ADC conversion raw result.

备注： This API is thread-safe. For more details, see ADC programming guide

备注： This API should NOT be called in an ISR context

参数
- **handle** -- **[in]** ADC handle
- **chan** -- **[in]** ADC channel
- **out_raw** -- **[out]** ADC conversion raw result

返回
- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_TIMEOUT: Timeout, the ADC result is invalid

esp_err_t **adc_oneshot_del_unit**(*adc_oneshot_unit_handle_t* handle)

Delete the ADC unit handle.

备注： This API is thread-safe. For more details, see ADC programming guide

参数 **handle** -- **[in]** ADC handle

返回
- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_NOT_FOUND: The ADC peripheral to be disclaimed isn't in use

esp_err_t **adc_oneshot_io_to_channel**(int io_num, *adc_unit_t* *const unit_id, *adc_channel_t* *const channel)

Get ADC channel from the given GPIO number.

参数
- **io_num** -- **[in]** GPIO number
- **unit_id** -- **[out]** ADC unit
- **channel** -- **[out]** ADC channel

返回
- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FOUND: The IO is not a valid ADC pad

esp_err_t **adc_oneshot_channel_to_io**(*adc_unit_t* unit_id, *adc_channel_t* channel, int *const io_num)

Get GPIO number from the given ADC channel.

参数

- **unit_id** -- **[in]** ADC unit
- **channel** -- **[in]** ADC channel
- **io_num** -- **[out]** GPIO number
- **–** -- ESP_OK: On success
  - ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **adc_oneshot_get_calibrated_result**(*adc_oneshot_unit_handle_t* handle, *adc_cali_handle_t* cali_handle, *adc_channel_t* chan, int *cali_result)

Convenience function to get ADC calibrated result.

This is an all-in-one function which does:

- oneshot read ADC raw result
- calibrate the raw result and convert it into calibrated result (in mV)

参数

- **handle** -- **[in]** ADC oneshot handle, you should call adc_oneshot_new_unit() to get this handle
- **cali_handle** -- **[in]** ADC calibration handle, you should call adc_cali_create_scheme_x() in adc_cali_scheme.h to create a handle
- **chan** -- **[in]** ADC channel
- **cali_result** -- **[out]** Calibrated ADC result (in mV)

返回

- ESP_OK Other return errors from adc_oneshot_read() and adc_cali_raw_to_voltage()

Structures

struct **adc_oneshot_unit_init_cfg_t**

ADC oneshot driver initial configurations.

Public Members

adc_unit_t **unit_id**

ADC unit.

adc_oneshot_clk_src_t **clk_src**

Clock source.

adc_ulp_mode_t **ulp_mode**

ADC controlled by ULP, see `adc_ulp_mode_t`

struct **adc_oneshot_chan_cfg_t**

ADC channel configurations.

Public Members

adc_atten_t **atten**

ADC attenuation.

adc_bitwidth_t **bitwidth**

ADC conversion result bits.

Type Definitions

typedef struct adc_oneshot_unit_ctx_t ***adc_oneshot_unit_handle_t**

Type of ADC unit handle for oneshot mode.

> **警告：** This document is not updated for ESP32P4 yet, so some of the content may not be correct.
>
> This warning was automatically inserted due to the source file being in the *add_warnings_pages* list.

2.5.2 模数转换器 (ADC) 连续转换模式驱动

简介

ESP32-P4 芯片集成了模数转换器 (ADC)，支持测量特定模拟 IO 管脚的模拟信号。此外，ADC 还支持直接内存访问 (DMA) 功能，高效获取 ADC 转换结果。

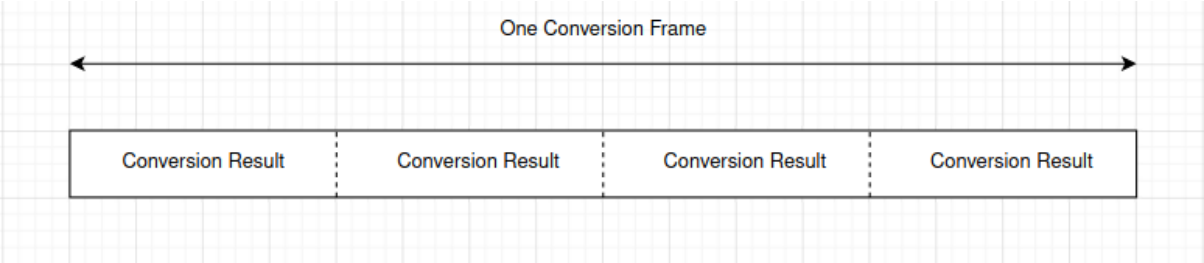
ESP32-P4 具有两个 ADC 单元，可应用于以下场景：

- 生成单次 ADC 转换结果
- 生成连续 ADC 转换结果

本指南介绍了 ADC 连续转换模式。

ADC 连续转换模式驱动概念 ADC 连续转换模式驱动由多个转换帧组成。

- 转换帧：一个转换帧包含多个转换结果。转换帧大小以字节为单位，在`adc_continuous_new_handle()` 中配置。
- 转换结果：一个转换结果包含多个字节，即`SOC_ADC_DIGI_RESULT_BYTES`。转换结果的数据结构由`adc_digi_output_data_t` 定义，包括 ADC 单元、ADC 通道以及原始数据。



功能概述

下文将分节概述安装 ADC 连续转换模式驱动、并从一组 ADC 通道连续读取 ADC 转换结果的基本步骤：

- 资源分配：介绍初始化 ADC 连续转换模式驱动所需设置的参数，以及如何将驱动去初始化。
- 配置 *ADC*：介绍如何将 ADC 配置为在连续转换模式下工作。
- *ADC* 控制：介绍 ADC 控制函数。
- 注册事件回调：介绍如何将特定用户代码链接到 ADC 连续转换模式事件回调函数。
- 读取转换结果：介绍如何获取 ADC 转换结果。
- 硬件限制：介绍与 ADC 相关的硬件限制。
- 电源管理：介绍电源管理的相关内容。

- *IRAM 安全*：介绍与 IRAM 安全相关的函数。
- 线程安全：介绍由驱动程序认证为线程安全的 API。

资源分配 ADC 连续转换模式驱动基于 ESP32-P4 SAR ADC 模块实现，不同的 ESP 目标芯片可能拥有不同数量的独立 ADC。

请按照以下步骤设置配置结构体*adc_continuous_handle_cfg_t*，创建 ADC 连续转换模式驱动的句柄：

- *adc_continuous_handle_cfg_t::max_store_buf_size*：以字节为单位设置最大缓冲池的大小，驱动程序将 ADC 转换结果保存到该缓冲池中。缓冲池已满时，新的转换将丢失。
- *adc_continuous_handle_cfg_t::conv_frame_size*：以字节为单位设置 ADC 转换帧大小。
- *adc_continuous_handle_cfg_t::flags*：设置可以改变驱动程序行为的标志。
  - flush_pool：缓冲池满时自动清空缓冲池。

完成以上 ADC 配置后，使用已设置的配置结构体*adc_continuous_handle_cfg_t* 调用*adc_continuous_new_handle()*。该函数可能将在特定情况下返回错误值，如无效参数、内存不足等。

函数返回*ESP_ERR_NOT_FOUND* 时，表明 GDMA 空闲通道不足。

如果不再使用 ADC 连续转换模式驱动，请调用*adc_continuous_deinit()* 将驱动去初始化。

初始化 ADC 连续转换模式驱动

```
adc_continuous_handle_cfg_t adc_config = {
    .max_store_buf_size = 1024,
    .conv_frame_size = 100,
};
ESP_ERROR_CHECK(adc_continuous_new_handle(&adc_config));
```

回收 ADC 单元

```
ESP_ERROR_CHECK(adc_continuous_deinit());
```

配置 ADC 初始化 ADC 连续转换模式驱动后，设置*adc_continuous_config_t* 配置 ADC IO，测量模拟信号：

- *adc_continuous_config_t::pattern_num*：要使用的 ADC 通道数量。
- *adc_continuous_config_t::adc_pattern*：每个要使用的 ADC 通道的配置列表，请参阅下文描述。
- *adc_continuous_config_t::sample_freq_hz*：期望的 ADC 采样频率，单位为 Hz。
- *adc_continuous_config_t::conv_mode*：连续转换模式。
- *adc_continuous_config_t::format*：转换模式结果的输出格式。

按照以下步骤设置*adc_digi_pattern_config_t*：

- *adc_digi_pattern_config_t::atten*：ADC 衰减。请参阅 技术参考手册 中的片上传感器与模拟信号处理章节。
- *adc_digi_pattern_config_t::channel*：IO 对应的 ADC 通道号，请参阅下文注意事项。
- *adc_digi_pattern_config_t::unit*：IO 所属的 ADC 单元。
- *adc_digi_pattern_config_t::bit_width*：原始转换结果的位宽。

备注： 对于 IO 对应的 ADC 通道号，请参阅 技术参考手册 获取 ADC IO 管脚的详细信息。另外，可以使用*adc_continuous_io_to_channel()* 和*adc_continuous_channel_to_io()* 获取 ADC 通道和 ADC IO 的对应关系。

为使这些设置生效，请使用上述配置结构体，调用*adc_continuous_config()*。此 API 可能由于*ESP_ERR_INVALID_ARG* 等原因返回错误。当它返回*ESP_ERR_INVALID_STATE* 时，意味着 ADC 连续转换模式驱动已经启动，此时不应调用此 API。

请参考 ADC 连续转换模式示例 peripherals/adc/continuous_read，查看相应配置代码。

ADC 控制

启动和停止 调用*adc_continuous_start()*，将使 ADC 开始从配置好的 ADC 通道测量模拟信号，并生成转换结果。

相反，调用*adc_continuous_stop()* 则会停止 ADC 转换。

```
ESP_ERROR_CHECK(adc_continuous_stop());
```

注册事件回调 调用*adc_continuous_register_event_callbacks()*，可以将自己的函数链接到驱动程序的 ISR 中。通过*adc_continuous_evt_cbs_t* 可查看所有支持的事件回调。

- *adc_continuous_evt_cbs_t::on_conv_done*：当一个转换帧完成时，触发此事件。
- *adc_continuous_evt_cbs_t::on_pool_ovf*：当内部缓冲池已满时，触发此事件，新的转换结果将丢失。

由于上述回调函数在 ISR 中调用，请确保回调函数适合在 ISR 上下文中运行，且这些回调不应涉及阻塞逻辑。回调函数的原型在*adc_continuous_callback_t* 中声明。

在调用*adc_continuous_register_event_callbacks()* 时，还可以通过参数 `user_data` 注册自己的上下文，该用户数据将直接传递给回调函数。

此回调函数可能由于*ESP_ERR_INVALID_ARG* 等原因返回错误。启用*CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE* 时，如果回调函数失败并报错，可能是因为回调函数不在内部 RAM 中，请查看错误日志了解详情。此外，如果回调函数出现*ESP_ERR_INVALID_STATE* 错误，表明 ADC 连续转换模式驱动已经启动，此时不应添加回调。

转换完成事件 当驱动程序完成一次转换后，会触发*adc_continuous_evt_cbs_t::on_conv_done* 事件，并填充事件数据。事件数据包含一个指向转换帧缓冲区的指针，以及转换帧缓冲区大小。要了解事件数据结构，请参阅*adc_continuous_evt_data_t*。

备注： 注意，数据缓冲区*adc_continuous_evt_data_t::conv_frame_buffer* 由驱动程序本身维护，请勿释放此内存。

备注： 启用 Kconfig 选项*CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE* 时，注册的回调函数以及回调函数中调用的函数应放置在 IRAM 中，涉及的变量也应放置在内部 RAM 中。

缓冲池溢出事件 ADC 连续转换模式驱动使用内部缓冲池保存转换结果，缓冲池满时将发生缓冲池溢出事件。此时，驱动程序不会继续填充事件数据。缓冲池溢出通常是因为调用*adc_continuous_read()* 从池中读取数据的速度远低于 ADC 转换的速度。

读取转换结果 调用*adc_continuous_start()* 启动 ADC 连续转换，调用*adc_continuous_read()* 可以获取 ADC 通道的转换结果。注意提供缓冲区，获取原始结果。

函数*adc_continuous_read()* 每次都会尝试以期望长度读取转换结果。

- 调用*adc_continuous_read()* 可以请求读取指定长度的转换结果。但有时实际可用的转换结果可能少于请求长度，此时，函数仍会将数据从内部池移动到你提供的缓冲区中。因此，请查看 out_length 的值，了解实际移动到缓冲区中的转换结果数量。
- 如果内部池中没有生成转换结果，函数将会阻塞一段时间，即 timeout_ms，直到转换结果生成。如果始终没有转换结果生成，函数将返回*ESP_ERR_TIMEOUT*。
- 如果 ADC 连续转换生成的结果填满了内部池，新产生的结果将丢失。下次调用*adc_continuous_read()* 时，将返回*ESP_ERR_INVALID_STATE*，提示此情况发生。

此 API 提供了一个读取所有 ADC 连续转换结果的机会。

从上述函数读取的 ADC 转换结果为原始数据。要根据 ADC 原始结果计算电压，可以使用以下公式：

```
Vout = Dout * Vmax / Dmax        (1)
```

其中：

Vout	**数据输出结果，代表电压。**
Dout	ADC 原始数据读取结果。
Vmax	可测量的最大模拟输入电压，与 ADC 衰减相关，请参考 技术参考手册 中的片上传感器与模拟信号处理章节。
Dmax	输出 ADC 原始数据读取结果的最大值，即 2^ 位宽，位宽即之前配置的*adc_digi_pattern_config_t::bit_width*。

若需进一步校准，将 ADC 原始结果转换为以 mV 为单位的电压数据，请参考*模数转换器 (ADC) 校准驱动程序*。

硬件限制

- 一个 ADC 单元一次只能运行一种操作模式，即连续模式或单次模式。*adc_continuous_start()* 提供了保护措施。
- 随机数生成器 (RNG) 以 ADC 为输入源。使用 ADC 连续转换模式驱动从 RNG 生成随机数时，随机性会减弱。

电源管理 启用电源管理，即启用*CONFIG_PM_ENABLE* 时，系统在空闲状态下，可能会调整 APB 时钟频率，这可能会改变 ADC 连续转换的行为。

然而，通过获取类型为*ESP_PM_APB_FREQ_MAX* 的电源管理锁，ADC 连续转换模式驱动可以阻止这种改变。调用*adc_continuous_start()* 启动连续转换后即可获取该锁。同样，调用*adc_continuous_stop()* 停止转换后将释放该锁。因此，必须确保*adc_continuous_start()* 和*adc_continuous_stop()* 成对出现，否则电源管理将失效。

IRAM 安全 ADC 连续转换模式驱动的所有 API 均非 IRAM 安全。禁用 cache 时，不应运行这类 API。启用 Kconfig 选项*CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE* 可确保驱动的内部 ISR 处理程序为 IRAM 安全，此时即使禁用 cache，驱动仍然会将转换结果保存到其内部缓冲池中。

线程安全 ADC 连续转换模式驱动的 API 不一定线程安全，但驱动程序提供了共享硬件互斥，详情请参阅*硬件限制*。

应用示例

- ADC 连续转换模式示例：peripherals/adc/continuous_read。

API 参考

Header File

- components/esp_adc/include/esp_adc/adc_continuous.h
- This header file can be included with:

```
#include "esp_adc/adc_continuous.h"
```

- This header file is a part of the API provided by the `esp_adc` component. To declare that your component depends on `esp_adc`, add the following to your CMakeLists.txt:

```
REQUIRES esp_adc
```

or

```
PRIV_REQUIRES esp_adc
```

Functions

esp_err_t **adc_continuous_new_handle**(const *adc_continuous_handle_cfg_t* *hdl_config, *adc_continuous_handle_t* *ret_handle)

Initialize ADC continuous driver and get a handle to it.

参数

- **hdl_config** -- **[in]** Pointer to ADC initialization config. Refer to *adc_continuous_handle_cfg_t*.
- **ret_handle** -- **[out]** ADC continuous mode driver handle

返回

- ESP_ERR_INVALID_ARG If the combination of arguments is invalid.
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags
- ESP_ERR_NO_MEM If out of memory
- ESP_OK On success

esp_err_t **adc_continuous_config**(*adc_continuous_handle_t* handle, const *adc_continuous_config_t* *config)

Set ADC continuous mode required configurations.

参数

- **handle** -- **[in]** ADC continuous mode driver handle
- **config** -- **[in]** Refer to `adc_digi_config_t`.

返回

- ESP_ERR_INVALID_STATE: Driver state is invalid, you shouldn't call this API at this moment
- ESP_ERR_INVALID_ARG: If the combination of arguments is invalid.
- ESP_OK: On success

esp_err_t **adc_continuous_register_event_callbacks**(*adc_continuous_handle_t* handle, const *adc_continuous_evt_cbs_t* *cbs, void *user_data)

Register callbacks.

备注: User can deregister a previously registered callback by calling this function and setting the to-be-deregistered callback member in the `cbs` structure to NULL.

备注: When CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. Involved variables (including `user_data`) should be in internal RAM as well.

备注： You should only call this API when the ADC continuous mode driver isn't started. Check return value to know this.

参数

- **handle** -- **[in]** ADC continuous mode driver handle
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be delivered to the callback functions directly

返回

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_INVALID_STATE: Driver state is invalid, you shouldn't call this API at this moment

esp_err_t **adc_continuous_start**(*adc_continuous_handle_t* handle)

Start the ADC under continuous mode. After this, the hardware starts working.

参数 **handle** -- **[in]** ADC continuous mode driver handle

返回

- ESP_ERR_INVALID_STATE Driver state is invalid.
- ESP_OK On success

esp_err_t **adc_continuous_read**(*adc_continuous_handle_t* handle, uint8_t *buf, uint32_t length_max, uint32_t *out_length, uint32_t timeout_ms)

Read bytes from ADC under continuous mode.

参数

- **handle** -- **[in]** ADC continuous mode driver handle
- **buf** -- **[out]** Conversion result buffer to read from ADC. Suggest convert to *adc_digi_output_data_t* for `ADC Conversion Results`. See the subsection `Driver Backgrounds` in this header file to learn about this concept.
- **length_max** -- **[in]** Expected length of the Conversion Results read from the ADC, in bytes.
- **out_length** -- **[out]** Real length of the Conversion Results read from the ADC via this API, in bytes.
- **timeout_ms** -- **[in]** Time to wait for data via this API, in millisecond.

返回

- ESP_ERR_INVALID_STATE Driver state is invalid. Usually it means the ADC sampling rate is faster than the task processing rate.
- ESP_ERR_TIMEOUT Operation timed out
- ESP_OK On success

esp_err_t **adc_continuous_stop**(*adc_continuous_handle_t* handle)

Stop the ADC. After this, the hardware stops working.

参数 **handle** -- **[in]** ADC continuous mode driver handle

返回

- ESP_ERR_INVALID_STATE Driver state is invalid.
- ESP_OK On success

esp_err_t **adc_continuous_deinit**(*adc_continuous_handle_t* handle)

Deinitialize the ADC continuous driver.

参数 **handle** -- **[in]** ADC continuous mode driver handle

返回

- ESP_ERR_INVALID_STATE Driver state is invalid.
- ESP_OK On success

esp_err_t **adc_continuous_flush_pool**(*adc_continuous_handle_t* handle)

Flush the driver internal pool.

备注： This API is not supposed to be called in an ISR context

参数 **handle** -- **[in]** ADC continuous mode driver handle

返回

- ESP_ERR_INVALID_STATE Driver state is invalid, you should call this API when it's in init state
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_OK On success

esp_err_t **adc_continuous_io_to_channel**(int io_num, *adc_unit_t* *const unit_id, *adc_channel_t* *const channel)

Get ADC channel from the given GPIO number.

参数

- **io_num** -- **[in]** GPIO number
- **unit_id** -- **[out]** ADC unit
- **channel** -- **[out]** ADC channel

返回

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FOUND: The IO is not a valid ADC pad

esp_err_t **adc_continuous_channel_to_io**(*adc_unit_t* unit_id, *adc_channel_t* channel, int *const io_num)

Get GPIO number from the given ADC channel.

参数

- **unit_id** -- **[in]** ADC unit
- **channel** -- **[in]** ADC channel
- **io_num** -- **[out]** GPIO number
- **–** -- ESP_OK: On success
  - ESP_ERR_INVALID_ARG: Invalid argument

Structures

struct **adc_continuous_handle_cfg_t**

ADC continuous mode driver initial configurations.

Public Members

uint32_t **max_store_buf_size**

Max length of the conversion results that driver can store, in bytes.

uint32_t **conv_frame_size**

Conversion frame size, in bytes. This should be in multiples of `SOC_ADC_DIGI_DATA_BYTES_PER_CONV`.

uint32_t **flush_pool**

Flush the internal pool when the pool is full.

struct *adc_continuous_handle_cfg_t*::[anonymous] **flags**

Driver flags.

struct **adc_continuous_config_t**

ADC continuous mode driver configurations.

Public Members

uint32_t **pattern_num**

Number of ADC channels that will be used.

adc_digi_pattern_config_t ***adc_pattern**

List of configs for each ADC channel that will be used.

uint32_t **sample_freq_hz**

The expected ADC sampling frequency in Hz. Please refer to `soc/soc_caps.h` to know available sampling frequency range

adc_digi_convert_mode_t **conv_mode**

ADC DMA conversion mode, see `adc_digi_convert_mode_t`.

adc_digi_output_format_t **format**

ADC DMA conversion output format, see `adc_digi_output_format_t`.

struct **adc_continuous_evt_data_t**

Event data structure.

备注： The `conv_frame_buffer` is maintained by the driver itself, so never free this piece of memory.

Public Members

uint8_t ***conv_frame_buffer**

Pointer to conversion result buffer for one conversion frame.

uint32_t **size**

Conversion frame size.

struct **adc_continuous_evt_cbs_t**

Group of ADC continuous mode callbacks.

备注： These callbacks are all running in an ISR environment.

备注： When CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. Involved variables should be in internal RAM as well.

Public Members

adc_continuous_callback_t **on_conv_done**

Event callback, invoked when one conversion frame is done. See the subsection `Driver Backgrounds` in this header file to learn about the `conversion frame` concept.

adc_continuous_callback_t **on_pool_ovf**

Event callback, invoked when the internal pool is full.

Macros

ADC_MAX_DELAY

ADC read max timeout value, it may make the `adc_continuous_read` block forever if the OS supports.

Type Definitions

typedef struct adc_continuous_ctx_t ***adc_continuous_handle_t**

Type of adc continuous mode driver handle.

typedef bool (***adc_continuous_callback_t**)(*adc_continuous_handle_t* handle, const *adc_continuous_evt_data_t* *edata, void *user_data)

Prototype of ADC continuous mode event callback.

Param handle [in] ADC continuous mode driver handle
Param edata [in] Pointer to ADC continuous mode event data
Param user_data [in] User registered context, registered when in `adc_continuous_register_event_callbacks()`
Return Whether a high priority task is woken up by this function

警告： This document is not updated for ESP32P4 yet, so some of the content may not be correct.

This warning was automatically inserted due to the source file being in the *add_warnings_pages* list.

2.5.3 模数转换器 (ADC) 校准驱动程序

简介

在 ESP32-P4 中，模数转换器 (ADC) 比较输入的模拟电压和参考电压，以确定每一位数字输出结果。ESP32-P4 设计的 ADC 参考电压为 1100 mV。然而，不同芯片的真实参考电压可能会略有变化，范围在 1000 mV 到 1200 mV 之间。本文介绍了 ADC 校准驱动程序，可以降低参考电压不同带来的影响，获取更准确的输出结果。

功能概述

下文将分节概述安装和使用 ADC 校准驱动程序的基本步骤：

- 创建校准方案 - 介绍如何创建和删除校准方案句柄。
- 结果转换 - 介绍如何将原始 ADC 结果转换为校准后的结果。
- 线程安全 - 列出由驱动程序认证为线程安全的 API。
- 减少噪声 - 介绍一种常见的降低噪声的方法。

创建校准方案 ADC 校准驱动程序会提供 ADC 校准方案。对于驱动程序来说，每个 ADC 校准方案对应一个 ADC 校准句柄*adc_cali_handle_t*。

使用*adc_cali_check_scheme()* 可以查看芯片支持的校准方案。若已了解芯片支持的校准方案，可以跳过该步骤，直接调用对应函数创建校准方案句柄。

使用自定义 ADC 校准方案时，可以选择调整函数*adc_cali_check_scheme()*，或直接跳过该步骤，调用自定义函数创建校准方案句柄。

备注： 要使用自定义校准方案，可以通过提供创建函数，创建自己的校准方案句柄。请参阅 `components/esp_adc/interface/adc_cali_interface.h` 中的函数表 `adc_cali_scheme_t`，了解 ESP ADC 校准接口。

结果转换 对驱动程序进行完上述配置和初始化工作后，可以调用*adc_cali_raw_to_voltage()*，将原始 ADC 结果转换为校准结果，校准结果以 mV 为单位。该函数可能因参数无效而调用失败。如果函数返回*ESP_ERR_INVALID_STATE*，说明校准方案尚未创建。因此你需要创建一个校准方案句柄，通过*adc_cali_check_scheme()* 可以了解当前芯片支持的校准方案；你也可以提供自定义校准方案，创建对应的校准方案句柄。

获取电压

```
ESP_ERROR_CHECK(adc_cali_raw_to_voltage(adc_cali_handle, adc_raw[0][0], &
↪voltage[0][0]));
ESP_LOGI(TAG, "ADC%d Channel[%d] Cali Voltage: %d mV", ADC_UNIT_1 + 1, EXAMPLE_
↪ADC1_CHAN0, voltage[0][0]);
```

线程安全 驱动程序会确保工厂函数 `esp_adc_cali_new_scheme()` 的线程安全，使用时，可以直接从不同的 RTOS 任务中调用此类函数，无需额外锁保护。

其他以*adc_cali_handle_t* 作为第一个位置参数的函数均非线程安全，在没有设置互斥锁保护的任务中，应避免从多个任务中调用这类函数。

减少噪声 ESP32-P4 ADC 对噪声敏感，可能导致 ADC 读数出现较大偏差。根据不同使用场景，要减少噪声影响，你可能需要将旁路电容（如 100 nF 陶瓷电容）连接到 ADC 使用的输入管脚。此外，也可以通过多次采样，进一步减轻噪声的影响。

API 参考

Header File

- components/esp_adc/include/esp_adc/adc_cali.h
- This header file can be included with:

```
#include "esp_adc/adc_cali.h"
```

- This header file is a part of the API provided by the `esp_adc` component. To declare that your component depends on `esp_adc`, add the following to your CMakeLists.txt:

```
REQUIRES esp_adc
```

or

```
PRIV_REQUIRES esp_adc
```

Functions

esp_err_t **adc_cali_check_scheme**(*adc_cali_scheme_ver_t* *scheme_mask)

Check the supported ADC calibration scheme.

参数 **scheme_mask** -- **[out]** Supported ADC calibration scheme(s)

返回

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_SUPPORTED: No supported calibration scheme

esp_err_t **adc_cali_raw_to_voltage**(*adc_cali_handle_t* handle, int raw, int *voltage)

Convert ADC raw data to calibrated voltage.

参数

- **handle** -- **[in]** ADC calibration handle
- **raw** -- **[in]** ADC raw data
- **voltage** -- **[out]** Calibrated ADC voltage (in mV)

返回

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid state, scheme didn't registered

Type Definitions

typedef struct adc_cali_scheme_t ***adc_cali_handle_t**

ADC calibration handle.

Enumerations

enum **adc_cali_scheme_ver_t**

ADC calibration scheme.

Values:

enumerator **ADC_CALI_SCHEME_VER_LINE_FITTING**

Line fitting scheme.

enumerator **ADC_CALI_SCHEME_VER_CURVE_FITTING**

Curve fitting scheme.

Header File

- components/esp_adc/include/esp_adc/adc_cali_scheme.h
- This header file can be included with:

```
#include "esp_adc/adc_cali_scheme.h"
```

- This header file is a part of the API provided by the `esp_adc` component. To declare that your component depends on `esp_adc`, add the following to your CMakeLists.txt:

```
REQUIRES esp_adc
```

or

```
PRIV_REQUIRES esp_adc
```

2.5.4 Analog Comparator

Introduction

Analog Comparator is a peripheral that can be used to compare a source signal with the internal reference voltage or an external reference signal.

It is a cost effective way to replace an amplifier comparator in some scenarios. But unlike the continuous comparing of the amplifier comparator, ESP Analog Comparator is driven by a source clock, which decides the sampling frequency.

Analog Comparator on ESP32-P4 has 2 unit(s), the channels in the unit(s) are:

UNIT0

- Source Channel: GPIO52
- External Reference Channel: GPIO51
- Internal Reference Channel: Range 0% ~ 70% of the VDD, the step is 10% of the VDD

UNIT1

- Source Channel: GPIO54
- External Reference Channel: GPIO53
- Internal Reference Channel: Range 0% ~ 70% of the VDD, the step is 10% of the VDD

Functional Overview

The following sections of this document cover the typical steps to install and operate an Analog Comparator unit:

- *Resource Allocation* - covers which parameters should be set up to get a unit handle and how to recycle the resources when it finishes working.
- *Further Configurations* - covers the other configurations that might need to specific and what they are used for.
- *Enable and Disable Unit* - covers how to enable and disable the unit.
- *Power Management* - describes how different source clock selections can affect power consumption.
- *IRAM Safe* - lists which functions are supposed to work even when the cache is disabled.
- *Thread Safety* - lists which APIs are guaranteed to be thread safe by the driver.
- *Kconfig Options* - lists the supported Kconfig options that can be used to make a different effect on driver behavior.
- *ETM Events* -

Resource Allocation An Analog Comparator unit channel is represented by *ana_cmpr_handle_t*. Each unit can support either an internal or an external reference.

To allocate the resource of the Analog Comparator unit, *ana_cmpr_new_unit()* need to be called to get the handle of the unit. Configurations *ana_cmpr_config_t* need to be specified while allocating the unit:

- *ana_cmpr_config_t::unit* selects the Analog Comparator unit.
- *ana_cmpr_config_t::clk_src* selects the source clock for Analog Comparator, it can affect the sampling frequency. Note that the clock source of the Analog Comparator comes from the io mux, it is shared with GPIO extension peripherals like SDM (Sigma-Delta Modulation) and Glitch Filter. The configuration will fail if you specific different clock sources for multiple GPIO extension peripherals. The default clock sources of these peripherals are same, typically, we select *soc_periph_ana_cmpr_clk_src_t::ANA_CMPR_CLK_SRC_DEFAULT* as the clock source.
- *ana_cmpr_config_t::ref_src* selects the reference source from internal voltage or external signal.
- *ana_cmpr_config_t::cross_type* selects which kind of cross type can trigger the interrupt.

The function *ana_cmpr_new_unit()* can fail due to various errors such as insufficient memory, invalid arguments, etc. If a previously created Analog Comparator unit is no longer required, you should recycle it by calling *ana_cmpr_del_unit()*. It allows the underlying HW channel to be used for other purposes. Before deleting an Analog Comparator unit handle, you should disable it by `ana_cmpr_unit_disable()` in advance, or make sure it has not enabled yet by `ana_cmpr_unit_enable()`.

```
#include "driver/ana_cmpr.h"

ana_cmpr_handle_t cmpr = NULL;
ana_cmpr_config_t config = {
    .unit = 0,
    .clk_src = ANA_CMPR_CLK_SRC_DEFAULT,
    .ref_src = ANA_CMPR_REF_SRC_INTERNAL,
    .cross_type = ANA_CMPR_CROSS_ANY,
};
ESP_ERROR_CHECK(ana_cmpr_new_unit(&config, &cmpr));
// ...
ESP_ERROR_CHECK(ana_cmpr_del_unit(cmpr));
```

Further Configurations

- `ana_cmpr_set_intl_reference()` - Specify the internal reference voltage when *ana_cmpr_ref_source_t::ANA_CMPR_REF_SRC_INTERNAL* is selected as reference source.

It requires *ana_cmpr_internal_ref_config_t::ref_volt* to specify the voltage. The voltage related to the VDD power supply, which can only support a certain fixed percentage of VDD. Currently on ESP32-P4, the internal reference voltage can be range to 0 ~ 70% VDD with a step 10%.

```
#include "driver/ana_cmpr.h"

ana_cmpr_internal_ref_config_t ref_cfg = {
    .ref_volt = ANA_CMPR_REF_VOLT_50_PCT_VDD,
};
ESP_ERROR_CHECK(ana_cmpr_set_internal_reference(cmpr, &ref_cfg));
```

- *ana_cmpr_set_debounce()* - Set the debounce configuration.

It requires *ana_cmpr_debounce_config_t::wait_us* to set the interrupt waiting time. The interrupt is disabled temporarily for *ana_cmpr_debounce_config_t::wait_us* micro seconds, so that the frequent triggering can be avoid while the source signal crossing the reference signal. That is, the waiting time is supposed to be inverse ratio to the relative frequency between the source and reference. If the waiting time is set too short, it can not bypass the jitter totally, but if too long, the next crossing interrupt might be missed.

```
#include "driver/ana_cmpr.h"

ana_cmpr_debounce_config_t dbc_cfg = {
    .wait_us = 1,
};
ESP_ERROR_CHECK(ana_cmpr_set_debounce(cmpr, &dbc_cfg));
```

- *ana_cmpr_set_cross_type()* - Set the source signal cross type.

The initial cross type is set int *ana_cmpr_new_unit()*, this function can update the cross type, even in ISR context.

```
#include "driver/ana_cmpr.h"

ESP_ERROR_CHECK(ana_cmpr_set_cross_type(cmpr, ANA_CMPR_CROSS_POS));
```

- *ana_cmpr_register_event_callbacks()* - Register the callbacks.

Currently it supports *ana_cmpr_event_callbacks_t::on_cross*, it will be called when the crossing event (specified by *ana_cmpr_config_t::cross_type*) occurs.

```
#include "driver/ana_cmpr.h"

static bool IRAM_ATTR example_ana_cmpr_on_cross_callback(ana_cmpr_handle_t cmpr,
```

(下页继续)

(续上页)

```
                                                    const ana_cmpr_cross_event_
↪data_t *edata,
                                                    void *user_ctx)
{
    // ...
    return false;
}
ana_cmpr_event_callbacks_t cbs = {
    .on_cross = example_ana_cmpr_on_cross_callback,
};
ESP_ERROR_CHECK(ana_cmpr_register_event_callbacks(cmpr, &cbs, NULL));
```

备注： When *CONFIG_ANA_CMPR_ISR_IRAM_SAFE* is enabled, you should guarantee the callback context and involved data to be in internal RAM by add the attribute `IRAM_ATTR`. (See more in *IRAM Safe*)

Enable and Disable Unit

- *ana_cmpr_enable()* - Enable the Analog Comparator unit.
- *ana_cmpr_disable()* - Disable the Analog Comparator unit.

After the Analog Comparator unit is enabled and the crossing event interrupt is enabled, a power management lock will be acquired if the power management is enabled (see *Power Management*). Under the **enable** state, only `ana_cmpr_set_intl_reference()` and *ana_cmpr_set_debounce()* can be called, other functions can only be called after the unit is disabled.

Calling *ana_cmpr_disable()* does the opposite.

Power Management When power management is enabled (i.e., *CONFIG_PM_ENABLE* is on), the system will adjust the APB frequency before going into light sleep, thus potentially changing the resolution of the Analog Comparator.

However, the driver can prevent the system from changing APB frequency by acquiring a power management lock of type *ESP_PM_NO_LIGHT_SLEEP*. Whenever the driver creates a Analog Comparator unit instance that has selected the clock source like *ANA_CMPR_CLK_SRC_DEFAULT* or *ANA_CMPR_CLK_SRC_XTAL* as its clock source, the driver guarantees that the power management lock is acquired when enable the channel by *ana_cmpr_enable()*. Likewise, the driver releases the lock when *ana_cmpr_disable()* is called for that channel.

IRAM Safe By default, the Analog Comparator interrupt will be deferred when the Cache is disabled for reasons like programming/erasing Flash. Thus the alarm interrupt will not get executed in time, which is not expected in a real-time application.

There is a Kconfig option *CONFIG_ANA_CMPR_ISR_IRAM_SAFE* that:

1. Enables the interrupt being serviced even when cache is disabled
2. Places all functions that used by the ISR into IRAM[1]
3. Places driver object into DRAM (in case it is allocated on PSRAM)

This allows the interrupt to run while the cache is disabled but comes at the cost of increased IRAM consumption.

There is a Kconfig option *CONFIG_ANA_CMPR_CTRL_FUNC_IN_IRAM* that can put commonly used IO control functions into IRAM as well. So that these functions can also be executable when the cache is disabled. These IO control functions are listed as follows:

- *ana_cmpr_set_internal_reference()*
- *ana_cmpr_set_debounce()*

[1] *ana_cmpr_event_callbacks_t::on_cross* callback and the functions invoked by itself should also be placed in IRAM, you need to take care of them by themselves.

- *ana_cmpr_set_cross_type()*

Thread Safety The factory function *ana_cmpr_new_unit()* is guaranteed to be thread safe by the driver, which means, user can call it from different RTOS tasks without protection by extra locks. The following functions are allowed to run under ISR context, the driver uses a critical section to prevent them being called concurrently in both task and ISR.

- *ana_cmpr_set_internal_reference()*
- *ana_cmpr_set_debounce()*
- *ana_cmpr_set_cross_type()*

Other functions that take the *ana_cmpr_handle_t* as the first positional parameter, are not treated as thread safe. Which means the user should avoid calling them from multiple tasks.

Kconfig Options

- *CONFIG_ANA_CMPR_ISR_IRAM_SAFE* controls whether the default ISR handler can work when cache is disabled, see *IRAM Safe* for more information.
- *CONFIG_ANA_CMPR_CTRL_FUNC_IN_IRAM* controls where to place the Analog Comparator control functions (IRAM or Flash), see *IRAM Safe* for more information.
- *CONFIG_ANA_CMPR_ENABLE_DEBUG_LOG* is used to enabled the debug log output. Enabling this option increases the firmware binary size.

ETM Events To create an analog comparator cross event, you need to include `driver/ana_cmpr_etm.h` additionally, and allocate the event by `ana_cmpr_new_etm_event()`. You can refer to *ETM* for how to connect an event to a task.

Application Example

- peripherals/analog_comparator shows the basic usage of the analog comparator, and other potential usages like hysteresis comparator and SPWM generator.

API Reference

Header File

- components/esp_driver_ana_cmpr/include/driver/ana_cmpr.h
- This header file can be included with:

```
#include "driver/ana_cmpr.h"
```

- This header file is a part of the API provided by the `esp_driver_ana_cmpr` component. To declare that your component depends on `esp_driver_ana_cmpr`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_ana_cmpr
```

or

```
PRIV_REQUIRES esp_driver_ana_cmpr
```

Functions

esp_err_t **ana_cmpr_new_unit**(const *ana_cmpr_config_t* *config, *ana_cmpr_handle_t* *ret_cmpr)

Allocating a new analog comparator unit handle.

参数

- **config** -- **[in]** The config of the analog comparator unit
- **ret_cmpr** -- **[out]** The returned analog comparator unit handle

返回

- ESP_OK Allocate analog comparator unit handle success
- ESP_ERR_NO_MEM No memory for the analog comparator structure
- ESP_ERR_INVALID_ARG NULL pointer of the parameters or wrong unit number
- ESP_ERR_INVALID_STATE The unit has been allocated or the clock source has been occupied

esp_err_t **ana_cmpr_del_unit**(*ana_cmpr_handle_t* cmpr)

Delete the analog comparator unit handle.

参数 **cmpr** -- **[in]** The handle of analog comparator unit

返回

- ESP_OK Delete analog comparator unit handle success
- ESP_ERR_INVALID_ARG NULL pointer of the parameters or wrong unit number
- ESP_ERR_INVALID_STATE The analog comparator is not disabled yet

esp_err_t **ana_cmpr_set_internal_reference**(*ana_cmpr_handle_t* cmpr, const *ana_cmpr_internal_ref_config_t* *ref_cfg)

Set internal reference configuration.

备 注: This function only need to be called when *ana_cmpr_config_t::ref_src* is ANA_CMPR_REF_SRC_INTERNAL.

备注: This function is allowed to run within ISR context including intr callbacks

备注: This function will be placed into IRAM if `CONFIG_ANA_CMPR_CTRL_FUNC_IN_IRAM` is on, so that it's allowed to be executed when Cache is disabled

参数

- **cmpr** -- **[in]** The handle of analog comparator unit
- **ref_cfg** -- **[in]** Internal reference configuration

返回

- ESP_OK Set denounce configuration success
- ESP_ERR_INVALID_ARG NULL pointer of the parameters
- ESP_ERR_INVALID_STATE The reference source is not `ANA_CMPR_REF_SRC_INTERNAL`

esp_err_t **ana_cmpr_set_debounce**(*ana_cmpr_handle_t* cmpr, const *ana_cmpr_debounce_config_t* *dbc_cfg)

Set debounce configuration to the analog comparator.

备注: This function is allowed to run within ISR context including intr callbacks

备注: This function will be placed into IRAM if `CONFIG_ANA_CMPR_CTRL_FUNC_IN_IRAM` is on, so that it's allowed to be executed when Cache is disabled

参数

- **cmpr** -- **[in]** The handle of analog comparator unit
- **dbc_cfg** -- **[in]** Debounce configuration

返回

- ESP_OK Set denounce configuration success
- ESP_ERR_INVALID_ARG NULL pointer of the parameters

esp_err_t **ana_cmpr_set_cross_type**(*ana_cmpr_handle_t* cmpr, *ana_cmpr_cross_type_t* cross_type)

Set the source signal cross type.

备注: The initial cross type is configured in `ana_cmpr_new_unit`, this function can update the cross type

备注: This function is allowed to run within ISR context including intr callbacks

备注: This function will be placed into IRAM if `CONFIG_ANA_CMPR_CTRL_FUNC_IN_IRAM` is on, so that it's allowed to be executed when Cache is disabled

参数

- **cmpr** -- **[in]** The handle of analog comparator unit
- **cross_type** -- **[in]** The source signal cross type that can trigger the interrupt

返回

- ESP_OK Set denounce configuration success
- ESP_ERR_INVALID_ARG NULL pointer of the parameters

esp_err_t **ana_cmpr_register_event_callbacks**(*ana_cmpr_handle_t* cmpr, const *ana_cmpr_event_callbacks_t* *cbs, void *user_data)

Register analog comparator interrupt event callbacks.

备注: This function can only be called before enabling the unit

参数

- **cmpr** -- **[in]** The handle of analog comparator unit
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** The user data that will be passed to callback functions directly

返回

- ESP_OK Register callbacks success
- ESP_ERR_INVALID_ARG NULL pointer of the parameters
- ESP_ERR_INVALID_STATE The analog comparator has been enabled

esp_err_t **ana_cmpr_enable**(*ana_cmpr_handle_t* cmpr)

Enable the analog comparator unit.

参数 **cmpr** -- **[in]** The handle of analog comparator unit

返回

- ESP_OK Enable analog comparator unit success
- ESP_ERR_INVALID_ARG NULL pointer of the parameters
- ESP_ERR_INVALID_STATE The analog comparator has been enabled

esp_err_t **ana_cmpr_disable**(*ana_cmpr_handle_t* cmpr)

Disable the analog comparator unit.

参数 **cmpr** -- **[in]** The handle of analog comparator unit

返回

- ESP_OK Disable analog comparator unit success
- ESP_ERR_INVALID_ARG NULL pointer of the parameters
- ESP_ERR_INVALID_STATE The analog comparator has disabled already

esp_err_t **ana_cmpr_get_gpio**(*ana_cmpr_unit_t* unit, *ana_cmpr_channel_type_t* chan_type, int *gpio_num)

Get the specific GPIO number of the analog comparator unit.

参数

- **unit** -- **[in]** The handle of analog comparator unit
- **chan_type** -- **[in]** The channel type of analog comparator, like source channel or reference channel
- **gpio_num** -- **[out]** The output GPIO number of this channel

返回

- ESP_OK Get GPIO success
- ESP_ERR_INVALID_ARG NULL pointer of the parameters or wrong unit number or wrong channel type

Structures

struct **ana_cmpr_config_t**

Analog comparator unit configuration.

Public Members

ana_cmpr_unit_t **unit**

Analog comparator unit

ana_cmpr_clk_src_t **clk_src**

The clock source of the analog comparator, which decide the resolution of the comparator

ana_cmpr_ref_source_t **ref_src**

Reference signal source of the comparator, select using ANA_CMPR_REF_SRC_INTERNAL or ANA_CMPR_REF_SRC_EXTERNAL. For internal reference, the reference voltage should be set to `internal_ref_volt`, for external reference, the reference signal should be connect to `ANA_CMPRx_EXT_REF_GPIO`

ana_cmpr_cross_type_t **cross_type**

The crossing types that can trigger interrupt

int **intr_priority**

The interrupt priority, range 0~7, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3) otherwise the larger the higher, 7 is NMI

uint32_t **io_loop_back**

Enable this field when the other signals that output on the comparison pins are supposed to be fed back. Normally used for debug/test scenario

struct *ana_cmpr_config_t*::[anonymous] **flags**

Analog comparator driver flags

struct **ana_cmpr_internal_ref_config_t**

Analog comparator internal reference configuration.

Public Members

ana_cmpr_ref_voltage_t **ref_volt**

The internal reference voltage. It can be specified to a certain fixed percentage of the VDD power supply, currently supports 0%~70% VDD with a step 10%

struct **ana_cmpr_debounce_config_t**

Analog comparator debounce filter configuration.

Public Members

uint32_t **wait_us**

The wait time of re-enabling the interrupt after the last triggering, it is used to avoid the spurious triggering while the source signal crossing the reference signal. The value should regarding how fast the source signal changes, e.g., a rapid signal requires a small wait time, otherwise the next crosses may be missed. (Unit: micro second)

struct **ana_cmpr_event_callbacks_t**

Group of Analog Comparator callbacks.

备注： The callbacks are all running under ISR environment

备注： When CONFIG_ANA_CMPR_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

Public Members

ana_cmpr_cross_cb_t **on_cross**

The callback function on cross interrupt

Header File

- components/esp_driver_ana_cmpr/include/driver/ana_cmpr_types.h
- This header file can be included with:

```
#include "driver/ana_cmpr_types.h"
```

- This header file is a part of the API provided by the `esp_driver_ana_cmpr` component. To declare that your component depends on `esp_driver_ana_cmpr`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_ana_cmpr
```

or

```
PRIV_REQUIRES esp_driver_ana_cmpr
```

Structures

struct **ana_cmpr_cross_event_data_t**

Analog comparator cross event data.

Public Members

ana_cmpr_cross_type_t **cross_type**

The cross type of the target signal to the reference signal. Will either be ANA_CMPR_CROSS_POS or ANA_CMPR_CROSS_NEG Always be ANA_CMPR_CROSS_ANY if target does not support independent interrupt (like ESP32H2)

Macros

ANA_CMPR_UNIT_0

Deprecated:

Analog comparator unit 0

Type Definitions

typedef int **ana_cmpr_unit_t**

Analog comparator unit.

typedef struct ana_cmpr_t ***ana_cmpr_handle_t**

Analog comparator unit handle.

typedef *soc_periph_ana_cmpr_clk_src_t* **ana_cmpr_clk_src_t**

Analog comparator clock source.

typedef bool (***ana_cmpr_cross_cb_t**)(*ana_cmpr_handle_t* cmpr, const *ana_cmpr_cross_event_data_t* *edata, void *user_ctx)

Prototype of Analog comparator event callback.

Param cmpr **[in]** Analog Comparator handle, created from `ana_cmpr_new_unit()`

Param edata **[in]** Point to Analog Comparator event data. The lifecycle of this pointer memory is inside this function, user should copy it into static memory if used outside this function. (Currently not use)

Param user_ctx **[in]** User registered context, passed from `ana_cmpr_register_event_callbacks()`

Return Whether a high priority task has been waken up by this callback function

Enumerations

enum **ana_cmpr_ref_source_t**

Analog comparator reference source.

Values:

enumerator **ANA_CMPR_REF_SRC_INTERNAL**

Analog Comparator internal reference source, related to VDD

enumerator **ANA_CMPR_REF_SRC_EXTERNAL**

Analog Comparator external reference source, from `ANA_CMPR0_EXT_REF_GPIO`

enum **ana_cmpr_channel_type_t**

Analog comparator channel type.

Values:

enumerator **ANA_CMPR_SOURCE_CHAN**

Analog Comparator source channel, which is used to input the signal that to be compared

enumerator **ANA_CMPR_EXT_REF_CHAN**

Analog Comparator external reference channel, which is used as the reference signal

enum **ana_cmpr_cross_type_t**

Analog comparator interrupt type.

Values:

enumerator **ANA_CMPR_CROSS_DISABLE**

Disable the cross event interrupt

enumerator **ANA_CMPR_CROSS_POS**

Positive cross can trigger event interrupt

enumerator **ANA_CMPR_CROSS_NEG**

Negative cross can trigger event interrupt

enumerator **ANA_CMPR_CROSS_ANY**

Any cross can trigger event interrupt

enum **ana_cmpr_ref_voltage_t**

Analog comparator internal reference voltage.

Values:

enumerator **ANA_CMPR_REF_VOLT_0_PCT_VDD**

Internal reference voltage equals to 0% VDD

enumerator **ANA_CMPR_REF_VOLT_10_PCT_VDD**

Internal reference voltage equals to 10% VDD

enumerator **ANA_CMPR_REF_VOLT_20_PCT_VDD**

Internal reference voltage equals to 20% VDD

enumerator **ANA_CMPR_REF_VOLT_30_PCT_VDD**

Internal reference voltage equals to 30% VDD

enumerator **ANA_CMPR_REF_VOLT_40_PCT_VDD**

Internal reference voltage equals to 40% VDD

enumerator **ANA_CMPR_REF_VOLT_50_PCT_VDD**

Internal reference voltage equals to 50% VDD

enumerator **ANA_CMPR_REF_VOLT_60_PCT_VDD**

Internal reference voltage equals to 60% VDD

enumerator **ANA_CMPR_REF_VOLT_70_PCT_VDD**

Internal reference voltage equals to 70% VDD

2.5.5 时钟树

ESP32-P4 的时钟子系统用于从一系列根时钟中提取并分配系统/模块时钟。时钟树驱动程序负责维护系统时钟的基本功能，并管理模块时钟间的复杂关系。

本文档首先介绍了根时钟和模块时钟，随后介绍了可供用户调用的时钟树 API，调用这些 API，可以监测模块时钟的运行状态。

简介

本节列出了 ESP32-P4 支持的根时钟和模块时钟的定义，这些定义通常用于驱动程序配置，有助于为外设选择合适的时钟源。

根时钟 根时钟会产生可靠的时钟信号，经各种门、复用器、分频器或倍频器传递，这些时钟信号最终成为 CPU 内核、Wi-Fi、蓝牙、RTC 及外设等功能模块的时钟源。

ESP32-P4 的根时钟列在*soc_root_clk_t* 中：

- 内部 17.5 MHz RC 振荡器 (RC_FAST)
  此 RC 振荡器可产生约 17.5 MHz 的时钟信号输出，标识为 `RC_FAST_CLK`。
  在运行时，通过校准，可以计算 `RC_FAST_CLK` 的实际频率。
- 外部 40 MHz 晶振 (XTAL)
- 内部 136 kHz RC 振荡器 (RC_SLOW)
  此 RC 振荡器产生约 136 kHz 的时钟信号输出，标识为 `RC_SLOW_CLK`。在运行时，通过校准，可以计算该时钟信号的实际频率。
- 外部 32 kHz 晶振 - 可选 (XTAL32K)
  `XTAL32K_CLK` 的时钟源是连接到 `XTAL_32K_P` 和 `XTAL_32K_N` 管脚的 32 kHz 晶振。
  通过校准，可以计算 `XTAL32K_CLK` 的实际频率。
- 内部 32 kHz RC 振荡器 (RC32K)
  在运行时，通过校准，可以计算该时钟信号的实际频率。

与晶振产生的信号相比，从 RC 振荡器电路产生的信号通常精度较低，且容易受环境影响。因此，ESP32-P4 为 `RTC_SLOW_CLK` 提供了几种时钟源选项，可以根据对系统时间精度和对功耗的要求选择。更多详情，请参阅*RTC 定时器时钟源*。

模块时钟 ESP32-P4 的可用模块时钟在*soc_module_clk_t* 中列出，每个模块时钟都有其唯一 ID。查阅文档中的枚举值，即可获取各模块时钟的详细信息。

使用 API

时钟树驱动程序提供了一个一体化接口，可以获取模块时钟的频率，即*esp_clk_tree_src_get_freq_hz()*。通过该函数，你可以在任何时刻，通过提供时钟名称*soc_module_clk_t* 和指定返回频率值的精度级别*esp_clk_tree_src_freq_precision_t*，获取时钟频率。

API 参考

Header File

- components/soc/esp32p4/include/soc/clk_tree_defs.h
- This header file can be included with:

```
#include "soc/clk_tree_defs.h"
```

Macros

SOC_CLK_RC_FAST_FREQ_APPROX

Approximate RC_FAST_CLK frequency in Hz

SOC_CLK_RC_SLOW_FREQ_APPROX

Approximate RC_SLOW_CLK frequency in Hz

SOC_CLK_RC32K_FREQ_APPROX

Approximate RC32K_CLK frequency in Hz

SOC_CLK_XTAL32K_FREQ_APPROX

Approximate XTAL32K_CLK frequency in Hz

SOC_GPTIMER_CLKS

Array initializer for all supported clock sources of GPTimer.

The following code can be used to iterate all possible clocks:

```
soc_periph_gptimer_clk_src_t gptimer_clks[] = (soc_periph_gptimer_clk_src_
↪t)SOC_GPTIMER_CLKS;
for (size_t i = 0; i< sizeof(gptimer_clks) / sizeof(gptimer_clks[0]); i++) {
    soc_periph_gptimer_clk_src_t clk = gptimer_clks[i];
// Test GPTimer with the clock `clk`
}
```

SOC_RMT_CLKS

Array initializer for all supported clock sources of RMT.

SOC_UART_CLKS

Array initializer for all supported clock sources of UART.

SOC_LP_UART_CLKS

Array initializer for all supported clock sources of LP_UART.

SOC_MCPWM_TIMER_CLKS

Array initializer for all supported clock sources of MCPWM Timer.

SOC_MCPWM_CAPTURE_CLKS

Array initializer for all supported clock sources of MCPWM Capture Timer.

SOC_MCPWM_CARRIER_CLKS

Array initializer for all supported clock sources of MCPWM Carrier.

SOC_I2S_CLKS

Array initializer for all supported clock sources of I2S.

SOC_LP_I2S_CLKS

Array initializer for all supported clock sources of LP I2S.

SOC_LCD_CLKS

Array initializer for all supported clock sources of LCD.

SOC_CAM_CLKS

Array initializer for all supported clock sources of CAM.

SOC_MIPI_CSI_PHY_CLKS

Array initializer for all supported clock sources of MIPI CSI PHY interface.

SOC_MIPI_DSI_DPI_CLKS

Array initializer for all supported clock sources of MIPI DSI DPI interface.

SOC_MIPI_DSI_PHY_CLKS

Array initializer for all supported clock sources of MIPI DSI PHY interface.

SOC_I2C_CLKS

Array initializer for all supported clock sources of I2C.

SOC_LP_I2C_CLKS

Array initializer for all supported clock sources of LP_I2C.

SOC_SPI_CLKS

Array initializer for all supported clock sources of SPI.

SOC_PSRAM_CLKS

Array initializer for all supported clock sources of PSRAM.

SOC_FLASH_CLKS

Array initializer for all supported clock sources of FLASH.

SOC_ISP_CLKS

Array initializer for all supported clock sources of ISP.

SOC_SDM_CLKS

Array initializer for all supported clock sources of SDM.

SOC_GLITCH_FILTER_CLKS

Array initializer for all supported clock sources of Glitch Filter.

SOC_ANA_CMPR_CLKS

Array initializer for all supported clock sources of Analog Comparator.

SOC_TWAI_CLKS

Array initializer for all supported clock sources of TWAI.

SOC_ADC_DIGI_CLKS

Array initializer for all supported clock sources of ADC digital controller.

SOC_ADC_RTC_CLKS

Array initializer for all supported clock sources of ADC RTC controller.

SOC_MWDT_CLKS

Array initializer for all supported clock sources of MWDT.

SOC_LEDC_CLKS

Array initializer for all supported clock sources of LEDC.

SOC_PARLIO_CLKS

Array initializer for all supported clock sources of PARLIO.

SOC_SDMMC_CLKS

Array initializer for all supported clock sources of SDMMC.

SOC_TEMP_SENSOR_CLKS

Array initializer for all supported clock sources of Temperature Sensor.

Enumerations

enum **soc_root_clk_t**

Root clock.

Values:

enumerator **SOC_ROOT_CLK_INT_RC_FAST**

Internal 17.5MHz RC oscillator

enumerator **SOC_ROOT_CLK_INT_RC_SLOW**

Internal 136kHz RC oscillator

enumerator **SOC_ROOT_CLK_EXT_XTAL**

External 40MHz crystal

enumerator **SOC_ROOT_CLK_EXT_XTAL32K**

External 32kHz crystal

enumerator **SOC_ROOT_CLK_INT_RC32K**

Internal 32kHz RC oscillator

enumerator **SOC_ROOT_CLK_EXT_OSC_SLOW**

External slow clock signal at pin1

enum **soc_cpu_clk_src_t**

CPU_CLK mux inputs, which are the supported clock sources for the CPU_CLK.

备注： Enum values are matched with the register field values on purpose

Values:

enumerator **SOC_CPU_CLK_SRC_XTAL**

Select XTAL_CLK as CPU_CLK source

enumerator **SOC_CPU_CLK_SRC_CPLL**

Select CPLL_CLK as CPU_CLK source (CPLL_CLK is the output of 40MHz crystal oscillator frequency multiplier, can be 320/360/400MHz)

enumerator **SOC_CPU_CLK_SRC_PLL**

Alias name for `SOC_CPU_CLK_SRC_CPLL`

enumerator **SOC_CPU_CLK_SRC_RC_FAST**

Select RC_FAST_CLK as CPU_CLK source

enumerator **SOC_CPU_CLK_SRC_INVALID**

Invalid CPU_CLK source

enum **soc_rtc_slow_clk_src_t**

RTC_SLOW_CLK mux inputs, which are the supported clock sources for the RTC_SLOW_CLK.

备注： Enum values are matched with the register field values on purpose

---

Values:

enumerator **SOC_RTC_SLOW_CLK_SRC_RC_SLOW**

Select RC_SLOW_CLK as RTC_SLOW_CLK source

enumerator **SOC_RTC_SLOW_CLK_SRC_XTAL32K**

Select XTAL32K_CLK as RTC_SLOW_CLK source

enumerator **SOC_RTC_SLOW_CLK_SRC_RC32K**

Select RC32K_CLK as RTC_SLOW_CLK source

enumerator **SOC_RTC_SLOW_CLK_SRC_INVALID**

Invalid RTC_SLOW_CLK source

enum **soc_rtc_fast_clk_src_t**

RTC_FAST_CLK mux inputs, which are the supported clock sources for the RTC_FAST_CLK.

备注： Enum values are matched with the register field values on purpose

---

Values:

enumerator **SOC_RTC_FAST_CLK_SRC_RC_FAST**

Select RC_FAST_CLK as RTC_FAST_CLK source

enumerator **SOC_RTC_FAST_CLK_SRC_XTAL**

Select XTAL_CLK as RTC_FAST_CLK source

enumerator **SOC_RTC_FAST_CLK_SRC_XTAL_DIV**

Alias name for `SOC_RTC_FAST_CLK_SRC_XTAL`

enumerator **SOC_RTC_FAST_CLK_SRC_LP_PLL**

Select LP_PLL_CLK as RTC_FAST_CLK source (LP_PLL_CLK is a 8MHz clock sourced from RC32K or XTAL32K)

enumerator **SOC_RTC_FAST_CLK_SRC_INVALID**

Invalid RTC_FAST_CLK source

enum **soc_lp_pll_clk_src_t**

LP_PLL_CLK mux inputs, which are the supported clock sources for the LP_PLL_CLK.

备注： Enum values are matched with the register field values on purpose

Values:

enumerator **SOC_LP_PLL_CLK_SRC_RC32K**

Select RC32K_CLK as LP_PLL_CLK source

enumerator **SOC_LP_PLL_CLK_SRC_XTAL32K**

Select XTAL32K_CLK as LP_PLL_CLK source

enumerator **SOC_LP_PLL_CLK_SRC_INVALID**

Invalid LP_PLL_CLK source

enum **soc_xtal_freq_t**

Possible main XTAL frequency options on the target.

备注： Enum values equal to the frequency value in MHz

备注： Not all frequency values listed here are supported in IDF. Please check SOC_XTAL_SUPPORT_XXX in soc_caps.h for the supported ones.

Values:

enumerator **SOC_XTAL_FREQ_40M**

40MHz XTAL

enum **soc_module_clk_t**

Supported clock sources for modules (CPU, peripherals, RTC, etc.)

备注： enum starts from 1, to save 0 for special purpose

Values:

enumerator **SOC_MOD_CLK_CPU**

CPU_CLK can be sourced from XTAL, CPLL, or RC_FAST by configuring soc_cpu_clk_src_t

enumerator **SOC_MOD_CLK_RTC_FAST**

RTC_FAST_CLK can be sourced from XTAL, RC_FAST, or LP_PLL by configuring soc_rtc_fast_clk_src_t

enumerator **SOC_MOD_CLK_RTC_SLOW**

RTC_SLOW_CLK can be sourced from RC_SLOW, XTAL32K, or RC32K by configuring soc_rtc_slow_clk_src_t

enumerator **SOC_MOD_CLK_PLL_F20M**

PLL_F20M_CLK is derived from SPLL (clock gating + "fixed" divider of 24), it has a fixed frequency of 20MHz

enumerator **SOC_MOD_CLK_PLL_F25M**

PLL_F25M_CLK is derived from MPLL (clock gating + configurable divider), it will have a frequency of 25MHz

enumerator **SOC_MOD_CLK_PLL_F80M**

PLL_F80M_CLK is derived from SPLL (clock gating + "fixed" divider of 6), it has a fixed frequency of 80MHz

enumerator **SOC_MOD_CLK_PLL_F160M**

PLL_F160M_CLK is derived from SPLL (clock gating + "fixed" divider of 3), it has a fixed frequency of 160MHz

enumerator **SOC_MOD_CLK_PLL_F240M**

PLL_F240M_CLK is derived from SPLL (clock gating + "fixed" divider of 2), it has a fixed frequency of 240MHz

enumerator **SOC_MOD_CLK_CPLL**

CPLL is from 40MHz XTAL oscillator frequency multipliers

enumerator **SOC_MOD_CLK_SPLL**

SPLL is from 40MHz XTAL oscillator frequency multipliers, it has a "fixed" frequency of 480MHz

enumerator **SOC_MOD_CLK_MPLL**

MPLL is from 40MHz XTAL oscillator frequency multipliers

enumerator **SOC_MOD_CLK_XTAL32K**

XTAL32K_CLK comes from the external 32kHz crystal, passing a clock gating to the peripherals

enumerator **SOC_MOD_CLK_RC_FAST**

RC_FAST_CLK comes from the internal 20MHz rc oscillator, passing a clock gating to the peripherals

enumerator **SOC_MOD_CLK_XTAL**

XTAL_CLK comes from the external 40MHz crystal

enumerator **SOC_MOD_CLK_APLL**

Audio PLL is sourced from PLL, and its frequency is configurable through APLL configuration registers

enumerator **SOC_MOD_CLK_XTAL_D2**

XTAL_D2_CLK comes from the external 40MHz crystal, passing a div of 2 to the LP peripherals

enumerator **SOC_MOD_CLK_LP_PLL**

LP_PLL is from 32kHz XTAL oscillator frequency multipliers, it has a fixed frequency of 8MHz

enumerator **SOC_MOD_CLK_LP_DYN_FAST**

LP_DYN_FAST can be derived from RTC_SLOW_CLK or RTC_FAST_CLK depending on the chip's power mode: in active mode, select RTC_FAST_CLK as the clock source; in light/deep sleep mode, select RTC_SLOW_CLK as the clock source

enumerator **SOC_MOD_CLK_LP_PERI**

LP_PERI_CLK is derived from LP_DYN_FAST (configurable divider)

enumerator **SOC_MOD_CLK_INVALID**

Indication of the end of the available module clock sources

enum **soc_periph_systimer_clk_src_t**

Type of SYSTIMER clock source.

Values:

enumerator **SYSTIMER_CLK_SRC_XTAL**

SYSTIMER source clock is XTAL

enumerator **SYSTIMER_CLK_SRC_RC_FAST**

SYSTIMER source clock is RC_FAST

enumerator **SYSTIMER_CLK_SRC_DEFAULT**

SYSTIMER source clock default choice is XTAL

enum **soc_periph_gptimer_clk_src_t**

Type of GPTimer clock source.

Values:

enumerator **GPTIMER_CLK_SRC_PLL_F80M**

Select PLL_F80M as the source clock

enumerator **GPTIMER_CLK_SRC_RC_FAST**

Select RC_FAST as the source clock

enumerator **GPTIMER_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **GPTIMER_CLK_SRC_DEFAULT**

Select PLL_F80M as the default choice

enum **soc_periph_tg_clk_src_legacy_t**

Type of Timer Group clock source, reserved for the legacy timer group driver.

Values:

enumerator **TIMER_SRC_CLK_PLL_F80M**

Timer group clock source is PLL_F80M

enumerator **TIMER_SRC_CLK_XTAL**

Timer group clock source is XTAL

enumerator **TIMER_SRC_CLK_DEFAULT**

Timer group clock source default choice is PLL_F80M

enum **soc_periph_rmt_clk_src_t**

Type of RMT clock source.

Values:

enumerator **RMT_CLK_SRC_PLL_F80M**

Select PLL_F80M as the source clock

enumerator **RMT_CLK_SRC_RC_FAST**

Select RC_FAST as the source clock

enumerator **RMT_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **RMT_CLK_SRC_DEFAULT**

Select PLL_F80M as the default choice

enum **soc_periph_rmt_clk_src_legacy_t**

Type of RMT clock source, reserved for the legacy RMT driver.

Values:

enumerator **RMT_BASECLK_PLL_F80M**

RMT source clock is PLL_F80M

enumerator **RMT_BASECLK_XTAL**

RMT source clock is XTAL

enumerator **RMT_BASECLK_DEFAULT**

RMT source clock default choice is PLL_F80M

enum **soc_periph_uart_clk_src_legacy_t**

Type of UART clock source, reserved for the legacy UART driver.

Values:

enumerator **UART_SCLK_PLL_F80M**

UART source clock is PLL_F80M

enumerator **UART_SCLK_RTC**

UART source clock is RC_FAST

enumerator **UART_SCLK_XTAL**

UART source clock is XTAL

enumerator **UART_SCLK_DEFAULT**

UART source clock default choice is PLL_F80M

enum **soc_periph_lp_uart_clk_src_t**

Type of LP_UART clock source.

Values:

enumerator **LP_UART_SCLK_LP_FAST**

LP_UART source clock is LP(RTC)_FAST

enumerator **LP_UART_SCLK_XTAL_D2**

LP_UART source clock is XTAL_D2

enumerator **LP_UART_SCLK_DEFAULT**

LP_UART source clock default choice is LP(RTC)_FAST

enum **soc_periph_mcpwm_timer_clk_src_t**

Type of MCPWM timer clock source.

Values:

enumerator **MCPWM_TIMER_CLK_SRC_PLL160M**

Select PLL_F160M as the source clock

enumerator **MCPWM_TIMER_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **MCPWM_TIMER_CLK_SRC_DEFAULT**

Select PLL_F160M as the default choice

enum **soc_periph_mcpwm_capture_clk_src_t**

Type of MCPWM capture clock source.

Values:

enumerator **MCPWM_CAPTURE_CLK_SRC_PLL160M**

Select PLL_F160M as the source clock

enumerator **MCPWM_CAPTURE_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **MCPWM_CAPTURE_CLK_SRC_DEFAULT**

Select PLL_F160M as the default choice

enum **soc_periph_mcpwm_carrier_clk_src_t**

Type of MCPWM carrier clock source.

Values:

enumerator **MCPWM_CARRIER_CLK_SRC_PLL160M**

Select PLL_F160M as the source clock

enumerator **MCPWM_CARRIER_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **MCPWM_CARRIER_CLK_SRC_DEFAULT**

Select PLL_F160M as the default choice

enum **soc_periph_i2s_clk_src_t**

I2S clock source enum.

Values:

enumerator **I2S_CLK_SRC_DEFAULT**

Select XTAL as the default source clock

enumerator **I2S_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **I2S_CLK_SRC_APLL**

Select APLL as the source clock

enumerator **I2S_CLK_SRC_EXTERNAL**

Select external clock as source clock

enum **soc_periph_lp_i2s_clk_src_t**

LP I2S clock source enum.

Values:

enumerator **LP_I2S_CLK_SRC_DEFAULT**

Select LP_PERI as the default source clock

enumerator **LP_I2S_CLK_SRC_LP_PERI**

Select LP_PERI as the source clock

enumerator **LP_I2S_CLK_SRC_XTAL_D2**

LP_I2S source clock is XTAL_D2

enum **soc_periph_lcd_clk_src_t**

Type of LCD clock source.

Values:

enumerator **LCD_CLK_SRC_PLL160M**

Select PLL_F160M as the source clock

enumerator **LCD_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **LCD_CLK_SRC_APLL**

Select APLL as the source clock

enumerator **LCD_CLK_SRC_DEFAULT**

Select PLL_F160M as the default choice

enum **soc_periph_cam_clk_src_t**

Type of CAM clock source.

Values:

enumerator **CAM_CLK_SRC_PLL160M**

Select PLL_F160M as the source clock

enumerator **CAM_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **CAM_CLK_SRC_APLL**

Select APLL as the source clock

enumerator **CAM_CLK_SRC_DEFAULT**

Select PLL_F160M as the default choice

enum **soc_periph_mipi_csi_phy_clk_src_t**

Type of MIPI CSI PHY clock source.

Values:

enumerator **MIPI_CSI_PHY_CLK_SRC_RC_FAST**

Select RC_FAST as MIPI CSI PHY source clock

enumerator **MIPI_CSI_PHY_CLK_SRC_PLL_F25M**

Select PLL_F25M as MIPI CSI PHY source clock

enumerator **MIPI_CSI_PHY_CLK_SRC_PLL_F20M**

Select PLL_F20M as MIPI CSI PHY source clock

enumerator **MIPI_CSI_PHY_CLK_SRC_DEFAULT**

Select PLL_F20M as default clock

enum **soc_periph_mipi_dsi_dpi_clk_src_t**

Type of MIPI DSI DPI clock source.

Values:

enumerator **MIPI_DSI_DPI_CLK_SRC_XTAL**

Select XTAL as MIPI DSI DPI source clock

enumerator **MIPI_DSI_DPI_CLK_SRC_PLL_F160M**

Select PLL_F160M as MIPI DSI DPI source clock

enumerator **MIPI_DSI_DPI_CLK_SRC_PLL_F240M**

Select PLL_F240M as MIPI DSI DPI source clock

enumerator **MIPI_DSI_DPI_CLK_SRC_DEFAULT**

Select PLL_F240M as default clock

enum **soc_periph_mipi_dsi_phy_clk_src_t**

Type of MIPI DSI PHY clock source.

Values:

enumerator **MIPI_DSI_PHY_CLK_SRC_RC_FAST**

Select RC_FAST as MIPI DSI PHY source clock

enumerator **MIPI_DSI_PHY_CLK_SRC_PLL_F25M**

Select PLL_F25M as MIPI DSI PHY source clock

enumerator **MIPI_DSI_PHY_CLK_SRC_PLL_F20M**

Select PLL_F20M as MIPI DSI PHY source clock

enumerator **MIPI_DSI_PHY_CLK_SRC_DEFAULT**

Select PLL_F20M as default clock

enum **soc_periph_i2c_clk_src_t**

Type of I2C clock source.

Values:

enumerator **I2C_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **I2C_CLK_SRC_RC_FAST**

Select RC_FAST as the source clock

enumerator **I2C_CLK_SRC_DEFAULT**

Select XTAL as the default source clock

enum **soc_periph_lp_i2c_clk_src_t**

Type of LP_I2C clock source.

Values:

enumerator **LP_I2C_SCLK_LP_FAST**

LP_I2C source clock is RTC_FAST

enumerator **LP_I2C_SCLK_XTAL_D2**

LP_I2C source clock is XTAL_D2

enumerator **LP_I2C_SCLK_DEFAULT**

LP_I2C source clock default choice is RTC_FAST

enum **soc_periph_spi_clk_src_t**

Type of SPI clock source.

Values:

enumerator **SPI_CLK_SRC_XTAL**

Select XTAL as SPI source clock

enumerator **SPI_CLK_SRC_RC_FAST**

Select RC_FAST_20M as SPI source clock

enumerator **SPI_CLK_SRC_SPLL**

Select SPLL as SPI source clock

enumerator **SPI_CLK_SRC_DEFAULT**

Select SPLL as SPI source clock

enum **soc_periph_psram_clk_src_t**

Type of PSRAM clock source.

Values:

enumerator **PSRAM_CLK_SRC_DEFAULT**

Select SOC_MOD_CLK_SPLL as PSRAM source clock

enumerator **PSRAM_CLK_SRC_XTAL**

Select SOC_MOD_CLK_XTAL as PSRAM source clock

enumerator **PSRAM_CLK_SRC_CPLL**

Select SOC_MOD_CLK_CPLL as PSRAM source clock

enumerator **PSRAM_CLK_SRC_SPLL**

Select SOC_MOD_CLK_SPLL as PSRAM source clock

enumerator **PSRAM_CLK_SRC_MPLL**

Select SOC_MOD_CLK_MPLL as PSRAM source clock

enum **soc_periph_flash_clk_src_t**

Type of FLASH clock source.

Values:

enumerator **FLASH_CLK_SRC_DEFAULT**

Select SOC_MOD_CLK_SPLL as FLASH source clock

enumerator **FLASH_CLK_SRC_XTAL**

Select SOC_MOD_CLK_XTAL as FLASH source clock

enumerator **FLASH_CLK_SRC_CPLL**

Select SOC_MOD_CLK_CPLL as FLASH source clock

enumerator **FLASH_CLK_SRC_SPLL**

Select SOC_MOD_CLK_SPLL as FLASH source clock

enum **soc_periph_isp_clk_src_t**

Type of ISP clock source.

Values:

enumerator **ISP_CLK_SRC_DEFAULT**

Select SOC_MOD_CLK_PLL_F160M as ISP source clock

enumerator **ISP_CLK_SRC_XTAL**

Select SOC_MOD_CLK_XTAL as ISP source clock

enumerator **ISP_CLK_SRC_PLL160**

Select SOC_MOD_CLK_PLL_F160M as ISP source clock

enumerator **ISP_CLK_SRC_PLL240**

Select SOC_MOD_CLK_PLL_F240M as ISP source clock

enum **soc_periph_sdm_clk_src_t**

Sigma Delta Modulator clock source.

Values:

enumerator **SDM_CLK_SRC_XTAL**

Select XTAL clock as the source clock

enumerator **SDM_CLK_SRC_PLL_F80M**

Select PLL_F80M clock as the source clock

enumerator **SDM_CLK_SRC_DEFAULT**

Select PLL_F80M clock as the default clock choice

enum **soc_periph_glitch_filter_clk_src_t**

Glitch filter clock source.

Values:

enumerator **GLITCH_FILTER_CLK_SRC_XTAL**

Select XTAL clock as the source clock

enumerator **GLITCH_FILTER_CLK_SRC_PLL_F80M**

Select PLL_F80M clock as the source clock

enumerator **GLITCH_FILTER_CLK_SRC_DEFAULT**

Select PLL_F80M clock as the default clock choice

enum **soc_periph_ana_cmpr_clk_src_t**

Analog Comparator clock source.

Values:

enumerator **ANA_CMPR_CLK_SRC_XTAL**

Select XTAL clock as the source clock

enumerator **ANA_CMPR_CLK_SRC_PLL_F80M**

Select PLL_F80M clock as the source clock

enumerator **ANA_CMPR_CLK_SRC_DEFAULT**

Select PLL_F80M as the default clock choice

enum **soc_periph_twai_clk_src_t**

TWAI clock source.

Values:

enumerator **TWAI_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **TWAI_CLK_SRC_RC_FAST**

Select RC_FAST as the source clock

enumerator **TWAI_CLK_SRC_DEFAULT**

Select XTAL as the default clock choice

enum **soc_periph_adc_digi_clk_src_t**

ADC digital controller clock source.

Values:

enumerator **ADC_DIGI_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **ADC_DIGI_CLK_SRC_RC_FAST**

Select RC_FAST as the source clock

enumerator **ADC_DIGI_CLK_SRC_PLL_F80M**

Select PLL_F80M as the source clock

enumerator **ADC_DIGI_CLK_SRC_DEFAULT**

Select PLL_F80M as the default clock choice

enum **soc_periph_adc_rtc_clk_src_t**

ADC RTC controller clock source.

Values:

enumerator **ADC_RTC_CLK_SRC_RC_FAST**

Select RC_FAST as the source clock

enumerator **ADC_RTC_CLK_SRC_DEFAULT**

Select RC_FAST as the default clock choice

enum **soc_periph_mwdt_clk_src_t**

MWDT clock source.

Values:

enumerator **MWDT_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **MWDT_CLK_SRC_PLL_F80M**

Select PLL fixed 80 MHz as the source clock

enumerator **MWDT_CLK_SRC_RC_FAST**

Select RTC fast as the source clock

enumerator **MWDT_CLK_SRC_DEFAULT**

Select XTAL 40 MHz as the default clock choice

enum **soc_periph_ledc_clk_src_legacy_t**

Type of LEDC clock source, reserved for the legacy LEDC driver.

Values:

enumerator **LEDC_AUTO_CLK**

LEDC source clock will be automatically selected based on the giving resolution and duty parameter when init the timer

enumerator **LEDC_USE_XTAL_CLK**

Select XTAL as the source clock

enumerator **LEDC_USE_PLL_DIV_CLK**

Select PLL_F80M clock as the source clock

enumerator **LEDC_USE_RC_FAST_CLK**

Select RC_FAST as the source clock

enum **soc_periph_parlio_clk_src_t**

PARLIO clock source.

Values:

enumerator **PARLIO_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **PARLIO_CLK_SRC_RC_FAST**

Select RC_FAST as the source clock

enumerator **PARLIO_CLK_SRC_PLL_F160M**

Select PLL_F160M as the source clock

enumerator **PARLIO_CLK_SRC_EXTERNAL**

Select EXTERNAL clock as the source clock

enumerator **PARLIO_CLK_SRC_DEFAULT**

Select PLL_F160M as the default clock choice

enum **soc_periph_sdmmc_clk_src_t**

Type of SDMMC clock source.

Values:

enumerator **SDMMC_CLK_SRC_DEFAULT**

Select PLL_160M as the default choice

enumerator **SDMMC_CLK_SRC_PLL160M**

Select PLL_160M as the source clock

enum **soc_periph_temperature_sensor_clk_src_t**

Type of Temp Sensor clock source.

Values:

enumerator **TEMPERATURE_SENSOR_CLK_SRC_LP_PERI**

Select LP_PERI as the source clock

enumerator **TEMPERATURE_SENSOR_CLK_SRC_DEFAULT**

Select LP_PERI as the default choice

enum **soc_clkout_sig_id_t**

Values:

enumerator **CLKOUT_SIG_MPLL**

MPLL is from 40MHz XTAL oscillator frequency multipliers

enumerator **CLKOUT_SIG_SPLL**

SPLL is from 40MHz XTAL oscillator frequency multipliers, it has a ”fixed” frequency of 480MHz

enumerator **CLKOUT_SIG_CPLL**

CPLL_CLK is the output of 40MHz crystal oscillator frequency multiplier, can be 320/360/400MHz

enumerator **CLKOUT_SIG_XTAL**

External 40MHz crystal

enumerator **CLKOUT_SIG_RC_FAST**

Internal 17.5MHz RC oscillator

enumerator **CLKOUT_SIG_RC_SLOW**

Internal 136kHz RC oscillator

enumerator **CLKOUT_SIG_RC_32K**

Internal 32kHz RC oscillator

enumerator **CLKOUT_SIG_XTAL32K**

External 32kHz crystal clock

enumerator **CLKOUT_SIG_I2S0**

I2S0 clock, depends on the i2s driver configuration

enumerator **CLKOUT_SIG_I2S1**

I2S1 clock, depends on the i2s driver configuration

enumerator **CLKOUT_SIG_I2S2**

I2S2 clock, depends on the i2s driver configuration

enumerator **CLKOUT_SIG_CPU**

CPU clock

enumerator **CLKOUT_SIG_MEM**

MEM clock

enumerator **CLKOUT_SIG_SYS**

SYS clock

enumerator **CLKOUT_SIG_APB**

APB clock

enumerator **CLKOUT_SIG_PLL_F80M**

From PLL, usually be 80MHz

enumerator **CLKOUT_SIG_INVALID**

Header File

- components/esp_hw_support/include/esp_clk_tree.h
- This header file can be included with:

```
#include "esp_clk_tree.h"
```

Functions

esp_err_t **esp_clk_tree_src_get_freq_hz**(*soc_module_clk_t* clk_src, *esp_clk_tree_src_freq_precision_t* precision, uint32_t *freq_value)

Get frequency of module clock source.

参数

- **clk_src** -- **[in]** Clock source available to modules, in soc_module_clk_t
- **precision** -- **[in]** Degree of precision, one of esp_clk_tree_src_freq_precision_t values This arg only applies to the clock sources that their frequencies can vary: SOC_MOD_CLK_RTC_FAST, SOC_MOD_CLK_RTC_SLOW, SOC_MOD_CLK_RC_FAST, SOC_MOD_CLK_RC_FAST_D256, SOC_MOD_CLK_XTAL32K For other clock sources, this field is ignored.

- **freq_value** -- **[out]** Frequency of the clock source, in Hz

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Calibration failed

Enumerations

enum **esp_clk_tree_src_freq_precision_t**

Degree of precision of frequency value to be returned by esp_clk_tree_src_get_freq_hz()

Values:

enumerator **ESP_CLK_TREE_SRC_FREQ_PRECISION_CACHED**

enumerator **ESP_CLK_TREE_SRC_FREQ_PRECISION_APPROX**

enumerator **ESP_CLK_TREE_SRC_FREQ_PRECISION_EXACT**

enumerator **ESP_CLK_TREE_SRC_FREQ_PRECISION_INVALID**

2.5.6 事件任务矩阵 (ETM)

简介

如果外设 X 需要向外设 Y 发起事件通知，一般只能通过 CPU 中断实现。在此过程中，CPU 会代表外设 X，给外设 Y 发送通知。然而，在对时间敏感的应用程序中，CPU 中断引发的延迟不容忽视。

通过引入事件任务矩阵 (ETM) 模块，部分外设可以直接通过预先设置的连接关系，将事件通知发送给其他外设，无需 CPU 中断介入。由此，外设实现精确、低延迟同步，并减轻 CPU 负担。

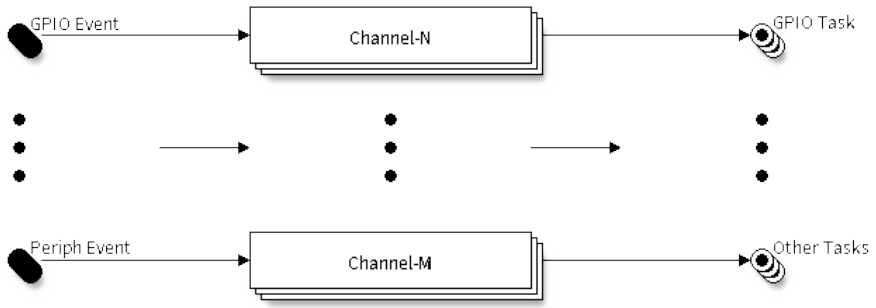


图 4: ETM 通道概述

ETM 模块具有多个通道，这些通道支持用户根据需要进行配置，连接特定 **事件**与 **任务**。激活某一事件时，ETM 将自动触发相应任务。

支持 ETM 功能的外设向 ETM 提供其事件和任务的连接关系。ETM 通道可以连接任意事件和任务，事件和任务甚至可以来自于同一个外设。然而，对于一个 ETM 通道，一次只能将一个事件与一个任务连接（即 1 对 1 关系）。如果要使用不同事件触发同一任务，则需申请多条 ETM 通道。

使用 ETM 通常可实现以下功能：

- 当定时器报警事件发生时，翻转 GPIO 电平
- 当在 GPIO 上监测到脉冲边沿时，启动 ADC 转换

功能概述

下文将分节概述 ETM 的功能，并介绍配置和使用 ETM 模块的基本步骤：

- *ETM 通道分配* - 介绍如何安装和卸载 ETM 通道。
- *ETM 事件* - 介绍如何分配新的 ETM 事件句柄，以及如何从不同外设获取现有句柄。
- *ETM 任务* - 介绍如何分配新的 ETM 任务句柄，以及如何从不同外设获取现有句柄。
- *ETM 通道控制* - 介绍常见的 ETM 通道控制函数。
- *线程安全* - 列出了驱动程序中始终线程安全的 API。
- *Kconfig 选项* - 列出了 ETM 支持的 Kconfig 选项，这些选项对驱动程序的行为会产生不同影响。

ETM 通道分配 在 ESP32-P4 中，存在许多相同的 ETM 通道[1]，各通道在软件中由 `esp_etm_channel_handle_t` 表示。可用硬件资源汇集在资源池内，由 ETM 核心驱动程序管理，无需手动管理通道的分配和释放。ETM 核心驱动程序会在调用 `esp_etm_new_channel()` 时自动分配通道，在调用 `esp_etm_del_channel()` 时删除通道。分配通道的要求通过 `esp_etm_channel_config_t` 配置。

在删除 ETM 通道前，请调用 `esp_etm_channel_disable()` 禁用要删除的通道，或确保该通道尚未由 `esp_etm_channel_enable()` 启用，再继续删除操作。

ETM 事件 ETM 事件对其事件源进行了抽象，屏蔽了具体事件源的细节，并在软件中表示为 `esp_etm_event_handle_t`，使应用程序可以更便捷地处理不同类型的事件。ETM 事件可以由各种外设产生，因此获取事件句柄的方法因外设而异。当不再需要某个事件时，请调用 `esp_etm_channel_connect()`，并传递一个 `NULL` 事件句柄，断开与事件的连接，随后调用 `esp_etm_del_event()`，释放事件资源。

GPIO 事件 GPIO **边沿**事件是最常见的事件类型，任何 GPIO 管脚均可触发这类事件。要创建 GPIO 事件句柄，请调用 `gpio_new_etm_event()`，并使用 `gpio_etm_event_config_t` 提供的配置信息：

- `gpio_etm_event_config_t::edge` 或 `gpio_etm_event_config_t::edges` 决定触发事件的边沿类型，支持的边沿类型已在 `gpio_etm_event_edge_t` 中列出。

接下来，请调用 `gpio_etm_event_bind_gpio()` 函数，连接 GPIO ETM 事件句柄与 GPIO 管脚。注意，要设置 GPIO 管脚，只能使用由 `gpio_new_etm_event()` 函数创建的 ETM 事件句柄。对于其他类型的 ETM 事件，调用此函数，将返回 `ESP_ERR_INVALID_ARG` 错误。该函数也无法完成 GPIO 的初始化，在使用 GPIO ETM 事件之前，仍需调用 `gpio_config()` 函数，设置 GPIO 管脚的属性，如方向、高/低电平模式等。

其他外设事件

- 调用 `esp_systick_new_etm_alarm_event()` 可以从 RTOS Systick 获取 ETM 事件句柄，每个 CPU 核心可以获取一个事件句柄。
- 要了解如何从 esp_timer 获取 ETM 事件句柄，请参阅 *ESP 定时器*。
- 要了解如何从 GPTimer 获取 ETM 事件句柄，请参阅 *通用定时器*。
- 要了解如何从 async memcpy 获取 ETM 事件句柄，请参阅 *异步内存复制*。
- 要了解如何从 MCPWM 中获取 ETM 事件句柄，请参阅 *电机控制脉宽调制器 (MCPWM)*。
- 要了解如何从模拟比较器获取 ETM 事件句柄，请参阅 *Analog Comparator*。

[1] 不同 ESP 芯片系列的 ETM 通道数量可能不同。要了解更多详情，请参阅 **ESP32-P4 技术参考手册 > 事件任务矩阵 (ETM)** [PDF]。驱动程序对通道申请数量不做限制，但当硬件资源用尽时，驱动程序将返回错误。因此，每次进行通道分配（即调用 `esp_etm_new_channel()`）时，请注意检查返回值。

ETM 任务 ETM 任务对其操作进行了抽象，在软件中表示为*esp_etm_task_handle_t*，使任务得以用同一方式管理和表示。ETM 任务可以分配给不同外设，因此获取任务句柄的方式因外设而异。当不再需要某个任务时，请调用*esp_etm_channel_connect()*，并传递一个 NULL 事件句柄，断开与任务的连接，随后调用*esp_etm_del_event()*，释放任务资源。

GPIO 任务 GPIO 任务是最常见的任务类型。一个 GPIO 可以采取一个或多个 GPIO 操作，而一个 GPIO 任务也可以同时管理多个 GPIO 管脚。当 ETM 通道激活任务时，任务可以同时设置管理的所有 GPIO 引脚，使其设置/清除/切换状态。要创建 GPIO 任务句柄，请调用*gpio_new_etm_task()*，并使用*gpio_etm_task_config_t* 提供的配置信息：

- *gpio_etm_task_config_t::action* 或*gpio_etm_task_config_t::actions* 决定 ETM 任务将采取的 GPIO 操作，支持的操作类型在*gpio_etm_task_action_t* 中列出。如果一个 GPIO 需要采取多个 GPIO 操作，这些操作任务的创建必须通过配置*gpio_etm_task_config_t::actions* 的数组并在一次*gpio_new_etm_task()* 调用中一并完成。

接下来，需要连接 GPIO ETM 任务句柄与 GPIO 管脚。为此，请调用*gpio_etm_task_add_gpio()* 函数。如果需要任务句柄管理更多的 GPIO 管脚，可以重复调用以上函数，注意，要设置 GPIO 管脚，只能使用由*gpio_new_etm_task()* 函数创建的 ETM 任务句柄。对于其他类型的 ETM 任务，调用此函数，将返回*ESP_ERR_INVALID_ARG* 错误。该函数也无法完成 GPIO 的初始化，在使用 GPIO ETM 任务之前，仍需调用*gpio_config()* 函数，设置 GPIO 管脚的属性，如方向、高/低电平模式等。

要删除 GPIO ETM 任务，请调用*esp_etm_del_task()*。在此之前，请确保已经调用过*gpio_etm_task_rm_gpio()*，删除了所有先前添加的 GPIO 管脚。

其他外设任务

- 要了解如何从 GPTimer 获取 ETM 任务句柄，请参阅通用定时器。

ETM 通道控制

映射事件与任务 在调用*esp_etm_channel_connect()* 将它们连接到同一个 ETM 通道之前，ETM 事件与 ETM 任务之间没有任何映射关系。注意，使用 NULL 任务/事件句柄调用该函数时，会将通道与任何任务或事件解除映射。此函数可以在通道启用之前或之后调用，但在运行时调用此函数更改映射关系存在一定风险，因为此时通道可能正处于周期的中间阶段，新的映射可能无法立即生效。

启用及禁用通道 调用*esp_etm_channel_enable()* 启用 ETM 通道，调用*esp_etm_channel_disable()* 禁用 ETM 通道。

ETM 通道分析 要检查是否为 ETM 通道设置了正确的事件和任务，可以调用*esp_etm_dump()*，输出所有工作中的 ETM 通道及其关联的事件和任务。输出格式如下：

```
==========ETM Dump Start==========
channel 0: event 48 ==> task 17
channel 1: event 48 ==> task 90
channel 2: event 48 ==> task 94
==========ETM Dump End============
```

以上输出信息打印的数字 ID 在 soc/soc_etm_source.h 文件中定义。

线程安全 ETM 驱动程序会确保工厂函数*esp_etm_new_channel()* 和*gpio_new_etm_task()* 的线程安全。使用时，可以直接从不同的 RTOS 任务中调用此类函数，无需额外锁保护。

在 ISR 环境中，不支持运行任何函数。

其他以*esp_etm_channel_handle_t*、*esp_etm_task_handle_t* 和*esp_etm_event_handle_t* 作为首个位置参数的函数，则非线程安全，应避免从不同任务中调用此类函数。

Kconfig 选项

- *CONFIG_ETM_ENABLE_DEBUG_LOG* 用于启用调试日志输出，启用此选项将增加固件的二进制文件大小。

API 参考

Header File

- components/esp_hw_support/include/esp_etm.h
- This header file can be included with:

```
#include "esp_etm.h"
```

Functions

esp_err_t **esp_etm_new_channel**(const *esp_etm_channel_config_t* *config, *esp_etm_channel_handle_t* *ret_chan)

Allocate an ETM channel.

备注： The channel can later be freed by `esp_etm_del_channel`

参数

- **config** -- **[in]** ETM channel configuration
- **ret_chan** -- **[out]** Returned ETM channel handle

返回

- ESP_OK: Allocate ETM channel successfully
- ESP_ERR_INVALID_ARG: Allocate ETM channel failed because of invalid argument
- ESP_ERR_NO_MEM: Allocate ETM channel failed because of out of memory
- ESP_ERR_NOT_FOUND: Allocate ETM channel failed because all channels are used up and no more free one
- ESP_FAIL: Allocate ETM channel failed because of other reasons

esp_err_t **esp_etm_del_channel**(*esp_etm_channel_handle_t* chan)

Delete an ETM channel.

参数 **chan** -- **[in]** ETM channel handle that created by `esp_etm_new_channel`

返回

- ESP_OK: Delete ETM channel successfully
- ESP_ERR_INVALID_ARG: Delete ETM channel failed because of invalid argument
- ESP_FAIL: Delete ETM channel failed because of other reasons

esp_err_t **esp_etm_channel_enable**(*esp_etm_channel_handle_t* chan)

Enable ETM channel.

备注： This function will transit the channel state from init to enable.

参数 **chan** -- **[in]** ETM channel handle that created by `esp_etm_new_channel`

返回

- ESP_OK: Enable ETM channel successfully
- ESP_ERR_INVALID_ARG: Enable ETM channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable ETM channel failed because the channel has been enabled already
- ESP_FAIL: Enable ETM channel failed because of other reasons

esp_err_t **esp_etm_channel_disable**(*esp_etm_channel_handle_t* chan)

Disable ETM channel.

备注： This function will transit the channel state from enable to init.

参数 **chan** -- **[in]** ETM channel handle that created by `esp_etm_new_channel`

返回

- ESP_OK: Disable ETM channel successfully
- ESP_ERR_INVALID_ARG: Disable ETM channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable ETM channel failed because the channel is not enabled yet
- ESP_FAIL: Disable ETM channel failed because of other reasons

esp_err_t **esp_etm_channel_connect**(*esp_etm_channel_handle_t* chan, *esp_etm_event_handle_t* event, *esp_etm_task_handle_t* task)

Connect an ETM event to an ETM task via a previously allocated ETM channel.

备注： Setting the ETM event/task handle to NULL means to disconnect the channel from any event/task

参数

- **chan** -- **[in]** ETM channel handle that created by `esp_etm_new_channel`
- **event** -- **[in]** ETM event handle obtained from a driver/peripheral, e.g. `xxx_new_etm_event`
- **task** -- **[in]** ETM task handle obtained from a driver/peripheral, e.g. `xxx_new_etm_task`

返回

- ESP_OK: Connect ETM event and task to the channel successfully
- ESP_ERR_INVALID_ARG: Connect ETM event and task to the channel failed because of invalid argument
- ESP_FAIL: Connect ETM event and task to the channel failed because of other reasons

esp_err_t **esp_etm_del_event**(*esp_etm_event_handle_t* event)

Delete ETM event.

备注： Although the ETM event comes from various peripherals, we provide the same user API to delete the event handle seamlessly.

参数 **event** -- **[in]** ETM event handle obtained from a driver/peripheral, e.g. `xxx_new_etm_event`

返回

- ESP_OK: Delete ETM event successfully
- ESP_ERR_INVALID_ARG: Delete ETM event failed because of invalid argument
- ESP_FAIL: Delete ETM event failed because of other reasons

esp_err_t **esp_etm_del_task**(*esp_etm_task_handle_t* task)

Delete ETM task.

备注： Although the ETM task comes from various peripherals, we provide the same user API to delete the task handle seamlessly.

参数 **task** -- **[in]** ETM task handle obtained from a driver/peripheral, e.g. `xxx_new_etm_task`

返回

- ESP_OK: Delete ETM task successfully
- ESP_ERR_INVALID_ARG: Delete ETM task failed because of invalid argument
- ESP_FAIL: Delete ETM task failed because of other reasons

esp_err_t **esp_etm_dump**(FILE *out_stream)

Dump ETM channel usages to the given IO stream.

参数 **out_stream** -- **[in]** IO stream (e.g. stdout)

返回

- ESP_OK: Dump ETM channel usages successfully
- ESP_ERR_INVALID_ARG: Dump ETM channel usages failed because of invalid argument
- ESP_FAIL: Dump ETM channel usages failed because of other reasons

Structures

struct **esp_etm_channel_config_t**

ETM channel configuration.

Type Definitions

typedef struct esp_etm_channel_t ***esp_etm_channel_handle_t**

ETM channel handle.

typedef struct esp_etm_event_t ***esp_etm_event_handle_t**

ETM event handle.

typedef struct esp_etm_task_t ***esp_etm_task_handle_t**

ETM task handle.

Header File

- components/esp_driver_gpio/include/driver/gpio_etm.h
- This header file can be included with:

```
#include "driver/gpio_etm.h"
```

- This header file is a part of the API provided by the `esp_driver_gpio` component. To declare that your component depends on `esp_driver_gpio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gpio
```

or

```
PRIV_REQUIRES esp_driver_gpio
```

Functions

esp_err_t **gpio_new_etm_event**(const *gpio_etm_event_config_t* *config, *esp_etm_event_handle_t* *ret_event, ...)

Create an ETM event object for the GPIO peripheral.

备注: The created ETM event object can be deleted later by calling `esp_etm_del_event`

备 注: The newly created ETM event object is not bind to any GPIO, you need to call `gpio_etm_event_bind_gpio` to bind the wanted GPIO

备注: Every success call to this function will acquire a free GPIO ETM event channel

参数

- **`config`** -- **[in]** GPIO ETM event configuration
- **`ret_event`** -- **[out]** Returned ETM event handle
- **`...`** -- **[out]** Other returned ETM event handles if any (the order of the returned event handles is aligned with the array order in field `edges` in *`gpio_etm_event_config_t`*)

返回

- ESP_OK: Create ETM event successfully
- ESP_ERR_INVALID_ARG: Create ETM event failed because of invalid argument
- ESP_ERR_NO_MEM: Create ETM event failed because of out of memory
- ESP_ERR_NOT_FOUND: Create ETM event failed because all events are used up and no more free one
- ESP_FAIL: Create ETM event failed because of other reasons

esp_err_t **`gpio_etm_event_bind_gpio`**(*esp_etm_event_handle_t* event, int gpio_num)

Bind the GPIO with the ETM event.

备注: Calling this function multiple times with different GPIO number can override the previous setting immediately.

备注: Only GPIO ETM object can call this function

参数

- **`event`** -- **[in]** ETM event handle that created by `gpio_new_etm_event`
- **`gpio_num`** -- **[in]** GPIO number that can trigger the ETM event

返回

- ESP_OK: Set the GPIO for ETM event successfully
- ESP_ERR_INVALID_ARG: Set the GPIO for ETM event failed because of invalid argument, e.g. GPIO is not input capable, ETM event is not of GPIO type
- ESP_FAIL: Set the GPIO for ETM event failed because of other reasons

esp_err_t **`gpio_new_etm_task`**(const *gpio_etm_task_config_t* *config, *esp_etm_task_handle_t* *ret_task, ...)

Create an ETM task object for the GPIO peripheral.

备注: The created ETM task object can be deleted later by calling `esp_etm_del_task`

备注: The GPIO ETM task works like a container, a newly created ETM task object doesn't have GPIO members to be managed. You need to call `gpio_etm_task_add_gpio` to put one or more GPIOs to the container.

备注: Every success call to this function will acquire a free GPIO ETM task channel

参数

- **config** -- **[in]** GPIO ETM task configuration
- **ret_task** -- **[out]** Returned ETM task handle
- ... -- **[out]** Other returned ETM task handles if any (the order of the returned task handles is aligned with the array order in field actions in *gpio_etm_task_config_t*)

返回

- ESP_OK: Create ETM task successfully
- ESP_ERR_INVALID_ARG: Create ETM task failed because of invalid argument
- ESP_ERR_NO_MEM: Create ETM task failed because of out of memory
- ESP_ERR_NOT_FOUND: Create ETM task failed because all tasks are used up and no more free one
- ESP_FAIL: Create ETM task failed because of other reasons

esp_err_t **gpio_etm_task_add_gpio**(*esp_etm_task_handle_t* task, int gpio_num)

Add GPIO to the ETM task.

备注: You can call this function multiple times to add more GPIOs

备注: Only GPIO ETM object can call this function

参数

- **task** -- **[in]** ETM task handle that created by gpio_new_etm_task
- **gpio_num** -- **[in]** GPIO number that can be controlled by the ETM task

返回

- ESP_OK: Add GPIO to the ETM task successfully
- ESP_ERR_INVALID_ARG: Add GPIO to the ETM task failed because of invalid argument, e.g. GPIO is not output capable, ETM task is not of GPIO type
- ESP_ERR_INVALID_STATE: Add GPIO to the ETM task failed because the GPIO is used by other ETM task already
- ESP_FAIL: Add GPIO to the ETM task failed because of other reasons

esp_err_t **gpio_etm_task_rm_gpio**(*esp_etm_task_handle_t* task, int gpio_num)

Remove the GPIO from the ETM task.

备注: Before deleting the ETM task, you need to remove all the GPIOs from the ETM task by this function

备注: Only GPIO ETM object can call this function

参数

- **task** -- **[in]** ETM task handle that created by gpio_new_etm_task
- **gpio_num** -- **[in]** GPIO number that to be remove from the ETM task

返回

- ESP_OK: Remove the GPIO from the ETM task successfully
- ESP_ERR_INVALID_ARG: Remove the GPIO from the ETM task failed because of invalid argument
- ESP_ERR_INVALID_STATE: Remove the GPIO from the ETM task failed because the GPIO is not controlled by this ETM task
- ESP_FAIL: Remove the GPIO from the ETM task failed because of other reasons

Structures

struct **gpio_etm_event_config_t**

GPIO ETM event configuration.

If more than one kind of ETM edge event want to be triggered on the same GPIO pin, you can configure them together. It helps to save GPIO ETM event channel resources for other GPIOs.

Public Members

gpio_etm_event_edge_t **edge**

Which kind of edge can trigger the ETM event module

gpio_etm_event_edge_t **edges**[GPIO_ETM_EVENT_EDGE_TYPES]

Array of kinds of edges to trigger the ETM event module on the same GPIO

struct **gpio_etm_task_config_t**

GPIO ETM task configuration.

If multiple actions wants to be added to the same GPIO pin, you have to configure all the GPIO ETM tasks together.

Public Members

gpio_etm_task_action_t **action**

Action to take by the ETM task module

gpio_etm_task_action_t **actions**[GPIO_ETM_TASK_ACTION_TYPES]

Array of actions to take by the ETM task module on the same GPIO

Macros

GPIO_ETM_EVENT_EDGE_TYPES

GPIO ETM edge events are POS/NEG/ANY

GPIO_ETM_TASK_ACTION_TYPES

GPIO ETM action tasks are SET/CLEAR/TOGGLE

Enumerations

enum **gpio_etm_event_edge_t**

GPIO edges that can be used as ETM event.

Values:

enumerator **GPIO_ETM_EVENT_EDGE_POS**

A rising edge on the GPIO will generate an ETM event signal

enumerator **GPIO_ETM_EVENT_EDGE_NEG**

A falling edge on the GPIO will generate an ETM event signal

enumerator **GPIO_ETM_EVENT_EDGE_ANY**

Any edge on the GPIO can generate an ETM event signal

enum **gpio_etm_task_action_t**

GPIO actions that can be taken by the ETM task.

Values:

enumerator **GPIO_ETM_TASK_ACTION_SET**

Set the GPIO level to high

enumerator **GPIO_ETM_TASK_ACTION_CLR**

Clear the GPIO level to low

enumerator **GPIO_ETM_TASK_ACTION_TOG**

Toggle the GPIO level

Header File

- components/esp_system/include/esp_systick_etm.h
- This header file can be included with:

```
#include "esp_systick_etm.h"
```

Functions

esp_err_t **esp_systick_new_etm_alarm_event**(int core_id, *esp_etm_event_handle_t* *out_event)

Get the ETM event handle of systick hardware's alarm/heartbeat event.

备注: The created ETM event object can be deleted later by calling `esp_etm_del_event`

参数

- **core_id** -- **[in]** CPU core ID
- **out_event** -- **[out]** Returned ETM event handle

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

2.5.7 GPIO & RTC GPIO

GPIO 汇总

ESP32-P4 芯片具有 55 个物理 GPIO 管脚（GPIO0 ~ GPIO54）。每个管脚都可用作一个通用 IO，或连接一个内部的外设信号。通过 GPIO 交换矩阵和 IO MUX，可配置外设模块的输入信号来源于任何的 IO 管脚，并且外设模块的输出信号也可连接到任意 IO 管脚。这些模块共同组成了芯片的 IO 控制。更多详细信息，请参阅 *ESP32-P4 技术参考手册 > IO MUX 和 GPIO 矩阵（GPIO、IO_MUX）* [PDF]。

下表提供了各管脚的详细信息，部分 GPIO 具有特殊的使用限制，具体可参考表中的注释列。

GPIO	模拟功能	LP GPIO	注释
GPIO0		LP_GPIO0	
GPIO1		LP_GPIO1	
GPIO2	TOUCH0	LP_GPIO2	

下页继续

表 3 – 续上页

GPIO	模拟功能	LP GPIO	注释
GPIO3	TOUCH1	LP_GPIO3	
GPIO4	TOUCH2	LP_GPIO4	
GPIO5	TOUCH3	LP_GPIO5	
GPIO6	TOUCH4	LP_GPIO6	
GPIO7	TOUCH5	LP_GPIO7	
GPIO8	TOUCH6	LP_GPIO8	
GPIO9	TOUCH7	LP_GPIO9	
GPIO10	TOUCH8	LP_GPIO10	
GPIO11	TOUCH9	LP_GPIO11	
GPIO12	TOUCH10	LP_GPIO12	
GPIO13	TOUCH11	LP_GPIO13	
GPIO14	TOUCH12	LP_GPIO14	
GPIO15	TOUCH13	LP_GPIO15	
GPIO16	ADC1_CH0		
GPIO17	ADC1_CH1		
GPIO18	ADC1_CH2		
GPIO19	ADC1_CH3		
GPIO20	ADC1_CH4		
GPIO21	ADC1_CH5		
GPIO22	ADC1_CH6		
GPIO23	ADC1_CH7		
GPIO24			
GPIO25			
GPIO26			
GPIO27			
GPIO28			
GPIO29			
GPIO30			
GPIO31			
GPIO32			
GPIO33			
GPIO34			Strapping 管脚
GPIO35			Strapping 管脚
GPIO36			Strapping 管脚
GPIO37			Strapping 管脚
GPIO38			Strapping 管脚
GPIO39			
GPIO40			
GPIO41			
GPIO42			
GPIO43			
GPIO44			
GPIO45			
GPIO46			
GPIO47			
GPIO48			
GPIO49	ADC1_CH8		
GPIO50	ADC1_CH9		
GPIO51	ADC1_CH10, ANA_CMPR_CH0 外部参考电压		

下页继续

表 3 – 续上页

GPIO	模拟功能	LP GPIO	注释
GPIO52	ADC1_CH11，ANA_CMPR_CH0 同相输入		
GPIO53	ADC1_CH12，ANA_CMPR_CH1 外部参考电压		
GPIO54	ADC1_CH13，ANA_CMPR_CH1 同相输入		

备注:

- Strapping 管脚：GPIO34, GPIO35、GPIO36、GPIO37 和 GPIO38 是 Strapping 管脚。更多信息请参考 ESP32-P4 技术规格书。
- USB-JTAG：GPIO24 和 GPIO25 默认用于 USB-JTAG。如果将它们配置为普通 GPIO，驱动程序将禁用 USB-JTAG 功能。

- 处于 Deep-sleep 模式时
- 超低功耗协处理器 *(ULP-LP-Core)* 运行时
- 使用 ADC/DAC 等模拟功能时
- 使用低功耗外设时，例如：LP_UART ， LP_I2C 等

获取 IO 管脚实时配置状态

GPIO 驱动提供了一个函数*gpio_dump_io_configuration()* 用来输出指定管脚的实时配置状态，包括上下拉、输入输出使能、管脚映射等。例如，以下命令可用于输出 GPIO4，GPIO18 与 GPIO26 的配置状态：

```
gpio_dump_io_configuration(stdout, (1ULL << 4) | (1ULL << 18) | (1ULL << 26));
```

其输出信息如下：

```
================IO DUMP Start================
IO[4] -
  Pullup: 1, Pulldown: 0, DriveCap: 2
  InputEn: 1, OutputEn: 0, OpenDrain: 0
  FuncSel: 1 (GPIO)
  GPIO Matrix SigIn ID: (simple GPIO input)
  SleepSelEn: 1

IO[18] -
  Pullup: 0, Pulldown: 0, DriveCap: 2
  InputEn: 0, OutputEn: 1, OpenDrain: 0
  FuncSel: 1 (GPIO)
  GPIO Matrix SigOut ID: 256 (simple GPIO output)
  SleepSelEn: 1

IO[26] **RESERVED** -
  Pullup: 1, Pulldown: 0, DriveCap: 2
  InputEn: 1, OutputEn: 0, OpenDrain: 0
  FuncSel: 0 (IOMUX)
  SleepSelEn: 1

================IO DUMP End================
```

如果你想要查看所有管脚的配置状态，可以使用命令

```
gpio_dump_all_io_configuration(stdout, SOC_GPIO_VALID_GPIO_MASK);
```

如果 IO 管脚通过 GPIO 交换矩阵连接到内部外设信号，输出信息打印中的外设信号 ID 定义可以在 soc/esp32p4/include/soc/gpio_sig_map.h 头文件中查看。**RESERVED** 字样则表示此 IO 用于连接 SPI flash 或 PSRAM，强烈建议不要重新配置这些管脚用于其他功能。

请不要依赖技术参考手册中记录的 GPIO 默认配置状态，因为特殊用途的 GPIO 可能会在 app_main 之前被引导程序或应用程序启动阶段的代码更改。

配置 USB PHY 管脚为普通 GPIO 管脚

要将 USB PHY 管脚配置为普通 GPIO 管脚，可使用函数*gpio_config()*，请参考以下代码片段来进行配置。

```
gpio_config_t usb_phy_conf = {
    .pin_bit_mask = (1ULL << USB_PHY_DP_PIN) | (1ULL << USB_PHY_DM_PIN),
    .mode = GPIO_MODE_INPUT_OUTPUT,
    .pull_up_en = 0,
    .pull_down_en = 0,
    .intr_type = GPIO_INTR_DISABLE,
};
gpio_config(&usb_phy_conf);
```

GPIO 毛刺过滤器

ESP32-P4 内置硬件的过滤器可以过滤掉 GPIO 输入端口上的毛刺信号，在一定程度上避免错误触发中断或者是错把噪声当成有效的外设信号。

每个 GPIO 都可以使用独立的毛刺过滤器，该过滤器可以将那些脉冲宽度窄于 **2** 个采样时钟的信号剔除掉，该宽度无法配置。GPIO 对输入信号的采样时钟通常是 IO_MUX 的时钟源。在驱动中，此类过滤器称为 *管脚毛刺过滤器*。可以调用*gpio_new_pin_glitch_filter()* 函数创建一个过滤器句柄。过滤器的相关配置保存在*gpio_pin_glitch_filter_config_t* 结构体中。

- *gpio_pin_glitch_filter_config_t::gpio_num* 设置启用毛刺过滤器的 GPIO 编号。

ESP32-P4 提供了 8 个灵活的毛刺过滤器，被过滤信号的脉冲宽度可以由软件进行配置。此类过滤器则称为 *灵活毛刺过滤器*。每个过滤器都可以应用于任意 GPIO 输入，然而，将多个过滤器应用于同一 GPIO 上效果并不会叠加。可以调用*gpio_new_flex_glitch_filter()* 函数来创建一个过滤器句柄。过滤器的相关配置保存在*gpio_flex_glitch_filter_config_t* 结构体中。

- *gpio_flex_glitch_filter_config_t::gpio_num* 设置启用毛刺过滤器的 GPIO 编号。
- *gpio_flex_glitch_filter_config_t::window_width_ns* 和*gpio_flex_glitch_filter_config_t*
  是毛刺过滤器的关键参数。在:cpp:member:*gpio_flex_glitch_filter_config_t::window_width_ns* 时间内，任何脉冲信号，如果它的宽度小于*gpio_flex_glitch_filter_config_t::window_thres_ns*，那么该脉冲信号就会被滤除掉。*gpio_flex_glitch_filter_config_t::window_thres_ns* 的值不能大于*gpio_flex_glitch_filter_config_t::window_width_ns*。

请注意，*管脚毛刺过滤器*和 *灵活毛刺过滤器*是各自独立的，支持为同一 GPIO 同时启用这两种过滤器。

毛刺过滤器默认关闭，可调用*gpio_glitch_filter_enable()* 使能过滤器。如需回收这个过滤器，可以调用*gpio_del_glitch_filter()* 函数。在回收句柄前，请确保过滤器处于关闭状态，否则需调用*gpio_glitch_filter_disable()*。

GPIO 迟滞过滤器

ESP32-P4 支持输入引脚的硬件迟滞，这可以减少由于输入电压在逻辑 0、1 临界值附近时采样不稳定造成的 GPIO 中断误触，尤其是当输入逻辑电平转换较慢，电平建立时间较长时。

每个引脚可以独立启用迟滞功能。默认情况下，它处于关闭状态。你可以通过配置*gpio_config_t::hys_ctrl_mode* 来选择启用与否。迟滞控制模式会和其余 GPIO 配置一起在*gpio_config()* 中生效。

应用示例

- GPIO 输出和输入中断示例：peripherals/gpio/generic_gpio。

API 参考 - 普通 GPIO

Header File

- components/esp_driver_gpio/include/driver/gpio.h
- This header file can be included with:

```
#include "driver/gpio.h"
```

- This header file is a part of the API provided by the `esp_driver_gpio` component. To declare that your component depends on `esp_driver_gpio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gpio
```

or

```
PRIV_REQUIRES esp_driver_gpio
```

Functions

esp_err_t **gpio_config**(const *gpio_config_t* *pGPIOConfig)

GPIO common configuration.

```
Configure GPIO's Mode,pull-up,PullDown,IntrType
```

参数 **pGPIOConfig** -- Pointer to GPIO configure struct

返回

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_reset_pin**(gpio_num_t gpio_num)

Reset an gpio to default state (select gpio function, enable pullup and disable input and output).

备注： This function also configures the IOMUX for this pin to the GPIO function, and disconnects any other peripheral output configured via GPIO Matrix.

参数 **gpio_num** -- GPIO number.

返回 Always return ESP_OK.

esp_err_t **gpio_set_intr_type**(gpio_num_t gpio_num, *gpio_int_type_t* intr_type)

GPIO set interrupt trigger type.

参数

- **gpio_num** -- GPIO number. If you want to set the trigger type of e.g. of GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **intr_type** -- Interrupt type, select from gpio_int_type_t

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_intr_enable**(gpio_num_t gpio_num)

Enable GPIO module interrupt signal.

备注: ESP32: Please do not use the interrupt of GPIO36 and GPIO39 when using ADC or Wi-Fi and Bluetooth with sleep mode enabled. Please refer to the comments of `adc1_get_raw`. Please refer to Section 3.11 of ESP32 ECO and Workarounds for Bugs for the description of this issue.

参数 **gpio_num** -- GPIO number. If you want to enable an interrupt on e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_intr_disable**(gpio_num_t gpio_num)

Disable GPIO module interrupt signal.

备注: This function is allowed to be executed when Cache is disabled within ISR context, by enabling `CONFIG_GPIO_CTRL_FUNC_IN_IRAM`

参数 **gpio_num** -- GPIO number. If you want to disable the interrupt of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);

返回

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_set_level**(gpio_num_t gpio_num, uint32_t level)

GPIO set output level.

备注: This function is allowed to be executed when Cache is disabled within ISR context, by enabling `CONFIG_GPIO_CTRL_FUNC_IN_IRAM`

参数

- **gpio_num** -- GPIO number. If you want to set the output level of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **level** -- Output level. 0: low ; 1: high

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO number error

int **gpio_get_level**(gpio_num_t gpio_num)

GPIO get input level.

警告: If the pad is not configured for input (or input and output) the returned value is always 0.

参数 **gpio_num** -- GPIO number. If you want to get the logic level of e.g. pin GPIO16, gpio_num should be GPIO_NUM_16 (16);

返回

- 0 the GPIO input level is 0
- 1 the GPIO input level is 1

esp_err_t **gpio_set_direction**(gpio_num_t gpio_num, *gpio_mode_t* mode)

GPIO set direction.

Configure GPIO direction,such as output_only,input_only,output_and_input

参数

- **gpio_num** -- Configure GPIO pins number, it should be GPIO number. If you want to set direction of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **mode** -- GPIO direction

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO error

esp_err_t **gpio_set_pull_mode**(gpio_num_t gpio_num, *gpio_pull_mode_t* pull)

Configure GPIO pull-up/pull-down resistors.

备注: ESP32: Only pins that support both input & output have integrated pull-up and pull-down resistors. Input-only GPIOs 34-39 do not.

参数

- **gpio_num** -- GPIO number. If you want to set pull up or down mode for e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **pull** -- GPIO pull up/down mode.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG : Parameter error

esp_err_t **gpio_wakeup_enable**(gpio_num_t gpio_num, *gpio_int_type_t* intr_type)

Enable GPIO wake-up function.

参数

- **gpio_num** -- GPIO number.
- **intr_type** -- GPIO wake-up type. Only GPIO_INTR_LOW_LEVEL or GPIO_INTR_HIGH_LEVEL can be used.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_wakeup_disable**(gpio_num_t gpio_num)

Disable GPIO wake-up function.

参数 **gpio_num** -- GPIO number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_isr_register**(void (*fn)(void*), void *arg, int intr_alloc_flags, *gpio_isr_handle_t* *handle)

Register GPIO interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

This ISR function is called whenever any GPIO interrupt occurs. See the alternative gpio_install_isr_service() and gpio_isr_handler_add() API in order to have the driver support per-GPIO ISRs.

To disable or remove the ISR, pass the returned handle to the *interrupt allocation functions*.

参数

- **fn** -- Interrupt handler function.
- **arg** -- Parameter for handler function
- **intr_alloc_flags** -- Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

- **handle** -- Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

返回

- ESP_OK Success ;
- ESP_ERR_INVALID_ARG GPIO error
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags

esp_err_t **gpio_pullup_en**(gpio_num_t gpio_num)

Enable pull-up on GPIO.

参数 **gpio_num** -- GPIO number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_pullup_dis**(gpio_num_t gpio_num)

Disable pull-up on GPIO.

参数 **gpio_num** -- GPIO number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_pulldown_en**(gpio_num_t gpio_num)

Enable pull-down on GPIO.

参数 **gpio_num** -- GPIO number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_pulldown_dis**(gpio_num_t gpio_num)

Disable pull-down on GPIO.

参数 **gpio_num** -- GPIO number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_install_isr_service**(int intr_alloc_flags)

Install the GPIO driver's ETS_GPIO_INTR_SOURCE ISR handler service, which allows per-pin GPIO interrupt handlers.

This function is incompatible with gpio_isr_register() - if that function is used, a single global ISR is registered for all GPIO interrupts. If this function is used, the ISR service provides a global GPIO ISR and individual pin handlers are registered via the gpio_isr_handler_add() function.

参数 **intr_alloc_flags** -- Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

返回

- ESP_OK Success
- ESP_ERR_NO_MEM No memory to install this service
- ESP_ERR_INVALID_STATE ISR service already installed.
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags
- ESP_ERR_INVALID_ARG GPIO error

void **gpio_uninstall_isr_service**(void)

Uninstall the driver's GPIO ISR service, freeing related resources.

esp_err_t **gpio_isr_handler_add**(gpio_num_t gpio_num, *gpio_isr_t* isr_handler, void *args)

Add ISR handler for the corresponding GPIO pin.

Call this function after using gpio_install_isr_service() to install the driver's GPIO ISR handler service.

The pin ISR handlers no longer need to be declared with IRAM_ATTR, unless you pass the ESP_INTR_FLAG_IRAM flag when allocating the ISR in gpio_install_isr_service().

This ISR handler will be called from an ISR. So there is a stack size limit (configurable as "ISR stack size" in menuconfig). This limit is smaller compared to a global GPIO interrupt handler due to the additional level of indirection.

参数

- **gpio_num** -- GPIO number
- **isr_handler** -- ISR handler function for the corresponding GPIO number.
- **args** -- parameter for ISR handler.

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE Wrong state, the ISR service has not been initialized.
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_isr_handler_remove**(gpio_num_t gpio_num)

Remove ISR handler for the corresponding GPIO pin.

参数 **gpio_num** -- GPIO number

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE Wrong state, the ISR service has not been initialized.
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_set_drive_capability**(gpio_num_t gpio_num, *gpio_drive_cap_t* strength)

Set GPIO pad drive capability.

参数

- **gpio_num** -- GPIO number, only support output GPIOs
- **strength** -- Drive capability of the pad

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_get_drive_capability**(gpio_num_t gpio_num, *gpio_drive_cap_t* *strength)

Get GPIO pad drive capability.

参数

- **gpio_num** -- GPIO number, only support output GPIOs
- **strength** -- Pointer to accept drive capability of the pad

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_hold_en**(gpio_num_t gpio_num)

Enable gpio pad hold function.

When a GPIO is set to hold, its state is latched at that moment and will not change when the internal signal or the IO MUX/GPIO configuration is modified (including input enable, output enable, output value, function, and drive strength values). This function can be used to retain the state of GPIOs when the power domain of where GPIO/IOMUX belongs to becomes off. For example, chip or system is reset (e.g. watchdog time-out, deep-sleep events are triggered), or peripheral power-down in light-sleep.

This function works in both input and output modes, and only applicable to output-capable GPIOs. If this function is enabled: in output mode: the output level of the GPIO will be locked and can not be changed. in input mode: the input read value can still reflect the changes of the input signal.

Please be aware that,

On ESP32P4, the states of IOs can not be hold after waking up from Deep-sleep.

Additionally, on ESP32/S2/C3/S3/C2, this function cannot be used to hold the state of a digital GPIO during Deep-sleep. Even if this function is enabled, the digital GPIO will be reset to its default state when the chip

wakes up from Deep-sleep. If you want to hold the state of a digital GPIO during Deep-sleep, please call `gpio_deep_sleep_hold_en`.

Power down or call `gpio_hold_dis` will disable this function.

参数 **gpio_num** -- GPIO number, only support output-capable GPIOs

返回

- ESP_OK Success
- ESP_ERR_NOT_SUPPORTED Not support pad hold function

esp_err_t **gpio_hold_dis**(gpio_num_t gpio_num)

Disable gpio pad hold function.

When the chip is woken up from peripheral power-down sleep, the gpio will be set to the default mode, so, the gpio will output the default level if this function is called. If you don't want the level changes, the gpio should be configured to a known state before this function is called. e.g. If you hold gpio18 high during Deep-sleep, after the chip is woken up and `gpio_hold_dis` is called, gpio18 will output low level(because gpio18 is input mode by default). If you don't want this behavior, you should configure gpio18 as output mode and set it to high level before calling `gpio_hold_dis`.

参数 **gpio_num** -- GPIO number, only support output-capable GPIOs

返回

- ESP_OK Success
- ESP_ERR_NOT_SUPPORTED Not support pad hold function

void **gpio_iomux_in**(uint32_t gpio_num, uint32_t signal_idx)

Set pad input to a peripheral signal through the IOMUX.

参数

- **gpio_num** -- GPIO number of the pad.
- **signal_idx** -- Peripheral signal id to input. One of the `*_IN_IDX` signals in `soc/gpio_sig_map.h`.

void **gpio_iomux_out**(uint8_t gpio_num, int func, bool out_en_inv)

Set peripheral output to an GPIO pad through the IOMUX.

参数

- **gpio_num** -- gpio_num GPIO number of the pad.
- **func** -- The function number of the peripheral pin to output pin. One of the `FUNC_X_*` of specified pin (X) in `soc/io_mux_reg.h`.
- **out_en_inv** -- True if the output enable needs to be inverted, otherwise False.

esp_err_t **gpio_force_hold_all**(void)

Force hold all digital and rtc gpio pads.

GPIO force hold, no matter the chip in active mode or sleep modes.

This function will immediately cause all pads to latch the current values of input enable, output enable, output value, function, and drive strength values.

警告:

a. This function will hold flash and UART pins as well. Therefore, this function, and all code run afterwards (till calling `gpio_force_unhold_all` to disable this feature), MUST be placed in internal RAM as holding the flash pins will halt SPI flash operation, and holding the UART pins will halt any UART logging.
b. The hold state of all pads will be cancelled during ROM boot, so it is not recommended to use this API to hold the pads state during deepsleep and reset.

esp_err_t **gpio_force_unhold_all**(void)

Unhold all digital and rtc gpio pads.

备注： The global hold signal and the hold signal of each IO act on the PAD through 'or' logic, so if a pad has already been configured to hold by `gpio_hold_en`, this API can't release its hold state.

esp_err_t **gpio_sleep_sel_en**(gpio_num_t gpio_num)

Enable SLP_SEL to change GPIO status automantically in lightsleep.

参数 **gpio_num** -- GPIO number of the pad.

返回

- ESP_OK Success

esp_err_t **gpio_sleep_sel_dis**(gpio_num_t gpio_num)

Disable SLP_SEL to change GPIO status automantically in lightsleep.

参数 **gpio_num** -- GPIO number of the pad.

返回

- ESP_OK Success

esp_err_t **gpio_sleep_set_direction**(gpio_num_t gpio_num, *gpio_mode_t* mode)

GPIO set direction at sleep.

Configure GPIO direction,such as output_only,input_only,output_and_input

参数

- **gpio_num** -- Configure GPIO pins number, it should be GPIO number. If you want to set direction of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **mode** -- GPIO direction

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO error

esp_err_t **gpio_sleep_set_pull_mode**(gpio_num_t gpio_num, *gpio_pull_mode_t* pull)

Configure GPIO pull-up/pull-down resistors at sleep.

备注： ESP32: Only pins that support both input & output have integrated pull-up and pull-down resistors. Input-only GPIOs 34-39 do not.

参数

- **gpio_num** -- GPIO number. If you want to set pull up or down mode for e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **pull** -- GPIO pull up/down mode.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG : Parameter error

esp_err_t **gpio_deep_sleep_wakeup_enable**(gpio_num_t gpio_num, *gpio_int_type_t* intr_type)

Enable GPIO deep-sleep wake-up function.

备注： Called by the SDK. User shouldn't call this directly in the APP.

参数

- **gpio_num** -- GPIO number.
- **intr_type** -- GPIO wake-up type. Only GPIO_INTR_LOW_LEVEL or GPIO_INTR_HIGH_LEVEL can be used.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_deep_sleep_wakeup_disable**(gpio_num_t gpio_num)

Disable GPIO deep-sleep wake-up function.

参数 **gpio_num** -- GPIO number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **gpio_dump_io_configuration**(FILE *out_stream, uint64_t io_bit_mask)

Dump IO configuration information to console.

参数

- **out_stream** -- IO stream (e.g. stdout)
- **io_bit_mask** -- IO pin bit mask, each bit maps to an IO

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Structures

struct **gpio_config_t**

Configuration parameters of GPIO pad for gpio_config function.

Public Members

uint64_t **pin_bit_mask**

GPIO pin: set with bit mask, each bit maps to a GPIO

gpio_mode_t **mode**

GPIO mode: set input/output mode

gpio_pullup_t **pull_up_en**

GPIO pull-up

gpio_pulldown_t **pull_down_en**

GPIO pull-down

gpio_int_type_t **intr_type**

GPIO interrupt type

gpio_hys_ctrl_mode_t **hys_ctrl_mode**

GPIO hysteresis: hysteresis filter on slope input

Macros

GPIO_PIN_COUNT

GPIO_IS_VALID_GPIO(gpio_num)

Check whether it is a valid GPIO number.

GPIO_IS_VALID_OUTPUT_GPIO(gpio_num)

Check whether it can be a valid GPIO number of output mode.

GPIO_IS_VALID_DIGITAL_IO_PAD(gpio_num)

Check whether it can be a valid digital I/O pad.

GPIO_IS_DEEP_SLEEP_WAKEUP_VALID_GPIO(gpio_num)

Type Definitions

typedef *intr_handle_t* **gpio_isr_handle_t**

typedef void (***gpio_isr_t**)(void *arg)

GPIO interrupt handler.

Param arg User registered data

Header File

- components/hal/include/hal/gpio_types.h
- This header file can be included with:

```
#include "hal/gpio_types.h"
```

Macros

GPIO_PIN_REG_0

GPIO_PIN_REG_1

GPIO_PIN_REG_2

GPIO_PIN_REG_3

GPIO_PIN_REG_4

GPIO_PIN_REG_5

GPIO_PIN_REG_6

GPIO_PIN_REG_7

GPIO_PIN_REG_8

GPIO_PIN_REG_9

GPIO_PIN_REG_10

GPIO_PIN_REG_11

GPIO_PIN_REG_12

GPIO_PIN_REG_13

GPIO_PIN_REG_14

GPIO_PIN_REG_15

GPIO_PIN_REG_16

GPIO_PIN_REG_17

GPIO_PIN_REG_18

GPIO_PIN_REG_19

GPIO_PIN_REG_20

GPIO_PIN_REG_21

GPIO_PIN_REG_22

GPIO_PIN_REG_23

GPIO_PIN_REG_24

GPIO_PIN_REG_25

GPIO_PIN_REG_26

GPIO_PIN_REG_27

GPIO_PIN_REG_28

GPIO_PIN_REG_29

GPIO_PIN_REG_30

GPIO_PIN_REG_31

GPIO_PIN_REG_32

GPIO_PIN_REG_33

GPIO_PIN_REG_34

GPIO_PIN_REG_35

GPIO_PIN_REG_36

GPIO_PIN_REG_37

GPIO_PIN_REG_38

GPIO_PIN_REG_39

GPIO_PIN_REG_40

GPIO_PIN_REG_41

GPIO_PIN_REG_42

GPIO_PIN_REG_43

GPIO_PIN_REG_44

GPIO_PIN_REG_45

GPIO_PIN_REG_46

GPIO_PIN_REG_47

GPIO_PIN_REG_48

GPIO_PIN_REG_49

GPIO_PIN_REG_50

GPIO_PIN_REG_51

GPIO_PIN_REG_52

GPIO_PIN_REG_53

GPIO_PIN_REG_54

Enumerations

enum **gpio_port_t**

Values:

enumerator **GPIO_PORT_0**

enumerator **GPIO_PORT_MAX**

enum **gpio_int_type_t**

Values:

enumerator **GPIO_INTR_DISABLE**

Disable GPIO interrupt

enumerator **GPIO_INTR_POSEDGE**

GPIO interrupt type : rising edge

enumerator **GPIO_INTR_NEGEDGE**

GPIO interrupt type : falling edge

enumerator **GPIO_INTR_ANYEDGE**

GPIO interrupt type : both rising and falling edge

enumerator **GPIO_INTR_LOW_LEVEL**

GPIO interrupt type : input low level trigger

enumerator **GPIO_INTR_HIGH_LEVEL**

GPIO interrupt type : input high level trigger

enumerator **GPIO_INTR_MAX**

enum **gpio_mode_t**

Values:

enumerator **GPIO_MODE_DISABLE**

GPIO mode : disable input and output

enumerator **GPIO_MODE_INPUT**

GPIO mode : input only

enumerator **GPIO_MODE_OUTPUT**

GPIO mode : output only mode

enumerator **GPIO_MODE_OUTPUT_OD**

GPIO mode : output only with open-drain mode

enumerator **GPIO_MODE_INPUT_OUTPUT_OD**

GPIO mode : output and input with open-drain mode

enumerator **GPIO_MODE_INPUT_OUTPUT**

GPIO mode : output and input mode

enum **gpio_pullup_t**

Values:

enumerator **GPIO_PULLUP_DISABLE**

Disable GPIO pull-up resistor

enumerator **GPIO_PULLUP_ENABLE**

Enable GPIO pull-up resistor

enum **gpio_pulldown_t**

Values:

enumerator **GPIO_PULLDOWN_DISABLE**

Disable GPIO pull-down resistor

enumerator **GPIO_PULLDOWN_ENABLE**

Enable GPIO pull-down resistor

enum **gpio_pull_mode_t**

Values:

enumerator **GPIO_PULLUP_ONLY**

Pad pull up

enumerator **GPIO_PULLDOWN_ONLY**

Pad pull down

enumerator **GPIO_PULLUP_PULLDOWN**

Pad pull up + pull down

enumerator **GPIO_FLOATING**

Pad floating

enum **gpio_drive_cap_t**

Values:

enumerator **GPIO_DRIVE_CAP_0**

Pad drive capability: weak

enumerator **GPIO_DRIVE_CAP_1**

Pad drive capability: stronger

enumerator **GPIO_DRIVE_CAP_2**

Pad drive capability: medium

enumerator **GPIO_DRIVE_CAP_DEFAULT**

Pad drive capability: medium

enumerator **GPIO_DRIVE_CAP_3**

Pad drive capability: strongest

enumerator **GPIO_DRIVE_CAP_MAX**

enum **gpio_hys_ctrl_mode_t**

Available option for configuring hysteresis feature of GPIOs.

Values:

enumerator **GPIO_HYS_SOFT_DISABLE**

Pad input hysteresis disable by software

enumerator **GPIO_HYS_SOFT_ENABLE**

Pad input hysteresis enable by software

API 参考 - RTC GPIO

Header File

- components/esp_driver_gpio/include/driver/rtc_io.h
- This header file can be included with:

```
#include "driver/rtc_io.h"
```

- This header file is a part of the API provided by the `esp_driver_gpio` component. To declare that your component depends on `esp_driver_gpio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gpio
```

or

```
PRIV_REQUIRES esp_driver_gpio
```

Functions

bool **rtc_gpio_is_valid_gpio**(gpio_num_t gpio_num)

Determine if the specified GPIO is a valid RTC GPIO.

参数 **gpio_num** -- GPIO number
返回 true if GPIO is valid for RTC GPIO use. false otherwise.

int **rtc_io_number_get**(gpio_num_t gpio_num)

Get RTC IO index number by gpio number.

参数 **gpio_num** -- GPIO number
返回 >=0: Index of rtcio. -1 : The gpio is not rtcio.

esp_err_t **rtc_gpio_init**(gpio_num_t gpio_num)

Init a GPIO as RTC GPIO.

This function must be called when initializing a pad for an analog function.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)
返回

- ESP_OK success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_deinit**(gpio_num_t gpio_num)

Init a GPIO as digital GPIO.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)
返回

- ESP_OK success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

uint32_t **rtc_gpio_get_level**(gpio_num_t gpio_num)

Get the RTC IO input level.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)

返回

- 1 High level
- 0 Low level
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_set_level**(gpio_num_t gpio_num, uint32_t level)

Set the RTC IO output level.

参数

- **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)
- **level** -- output level

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_set_direction**(gpio_num_t gpio_num, *rtc_gpio_mode_t* mode)

RTC GPIO set direction.

Configure RTC GPIO direction, such as output only, input only, output and input.

参数

- **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)
- **mode** -- GPIO direction

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_set_direction_in_sleep**(gpio_num_t gpio_num, *rtc_gpio_mode_t* mode)

RTC GPIO set direction in deep sleep mode or disable sleep status (default). In some application scenarios, IO needs to have another states during deep sleep.

NOTE: ESP32 supports INPUT_ONLY mode. The rest targets support INPUT_ONLY, OUTPUT_ONLY, INPUT_OUTPUT mode.

参数

- **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)
- **mode** -- GPIO direction

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_pullup_en**(gpio_num_t gpio_num)

RTC GPIO pullup enable.

This function only works for RTC IOs. In general, call gpio_pullup_en, which will work both for normal GPIOs and RTC IOs.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_pulldown_en**(gpio_num_t gpio_num)

RTC GPIO pulldown enable.

This function only works for RTC IOs. In general, call gpio_pulldown_en, which will work both for normal GPIOs and RTC IOs.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_pullup_dis**(gpio_num_t gpio_num)

RTC GPIO pullup disable.

This function only works for RTC IOs. In general, call gpio_pullup_dis, which will work both for normal GPIOs and RTC IOs.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_pulldown_dis**(gpio_num_t gpio_num)

RTC GPIO pulldown disable.

This function only works for RTC IOs. In general, call gpio_pulldown_dis, which will work both for normal GPIOs and RTC IOs.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_set_drive_capability**(gpio_num_t gpio_num, *gpio_drive_cap_t* strength)

Set RTC GPIO pad drive capability.

参数

- **gpio_num** -- GPIO number, only support output GPIOs
- **strength** -- Drive capability of the pad

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **rtc_gpio_get_drive_capability**(gpio_num_t gpio_num, *gpio_drive_cap_t* *strength)

Get RTC GPIO pad drive capability.

参数

- **gpio_num** -- GPIO number, only support output GPIOs
- **strength** -- Pointer to accept drive capability of the pad

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **rtc_gpio_iomux_func_sel**(gpio_num_t gpio_num, int func)

Select a RTC IOMUX function for the RTC IO.

参数

- **gpio_num** -- GPIO number
- **func** -- Function to assign to the pin

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **rtc_gpio_hold_en**(gpio_num_t gpio_num)

Enable hold function on an RTC IO pad.

Enabling HOLD function will cause the pad to latch current values of input enable, output enable, output value, function, drive strength values. This function is useful when going into light or deep sleep mode to prevent the pin configuration from changing.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_hold_dis**(gpio_num_t gpio_num)

Disable hold function on an RTC IO pad.

Disabling hold function will allow the pad receive the values of input enable, output enable, output value, function, drive strength from RTC_IO peripheral.

参数 **gpio_num** -- GPIO number (e.g. GPIO_NUM_12)

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t **rtc_gpio_force_hold_en_all**(void)

Enable force hold signal for all RTC IOs.

Each RTC pad has a "force hold" input signal from the RTC controller. If this signal is set, pad latches current values of input enable, function, output enable, and other signals which come from the RTC mux. Force hold signal is enabled before going into deep sleep for pins which are used for EXT1 wakeup.

esp_err_t **rtc_gpio_force_hold_dis_all**(void)

Disable force hold signal for all RTC IOs.

esp_err_t **rtc_gpio_wakeup_enable**(gpio_num_t gpio_num, *gpio_int_type_t* intr_type)

Enable wakeup from sleep mode using specific GPIO.

参数

- **gpio_num** -- GPIO number
- **intr_type** -- Wakeup on high level (GPIO_INTR_HIGH_LEVEL) or low level (GPIO_INTR_LOW_LEVEL)

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if gpio_num is not an RTC IO, or intr_type is not one of GPIO_INTR_HIGH_LEVEL, GPIO_INTR_LOW_LEVEL.

esp_err_t **rtc_gpio_wakeup_disable**(gpio_num_t gpio_num)

Disable wakeup from sleep mode using specific GPIO.

参数 **gpio_num** -- GPIO number

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if gpio_num is not an RTC IO

Macros

RTC_GPIO_IS_VALID_GPIO(gpio_num)

Header File

- components/esp_driver_gpio/include/driver/lp_io.h
- This header file can be included with:

```
#include "driver/lp_io.h"
```

- This header file is a part of the API provided by the `esp_driver_gpio` component. To declare that your component depends on `esp_driver_gpio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gpio
```

or

```
PRIV_REQUIRES esp_driver_gpio
```

Functions

esp_err_t **lp_gpio_connect_in_signal**(gpio_num_t gpio_num, uint32_t signal_idx, bool inv)

Connect a RTC(LP) GPIO input with a peripheral signal, which tagged as input attribute.

备注: There's no limitation on the number of signals that a RTC(LP) GPIO can connect with

参数

- **gpio_num** -- GPIO number, especially, LP_GPIO_MATRIX_CONST_ZERO_INPUT means connect logic 0 to signal LP_GPIO_MATRIX_CONST_ONE_INPUT means connect logic 1 to signal
- **signal_idx** -- LP peripheral signal index (tagged as input attribute)
- **inv** -- Whether the RTC(LP) GPIO input to be inverted or not

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **lp_gpio_connect_out_signal**(gpio_num_t gpio_num, uint32_t signal_idx, bool out_inv, bool out_en_inv)

Connect a peripheral signal which tagged as output attribute with a RTC(LP) GPIO.

备注: There's no limitation on the number of RTC(LP) GPIOs that a signal can connect with

参数

- **gpio_num** -- GPIO number
- **signal_idx** -- LP peripheral signal index (tagged as input attribute), especially, SIG_LP_GPIO_OUT_IDX means disconnect RTC(LP) GPIO and other peripherals. Only the RTC GPIO driver can control the output level
- **out_inv** -- Whether to signal to be inverted or not
- **out_en_inv** -- Whether the output enable control is inverted or not

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Header File

- components/hal/include/hal/rtc_io_types.h
- This header file can be included with:

```
#include "hal/rtc_io_types.h"
```

Enumerations

enum **rtc_gpio_mode_t**

RTCIO output/input mode type.

Values:

enumerator **RTC_GPIO_MODE_INPUT_ONLY**

Pad input

enumerator **RTC_GPIO_MODE_OUTPUT_ONLY**

Pad output

enumerator **RTC_GPIO_MODE_INPUT_OUTPUT**

Pad input + output

enumerator **RTC_GPIO_MODE_DISABLED**

Pad (output + input) disable

enumerator **RTC_GPIO_MODE_OUTPUT_OD**

Pad open-drain output

enumerator **RTC_GPIO_MODE_INPUT_OUTPUT_OD**

Pad input + open-drain output

API 参考 - GPIO 毛刺过滤器

Header File

- components/esp_driver_gpio/include/driver/gpio_filter.h
- This header file can be included with:

```
#include "driver/gpio_filter.h"
```

- This header file is a part of the API provided by the `esp_driver_gpio` component. To declare that your component depends on `esp_driver_gpio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gpio
```

or

```
PRIV_REQUIRES esp_driver_gpio
```

Functions

esp_err_t **gpio_new_pin_glitch_filter**(const *gpio_pin_glitch_filter_config_t* *config, *gpio_glitch_filter_handle_t* *ret_filter)

Create a pin glitch filter.

备注： Pin glitch filter parameters are fixed, pulses shorter than two sample clocks (IO-MUX's source clock) will be filtered out. It's independent with "flex" glitch filter. See also `gpio_new_flex_glitch_filter`.

备注： The created filter handle can later be deleted by `gpio_del_glitch_filter`.

参数

- **config** -- **[in]** Glitch filter configuration
- **ret_filter** -- **[out]** Returned glitch filter handle

返回

- ESP_OK: Create a pin glitch filter successfully
- ESP_ERR_INVALID_ARG: Create a pin glitch filter failed because of invalid arguments (e.g. wrong GPIO number)
- ESP_ERR_NO_MEM: Create a pin glitch filter failed because of out of memory
- ESP_FAIL: Create a pin glitch filter failed because of other error

esp_err_t **gpio_new_flex_glitch_filter**(const *gpio_flex_glitch_filter_config_t* *config, *gpio_glitch_filter_handle_t* *ret_filter)

Allocate a flex glitch filter.

备注： "flex" means the filter parameters (window, threshold) are adjustable. It's independent with pin glitch filter. See also `gpio_new_pin_glitch_filter`.

备注： The created filter handle can later be deleted by `gpio_del_glitch_filter`.

参数

- **config** -- **[in]** Glitch filter configuration
- **ret_filter** -- **[out]** Returned glitch filter handle

返回

- ESP_OK: Allocate a flex glitch filter successfully
- ESP_ERR_INVALID_ARG: Allocate a flex glitch filter failed because of invalid arguments (e.g. wrong GPIO number, filter parameters out of range)
- ESP_ERR_NO_MEM: Allocate a flex glitch filter failed because of out of memory
- ESP_ERR_NOT_FOUND: Allocate a flex glitch filter failed because the underlying hardware resources are used up
- ESP_FAIL: Allocate a flex glitch filter failed because of other error

esp_err_t **gpio_del_glitch_filter**(*gpio_glitch_filter_handle_t* filter)

Delete a glitch filter.

参数 **filter** -- **[in]** Glitch filter handle returned from `gpio_new_flex_glitch_filter` or `gpio_new_pin_glitch_filter`

返回

- ESP_OK: Delete glitch filter successfully
- ESP_ERR_INVALID_ARG: Delete glitch filter failed because of invalid arguments
- ESP_ERR_INVALID_STATE: Delete glitch filter failed because the glitch filter is still in working
- ESP_FAIL: Delete glitch filter failed because of other error

esp_err_t **gpio_glitch_filter_enable**(*gpio_glitch_filter_handle_t* filter)

Enable a glitch filter.

参数 **filter** -- **[in]** Glitch filter handle returned from `gpio_new_flex_glitch_filter` or `gpio_new_pin_glitch_filter`

返回

- ESP_OK: Enable glitch filter successfully
- ESP_ERR_INVALID_ARG: Enable glitch filter failed because of invalid arguments
- ESP_ERR_INVALID_STATE: Enable glitch filter failed because the glitch filter is already enabled
- ESP_FAIL: Enable glitch filter failed because of other error

esp_err_t **gpio_glitch_filter_disable**(*gpio_glitch_filter_handle_t* filter)

Disable a glitch filter.

参数 **filter** -- **[in]** Glitch filter handle returned from `gpio_new_flex_glitch_filter` or `gpio_new_pin_glitch_filter`

返回

- ESP_OK: Disable glitch filter successfully
- ESP_ERR_INVALID_ARG: Disable glitch filter failed because of invalid arguments
- ESP_ERR_INVALID_STATE: Disable glitch filter failed because the glitch filter is not enabled yet
- ESP_FAIL: Disable glitch filter failed because of other error

Structures

struct **gpio_pin_glitch_filter_config_t**

Configuration of GPIO pin glitch filter.

Public Members

glitch_filter_clock_source_t **clk_src**

Clock source for the glitch filter

gpio_num_t **gpio_num**

GPIO number

struct **gpio_flex_glitch_filter_config_t**

Configuration of GPIO flex glitch filter.

Public Members

glitch_filter_clock_source_t **clk_src**

Clock source for the glitch filter

gpio_num_t **gpio_num**

GPIO number

uint32_t **window_width_ns**

Sample window width (in ns)

uint32_t **window_thres_ns**

Sample window threshold (in ns), during the `window_width_ns` sample window, any pulse whose width < window_thres_ns will be discarded.

Type Definitions

typedef struct gpio_glitch_filter_t ***gpio_glitch_filter_handle_t**

Typedef of GPIO glitch filter handle.

2.5.8 通用定时器

简介

通用定时器是 ESP32-P4 定时器组外设的驱动程序。ESP32-P4 硬件定时器分辨率高，具有灵活的报警功能。定时器内部计数器达到特定目标数值的行为被称为定时器报警。定时器报警时将调用用户注册的不同定时器回调函数。

通用定时器通常在以下场景中使用：

- 如同挂钟一般自由运行，随时随地获取高分辨率时间戳；
- 生成周期性警报，定期触发事件；
- 生成一次性警报，在目标时间内响应。

功能概述

下文介绍了配置和操作定时器的常规步骤：

- 资源分配 - 获取定时器句柄应设置的参数，以及如何在通用定时器完成工作时回收资源。
- 设置和获取计数值 - 如何强制定时器从起点开始计数，以及如何随时获取计数值。
- 设置警报动作 - 启动警报事件应设置的参数。
- 注册事件回调函数 - 如何将用户的特定代码挂载到警报事件回调函数。
- 使能和禁用定时器 - 如何使能和禁用定时器。
- 启动和停止定时器 - 通过不同报警行为启动定时器的典型使用场景。
- *ETM* 事件与任务 - 定时器提供了哪些事件和任务可以连接到 ETM 通道上。
- 电源管理 - 选择不同的时钟源将会如何影响功耗。
- *IRAM* 安全 - 在 cache 禁用的情况下，如何更好地让定时器处理中断事务以及实现 IO 控制功能。
- 线程安全 - 驱动程序保证哪些 API 线程安全。
- *Kconfig* 选项 - 支持的 Kconfig 选项，这些选项会对驱动程序行为产生不同影响。

资源分配 不同的 ESP 芯片可能有不同数量的独立定时器组，每组内也可能有若干个独立定时器。[1]

通用定时器实例由 *gptimer_handle_t* 表示。可用硬件资源汇集在资源池内，由后台驱动程序管理，无需考虑硬件所属的定时器以及定时器组。

要安装一个定时器实例，需要提前提供配置结构体 *gptimer_config_t*：

- *gptimer_config_t::clk_src* 选择定时器的时钟源。*gptimer_clock_source_t* 中列出多个可用时钟，仅可选择其中一个时钟。了解不同时钟源对功耗的影响，请查看章节电源管理。
- *gptimer_config_t::direction* 设置定时器的计数方向，*gptimer_count_direction_t* 中列出多个支持的方向，仅可选择其中一个方向。
- *gptimer_config_t::resolution_hz* 设置内部计数器的分辨率。计数器每滴答一次相当于 **1 / resolution_hz** 秒。
- `gptimer_config::intr_priority` 设置中断的优先级。如果设置为 `0`，则会分配一个默认优先级的中断，否则会使用指定的优先级。
- `gptimer_config::backup_before_sleep` 用于使能在进入睡眠模式前备份 GPTimer 寄存器。这个选项需要用户在功耗和内存使用之间取得平衡。如果功耗不是一个问题，可以禁用这个选项来节省内存。但如果想要节省功耗，应该使能这个选项，在进入睡眠模式前备份 GPTimer 寄存器，并在唤醒后恢复它们。这个功能依赖于特定的硬件模块，如果你在不支持的芯片上启用它，你会得到一个错误信息，如 `register back up is not supported`。
- 可选地，*gptimer_config_t::intr_shared* 设置是否将定时器中断源标记为共享源。了解共享中断的优缺点，请参考 *Interrupt Handling*。

完成上述结构配置之后，可以将结构传递给 *gptimer_new_timer()*，用以实例化定时器实例并返回定时器句柄。

该函数可能由于内存不足、参数无效等错误而失败。具体来说，当没有更多的空闲定时器（即所有硬件资源已用完）时，将返回 *ESP_ERR_NOT_FOUND*。可用定时器总数由 *SOC_TIMER_GROUP_TOTAL_TIMERS* 表示，不同的 ESP 芯片该数值不同。

如已不再需要之前创建的通用定时器实例，应通过调用 *gptimer_del_timer()* 回收定时器，以便底层硬件定时器用于其他目的。在删除通用定时器句柄之前，请通过 *gptimer_disable()* 禁用定时器，或者通过 *gptimer_enable()* 确认定时器尚未使能。

创建分辨率为 1 MHz 的通用定时器句柄

```
gptimer_handle_t gptimer = NULL;
gptimer_config_t timer_config = {
    .clk_src = GPTIMER_CLK_SRC_DEFAULT,
    .direction = GPTIMER_COUNT_UP,
```

(下页继续)

[1] 不同 ESP 芯片系列的通用定时器实例数量可能不同。了解详细信息，请参考《ESP32-P4 技术参考手册》> 章节定时器组 (TIMG) [PDF]。驱动程序对通道申请数量不做限制，但当硬件资源用尽时，驱动程序将返回错误。在分配资源时，请务必检查返回值（例如 *gptimer_new_timer()*）。

(续上页)

```
    .resolution_hz = 1 * 1000 * 1000, // 1MHz, 1 tick = 1us
};
ESP_ERROR_CHECK(gptimer_new_timer(&timer_config, &gptimer));
```

设置和获取计数值 创建通用定时器时，内部计数器将默认重置为零。计数值可以通过*gptimer_set_raw_count()* 异步更新。最大计数值取决于硬件定时器的位宽，这也会在 SOC 宏*SOC_TIMER_GROUP_COUNTER_BIT_WIDTH* 中有所反映。当更新活动定时器的原始计数值时，定时器将立即从新值开始计数。

计数值可以随时通过*gptimer_get_raw_count()* 获取。

设置警报动作 对于大多数通用定时器使用场景而言，应在启动定时器之前设置警报动作，但不包括简单的挂钟场景，该场景仅需自由运行的定时器。设置警报动作，需要根据如何使用警报事件来配置*gptimer_alarm_config_t* 的不同参数：

- *gptimer_alarm_config_t::alarm_count* 设置触发警报事件的目标计数值。设置警报值时还需考虑计数方向。尤其是当*gptimer_alarm_config_t::auto_reload_on_alarm* 为 true 时，*gptimer_alarm_config_t::alarm_count* 和*gptimer_alarm_config_t::reload_count* 不能设置为相同的值，因为警报值和重载值相同时没有意义。
- *gptimer_alarm_config_t::reload_count* 代表警报事件发生时要重载的计数值。此配置仅在*gptimer_alarm_config_t::auto_reload_on_alarm* 设置为 true 时生效。
- *gptimer_alarm_config_t::auto_reload_on_alarm* 标志设置是否使能自动重载功能。如果使能，硬件定时器将在警报事件发生时立即将*gptimer_alarm_config_t::reload_count* 的值重载到计数器中。

要使警报配置生效，需要调用*gptimer_set_alarm_action()*。特别是当*gptimer_alarm_config_t* 设置为 NULL 时，报警功能将被禁用。

备注： 如果警报值已设置且定时器超过该值，则会立即触发警报。

注册事件回调函数 定时器启动后，可动态产生特定事件（如“警报事件”）。如需在事件发生时调用某些函数，请通过*gptimer_register_event_callbacks()* 将函数挂载到中断服务例程 (ISR)。*gptimer_event_callbacks_t* 中列出了所有支持的事件回调函数：

- *gptimer_event_callbacks_t::on_alarm* 设置警报事件的回调函数。由于此函数在 ISR 上下文中调用，必须确保该函数不会试图阻塞（例如，确保仅从函数内调用具有 ISR 后缀的 FreeRTOS API）。函数原型在*gptimer_alarm_cb_t* 中有所声明。

也可以通过参数 user_data，将自己的上下文保存到*gptimer_register_event_callbacks()* 中。用户数据将直接传递给回调函数。

此功能将为定时器延迟安装中断服务，但不使能中断服务。所以，请在*gptimer_enable()* 之前调用这一函数，否则将返回*ESP_ERR_INVALID_STATE* 错误。了解详细信息，请查看章节使能和禁用定时器。

使能和禁用定时器 在对定时器进行 IO 控制之前，需要先调用*gptimer_enable()* 使能定时器。此函数功能如下：

- 此函数将把定时器驱动程序的状态从 **init** 切换为 **enable**。
- 如果*gptimer_register_event_callbacks()* 已经延迟安装中断服务，此函数将使能中断服务。
- 如果选择了特定的时钟源（例如 APB 时钟），此函数将获取适当的电源管理锁。了解更多信息，请查看章节电源管理。

调用*gptimer_disable()* 会进行相反的操作，即将定时器驱动程序恢复到 **init** 状态，禁用中断服务并释放电源管理锁。

启动和停止定时器 启动和停止是定时器的基本 IO 操作。调用*gptimer_start()* 可以使内部计数器开始工作，而*gptimer_stop()* 可以使计数器停止工作。下文说明了如何在存在或不存在警报事件的情况下启动定时器。

调用*gptimer_start()* 将使驱动程序状态从 enable 转换为 run, 反之亦然。注意确保 start 和 stop 函数成对使用，否则，函数可能返回*ESP_ERR_INVALID_STATE*。

将定时器作为挂钟启动

```
ESP_ERROR_CHECK(gptimer_enable(gptimer));
ESP_ERROR_CHECK(gptimer_start(gptimer));
// Retrieve the timestamp at anytime
uint64_t count;
ESP_ERROR_CHECK(gptimer_get_raw_count(gptimer, &count));
```

触发周期性事件

```
typedef struct {
    uint64_t event_count;
} example_queue_element_t;

static bool example_timer_on_alarm_cb(gptimer_handle_t timer, const gptimer_alarm_
↪event_data_t *edata, void *user_ctx)
{
    BaseType_t high_task_awoken = pdFALSE;
    QueueHandle_t queue = (QueueHandle_t)user_ctx;
    // Retrieve the count value from event data
    example_queue_element_t ele = {
        .event_count = edata->count_value
    };
    // Optional: send the event data to other task by OS queue
    // Don't introduce complex logics in callbacks
    // Suggest dealing with event data in the main loop, instead of in this␣
↪callback
    xQueueSendFromISR(queue, &ele, &high_task_awoken);
    // return whether we need to yield at the end of ISR
    return high_task_awoken == pdTRUE;
}

gptimer_alarm_config_t alarm_config = {
    .reload_count = 0, // counter will reload with 0 on alarm event
    .alarm_count = 1000000, // period = 1s @resolution 1MHz
    .flags.auto_reload_on_alarm = true, // enable auto-reload
};
ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));

gptimer_event_callbacks_t cbs = {
    .on_alarm = example_timer_on_alarm_cb, // register user callback
};
ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &cbs, queue));
ESP_ERROR_CHECK(gptimer_enable(gptimer));
ESP_ERROR_CHECK(gptimer_start(gptimer));
```

触发一次性事件

```
typedef struct {
    uint64_t event_count;
} example_queue_element_t;

static bool example_timer_on_alarm_cb(gptimer_handle_t timer, const gptimer_alarm_
↪event_data_t *edata, void *user_ctx)
```

(下页继续)

(续上页)

```
{
    BaseType_t high_task_awoken = pdFALSE;
    QueueHandle_t queue = (QueueHandle_t)user_ctx;
    // Stop timer the sooner the better
    gptimer_stop(timer);
    // Retrieve the count value from event data
    example_queue_element_t ele = {
        .event_count = edata->count_value
    };
    // Optional: send the event data to other task by OS queue
    xQueueSendFromISR(queue, &ele, &high_task_awoken);
    // return whether we need to yield at the end of ISR
    return high_task_awoken == pdTRUE;
}

gptimer_alarm_config_t alarm_config = {
    .alarm_count = 1 * 1000 * 1000, // alarm target = 1s @resolution 1MHz
};
ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));

gptimer_event_callbacks_t cbs = {
    .on_alarm = example_timer_on_alarm_cb, // register user callback
};
ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &cbs, queue));
ESP_ERROR_CHECK(gptimer_enable(gptimer));
ESP_ERROR_CHECK(gptimer_start(gptimer));
```

警报值动态更新 通过更改*gptimer_alarm_event_data_t::alarm_value*，可以在 ISR 程序回调中动态更新警报值。警报值将在回调函数返回后更新。

```
typedef struct {
    uint64_t event_count;
} example_queue_element_t;

static bool example_timer_on_alarm_cb(gptimer_handle_t timer, const gptimer_alarm_
↪event_data_t *edata, void *user_ctx)
{
    BaseType_t high_task_awoken = pdFALSE;
    QueueHandle_t queue = (QueueHandle_t)user_data;
    // Retrieve the count value from event data
    example_queue_element_t ele = {
        .event_count = edata->count_value
    };
    // Optional: send the event data to other task by OS queue
    xQueueSendFromISR(queue, &ele, &high_task_awoken);
    // reconfigure alarm value
    gptimer_alarm_config_t alarm_config = {
        .alarm_count = edata->alarm_value + 1000000, // alarm in next 1s
    };
    gptimer_set_alarm_action(timer, &alarm_config);
    // return whether we need to yield at the end of ISR
    return high_task_awoken == pdTRUE;
}

gptimer_alarm_config_t alarm_config = {
    .alarm_count = 1000000, // initial alarm target = 1s @resolution 1MHz
};
ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));

gptimer_event_callbacks_t cbs = {
```

(下页继续)

(续上页)

```
    .on_alarm = example_timer_on_alarm_cb, // register user callback
};
ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &cbs, queue));
ESP_ERROR_CHECK(gptimer_enable(gptimer));
ESP_ERROR_CHECK(gptimer_start(gptimer));
```

ETM 事件与任务 定时器可以产生多种事件，这些事件可以连接到*ETM* 模块。*gptimer_etm_event_type_t* 中列出了定时器能够产生的事件类型。用户可以通过调用*gptimer_new_etm_event()* 来获得相应事件的 ETM event 句柄。同样地，定时器还公开了一些可被其他事件触发然后自动执行的任务。*gptimer_etm_task_type_t* 中列出了定时器能够支持的任务类型。用户可以通过调用*gptimer_new_etm_task()* 来获得相应任务的 ETM task 句柄。

关于如何将定时器事件和任务连接到 ETM 通道中，请参阅*ETM* 文档。

电源管理 当电源管理*CONFIG_PM_ENABLE* 被启用的时候，系统在进入睡眠前可能会调整或禁用时钟源。结果导致 GPTimer 的计时不准确。

驱动程序可以通过创建一个电源管理锁来防止上述问题。锁的类型会根据不同的时钟源来设置。驱动程序将在*gptimer_enable()* 中拿锁，并在*gptimer_disable()* 中释放锁。这意味着，在这两个函数之间，定时器可以正确工作，因为此时时钟源不会被禁用或改变频率。

除了时钟源的潜在变化外，当启用电源管理时，系统还可以关闭 GPTimer 寄存器所在的电源域。为确保 GPTimer 驱动程序在睡眠后继续工作，用户要么选择将 GPTimer 相关的寄存器备份到 RAM 中，要么拒绝关闭电源域。你可以根据应用需求在*gptimer_config_t::backup_before_sleep* 中设置是否需要启用寄存器备份，在功耗和内存使用之间做权衡。

IRAM 安全 默认情况下，当 cache 因写入或擦除 flash 等原因而被禁用时，通用定时器的中断服务将会延迟，造成警报中断无法及时执行。在实时应用程序中通常需要避免这一情况发生。

调用 Kconfig 选项*CONFIG_GPTIMER_ISR_IRAM_SAFE* 可实现如下功能：

- 即使禁用 cache 也可使能正在运行的中断
- 将 ISR 使用的所有函数放入 IRAM[2]
- 将驱动程序对象放入 DRAM（以防意外映射到 PSRAM）

这将允许中断在 cache 禁用时运行，但会增加 IRAM 使用量。

调用另一 Kconfig 选项*CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM* 也可将常用的 IO 控制功能放入 IRAM，以便这些函数在 cache 禁用时也能执行。常用的 IO 控制功能如下：

- *gptimer_start()*
- *gptimer_stop()*
- *gptimer_get_raw_count()*
- *gptimer_set_raw_count()*
- *gptimer_set_alarm_action()*

线程安全 驱动提供的所有 API 都是线程安全的。使用时，可以直接从不同的 RTOS 任务中调用此类函数，无需额外锁保护。以下这些函数还支持在中断上下文中运行。

- *gptimer_start()*
- *gptimer_stop()*
- *gptimer_get_raw_count()*
- *gptimer_set_raw_count()*
- *gptimer_get_captured_count()*
- *gptimer_set_alarm_action()*

[2] *gptimer_event_callbacks_t::on_alarm* 回调函数和这一函数调用的函数也需放在 IRAM 中，请自行处理。

Kconfig 选项

- *CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM* 控制着定时器控制函数的存放位置（IRAM 或 flash）。
- *CONFIG_GPTIMER_ISR_HANDLER_IN_IRAM* 控制着定时器中断处理函数的存放位置（IRAM 或 flash）。
- *CONFIG_GPTIMER_ISR_IRAM_SAFE* 控制着中断处理函数是否需要在 cache 关闭的时候被屏蔽掉。更多信息，请参阅*IRAM 安全*。
- *CONFIG_GPTIMER_ENABLE_DEBUG_LOG* 用于启用调试日志输出。启用这一选项将增加固件二进制文件大小。

应用示例

- 示例 peripherals/timer_group/gptimer 中列出了通用定时器的典型用例。
- 示例 peripherals/timer_group/gptimer_capture_hc_sr04 展示了如何在 ETM 模块的帮助下，用定时器捕获外部事件的时间戳。

API 参考

Header File

- components/esp_driver_gptimer/include/driver/gptimer.h
- This header file can be included with:

```
#include "driver/gptimer.h"
```

- This header file is a part of the API provided by the `esp_driver_gptimer` component. To declare that your component depends on `esp_driver_gptimer`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gptimer
```

or

```
PRIV_REQUIRES esp_driver_gptimer
```

Functions

esp_err_t **gptimer_new_timer**(const *gptimer_config_t* *config, *gptimer_handle_t* *ret_timer)

Create a new General Purpose Timer, and return the handle.

备注： The newly created timer is put in the ”init” state.

参数

- **config** -- **[in]** GPTimer configuration
- **ret_timer** -- **[out]** Returned timer handle

返回

- ESP_OK: Create GPTimer successfully
- ESP_ERR_INVALID_ARG: Create GPTimer failed because of invalid argument
- ESP_ERR_NO_MEM: Create GPTimer failed because out of memory
- ESP_ERR_NOT_FOUND: Create GPTimer failed because all hardware timers are used up and no more free one
- ESP_FAIL: Create GPTimer failed because of other error

esp_err_t **gptimer_del_timer**(*gptimer_handle_t* timer)

Delete the GPTimer handle.

备注： A timer must be in the ”init” state before it can be deleted.

参数 **timer** -- **[in]** Timer handle created by `gptimer_new_timer`

返回

- ESP_OK: Delete GPTimer successfully
- ESP_ERR_INVALID_ARG: Delete GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete GPTimer failed because the timer is not in init state
- ESP_FAIL: Delete GPTimer failed because of other error

esp_err_t **gptimer_set_raw_count**(*gptimer_handle_t* timer, uint64_t value)

Set GPTimer raw count value.

备注： When updating the raw count of an active timer, the timer will immediately start counting from the new value.

备注： This function is allowed to run within ISR context

备注： If `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Flash Cache is disabled.

参数

- **timer** -- **[in]** Timer handle created by `gptimer_new_timer`
- **value** -- **[in]** Count value to be set

返回

- ESP_OK: Set GPTimer raw count value successfully
- ESP_ERR_INVALID_ARG: Set GPTimer raw count value failed because of invalid argument
- ESP_FAIL: Set GPTimer raw count value failed because of other error

esp_err_t **gptimer_get_raw_count**(*gptimer_handle_t* timer, uint64_t *value)

Get GPTimer raw count value.

备注： This function will trigger a software capture event and then return the captured count value.

备注： With the raw count value and the resolution returned from `gptimer_get_resolution`, you can convert the count value into seconds.

备注： This function is allowed to run within ISR context

备注： If `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Flash Cache is disabled.

参数

- **timer** -- **[in]** Timer handle created by `gptimer_new_timer`

- **value** -- **[out]** Returned GPTimer count value

返回

- ESP_OK: Get GPTimer raw count value successfully
- ESP_ERR_INVALID_ARG: Get GPTimer raw count value failed because of invalid argument
- ESP_FAIL: Get GPTimer raw count value failed because of other error

esp_err_t **gptimer_get_resolution**(*gptimer_handle_t* timer, uint32_t *out_resolution)

Return the real resolution of the timer.

备 注: usually the timer resolution is same as what you configured in the *gptimer_config_t::resolution_hz*, but some unstable clock source (e.g. RC_FAST) will do a calibration, the real resolution can be different from the configured one.

参数

- **timer** -- **[in]** Timer handle created by `gptimer_new_timer`
- **out_resolution** -- **[out]** Returned timer resolution, in Hz

返回

- ESP_OK: Get GPTimer resolution successfully
- ESP_ERR_INVALID_ARG: Get GPTimer resolution failed because of invalid argument
- ESP_FAIL: Get GPTimer resolution failed because of other error

esp_err_t **gptimer_get_captured_count**(*gptimer_handle_t* timer, uint64_t *value)

Get GPTimer captured count value.

备 注: The capture action can be issued either by ETM event or by software (see also `gptimer_get_raw_count`).

备注: This function is allowed to run within ISR context

备注: If `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Flash Cache is disabled.

参数

- **timer** -- **[in]** Timer handle created by `gptimer_new_timer`
- **value** -- **[out]** Returned captured count value

返回

- ESP_OK: Get GPTimer captured count value successfully
- ESP_ERR_INVALID_ARG: Get GPTimer captured count value failed because of invalid argument
- ESP_FAIL: Get GPTimer captured count value failed because of other error

esp_err_t **gptimer_register_event_callbacks**(*gptimer_handle_t* timer, const *gptimer_event_callbacks_t* *cbs, void *user_data)

Set callbacks for GPTimer.

备注: User registered callbacks are expected to be runnable within ISR context

备注： The first call to this function needs to be before the call to `gptimer_enable`

备注： User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

参数

- **`timer`** -- **[in]** Timer handle created by `gptimer_new_timer`
- **`cbs`** -- **[in]** Group of callback functions
- **`user_data`** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_ERR_INVALID_STATE: Set event callbacks failed because the timer is not in init state
- ESP_FAIL: Set event callbacks failed because of other error

esp_err_t **`gptimer_set_alarm_action`**(*gptimer_handle_t* timer, const *gptimer_alarm_config_t* *config)

Set alarm event actions for GPTimer.

备注： This function is allowed to run within ISR context, so you can update new alarm action immediately in any ISR callback.

备注： If `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Flash Cache is disabled. In this case, please also ensure the *`gptimer_alarm_config_t`* instance is placed in the static data section instead of in the read-only data section. e.g.: `static gptimer_alarm_config_t alarm_config = { ... };`

参数

- **`timer`** -- **[in]** Timer handle created by `gptimer_new_timer`
- **`config`** -- **[in]** Alarm configuration, especially, set config to NULL means disabling the alarm function

返回

- ESP_OK: Set alarm action for GPTimer successfully
- ESP_ERR_INVALID_ARG: Set alarm action for GPTimer failed because of invalid argument
- ESP_FAIL: Set alarm action for GPTimer failed because of other error

esp_err_t **`gptimer_enable`**(*gptimer_handle_t* timer)

Enable GPTimer.

备注： This function will transit the timer state from ”init” to ”enable”.

备注： This function will enable the interrupt service, if it's lazy installed in `gptimer_register_event_callbacks`.

备注： This function will acquire a PM lock, if a specific source clock (e.g. APB) is selected in the *`gptimer_config_t`*, while `CONFIG_PM_ENABLE` is enabled.

备注： Enable a timer doesn't mean to start it. See also `gptimer_start` for how to make the timer start counting.

参数 **`timer`** -- **[in]** Timer handle created by `gptimer_new_timer`

返回

- ESP_OK: Enable GPTimer successfully
- ESP_ERR_INVALID_ARG: Enable GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable GPTimer failed because the timer is already enabled
- ESP_FAIL: Enable GPTimer failed because of other error

esp_err_t **`gptimer_disable`** (*gptimer_handle_t* timer)

Disable GPTimer.

备注： This function will transit the timer state from "enable" to "init".

备注： This function will disable the interrupt service if it's installed.

备注： This function will release the PM lock if it's acquired in the `gptimer_enable`.

备注： Disable a timer doesn't mean to stop it. See also `gptimer_stop` for how to make the timer stop counting.

参数 **`timer`** -- **[in]** Timer handle created by `gptimer_new_timer`

返回

- ESP_OK: Disable GPTimer successfully
- ESP_ERR_INVALID_ARG: Disable GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable GPTimer failed because the timer is not enabled yet
- ESP_FAIL: Disable GPTimer failed because of other error

esp_err_t **`gptimer_start`** (*gptimer_handle_t* timer)

Start GPTimer (internal counter starts counting)

备注： This function will transit the timer state from "enable" to "run".

备注： This function is allowed to run within ISR context

备注： If `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Flash Cache is disabled.

参数 **`timer`** -- **[in]** Timer handle created by `gptimer_new_timer`

返回

- ESP_OK: Start GPTimer successfully
- ESP_ERR_INVALID_ARG: Start GPTimer failed because of invalid argument

- ESP_ERR_INVALID_STATE: Start GPTimer failed because the timer is not enabled or is already in running
- ESP_FAIL: Start GPTimer failed because of other error

esp_err_t **gptimer_stop**(*gptimer_handle_t* timer)

Stop GPTimer (internal counter stops counting)

备注: This function will transit the timer state from ”run” to ”enable”.

备注: This function is allowed to run within ISR context

备注: If `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Flash Cache is disabled.

参数 **timer** -- **[in]** Timer handle created by `gptimer_new_timer`

返回

- ESP_OK: Stop GPTimer successfully
- ESP_ERR_INVALID_ARG: Stop GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Stop GPTimer failed because the timer is not in running.
- ESP_FAIL: Stop GPTimer failed because of other error

Structures

struct **gptimer_config_t**

General Purpose Timer configuration.

Public Members

gptimer_clock_source_t **clk_src**

GPTimer clock source

gptimer_count_direction_t **direction**

Count direction

uint32_t **resolution_hz**

Counter resolution (working frequency) in Hz, hence, the step size of each count tick equals to (1 / resolution_hz) seconds

int **intr_priority**

GPTimer interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **intr_shared**

Set true, the timer interrupt number can be shared with other peripherals

uint32_t **backup_before_sleep**

If set, the driver will backup/restore the GPTimer registers before/after entering/exist sleep mode. By this approach, the system can power off GPTimer's power domain. This can save power, but at the expense of more RAM being consumed

struct *gptimer_config_t*::[anonymous] **flags**

GPTimer config flags

struct **gptimer_event_callbacks_t**

Group of supported GPTimer callbacks.

备注： The callbacks are all running under ISR environment

备注： When CONFIG_GPTIMER_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM.

Public Members

gptimer_alarm_cb_t **on_alarm**

Timer alarm callback

struct **gptimer_alarm_config_t**

General Purpose Timer alarm configuration.

Public Members

uint64_t **alarm_count**

Alarm target count value

uint64_t **reload_count**

Alarm reload count value, effect only when `auto_reload_on_alarm` is set to true

uint32_t **auto_reload_on_alarm**

Reload the count value by hardware, immediately at the alarm event

struct *gptimer_alarm_config_t*::[anonymous] **flags**

Alarm config flags

Header File

- components/esp_driver_gptimer/include/driver/gptimer_etm.h
- This header file can be included with:

```
#include "driver/gptimer_etm.h"
```

- This header file is a part of the API provided by the `esp_driver_gptimer` component. To declare that your component depends on `esp_driver_gptimer`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gptimer
```

or

```
PRIV_REQUIRES esp_driver_gptimer
```

Functions

esp_err_t **gptimer_new_etm_event**(*gptimer_handle_t* timer, const *gptimer_etm_event_config_t* *config, *esp_etm_event_handle_t* *out_event)

Get the ETM event for GPTimer.

备注: The created ETM event object can be deleted later by calling `esp_etm_del_event`

参数

- **timer** -- **[in]** Timer handle created by `gptimer_new_timer`
- **config** -- **[in]** GPTimer ETM event configuration
- **out_event** -- **[out]** Returned ETM event handle

返回

- ESP_OK: Get ETM event successfully
- ESP_ERR_INVALID_ARG: Get ETM event failed because of invalid argument
- ESP_FAIL: Get ETM event failed because of other error

esp_err_t **gptimer_new_etm_task**(*gptimer_handle_t* timer, const *gptimer_etm_task_config_t* *config, *esp_etm_task_handle_t* *out_task)

Get the ETM task for GPTimer.

备注: The created ETM task object can be deleted later by calling `esp_etm_del_task`

参数

- **timer** -- **[in]** Timer handle created by `gptimer_new_timer`
- **config** -- **[in]** GPTimer ETM task configuration
- **out_task** -- **[out]** Returned ETM task handle

返回

- ESP_OK: Get ETM task successfully
- ESP_ERR_INVALID_ARG: Get ETM task failed because of invalid argument
- ESP_FAIL: Get ETM task failed because of other error

Structures

struct **gptimer_etm_event_config_t**

GPTimer ETM event configuration.

Public Members

gptimer_etm_event_type_t **event_type**

GPTimer ETM event type

struct **gptimer_etm_task_config_t**

GPTimer ETM task configuration.

Public Members

gptimer_etm_task_type_t **task_type**

GPTimer ETM task type

Header File

- components/esp_driver_gptimer/include/driver/gptimer_types.h
- This header file can be included with:

```
#include "driver/gptimer_types.h"
```

- This header file is a part of the API provided by the `esp_driver_gptimer` component. To declare that your component depends on `esp_driver_gptimer`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gptimer
```

or

```
PRIV_REQUIRES esp_driver_gptimer
```

Structures

struct **gptimer_alarm_event_data_t**

GPTimer alarm event data.

Public Members

uint64_t **count_value**

Current count value

uint64_t **alarm_value**

Current alarm value

Type Definitions

typedef struct gptimer_t ***gptimer_handle_t**

Type of General Purpose Timer handle.

typedef bool (***gptimer_alarm_cb_t**)(*gptimer_handle_t* timer, const *gptimer_alarm_event_data_t* *edata, void *user_ctx)

Timer alarm callback prototype.

Param timer **[in]** Timer handle created by `gptimer_new_timer`
Param edata **[in]** Alarm event data, fed by driver
Param user_ctx **[in]** User data, passed from `gptimer_register_event_callbacks`
Return Whether a high priority task has been waken up by this function

Header File

- components/hal/include/hal/timer_types.h
- This header file can be included with:

```
#include "hal/timer_types.h"
```

Type Definitions

typedef *soc_periph_gptimer_clk_src_t* **gptimer_clock_source_t**

GPTimer clock source.

备注： User should select the clock source based on the power and resolution requirement

Enumerations

enum **gptimer_count_direction_t**

GPTimer count direction.

Values:

enumerator **GPTIMER_COUNT_DOWN**

Decrease count value

enumerator **GPTIMER_COUNT_UP**

Increase count value

enum **gptimer_etm_task_type_t**

GPTimer specific tasks that supported by the ETM module.

Values:

enumerator **GPTIMER_ETM_TASK_START_COUNT**

Start the counter

enumerator **GPTIMER_ETM_TASK_STOP_COUNT**

Stop the counter

enumerator **GPTIMER_ETM_TASK_EN_ALARM**

Enable the alarm

enumerator **GPTIMER_ETM_TASK_RELOAD**

Reload preset value into counter

enumerator **GPTIMER_ETM_TASK_CAPTURE**

Capture current count value into specific register

enumerator **GPTIMER_ETM_TASK_MAX**

Maximum number of tasks

enum **gptimer_etm_event_type_t**

GPTimer specific events that supported by the ETM module.

Values:

enumerator **GPTIMER_ETM_EVENT_ALARM_MATCH**

Count value matches the alarm target value

enumerator **GPTIMER_ETM_EVENT_MAX**

Maximum number of events

2.5.9 专用 GPIO

概述

专用 GPIO 专为 CPU 与 GPIO 矩阵和 IO MUX 交互而设计。任何配置为“专用”的 GPIO 都可以通过 CPU 指令直接访问，从而轻松提高 GPIO 翻转速度，方便用户以 bit-banging 的方式模拟串行/并行接口。通过 CPU 指令的方式控制 GPIO 的软件开销非常小，因此能够胜任一些特殊场合，比如通过示波器观测“GPIO 翻转信号”来间接测量某些性能指标。

创建/销毁 GPIO 捆绑包

GPIO 捆绑包是一组 GPIO，该组 GPIO 可以在一个 CPU 周期内同时操作。一个包能够包含 GPIO 的最大数量受每个 CPU 的限制。另外，GPIO 捆绑包与派生它的 CPU 有很强的相关性。**注意，任何对 GPIO 捆绑包操作的任务都必须运行在 GPIO 捆绑包所属的 CPU 内核**。同理，只有那些安装在同一个 CPU 内核上的 ISR 才允许对该 GPIO 捆绑包进行操作。

备注： 专用 GPIO 更像是 CPU 外设，因此与 CPU 内核关系密切。强烈建议在 pin-to-core 任务中安装和操作 GPIO 捆绑包。例如，如果 GPIOA 连接到了 CPU0，而专用的 GPIO 指令却是从 CPU1 发出的，那么就无法控制 GPIOA。

安装 GPIO 捆绑包需要调用 *dedic_gpio_new_bundle()* 来分配软件资源并将专用通道连接到用户选择的 GPIO。GPIO 捆绑包的配置在 *dedic_gpio_bundle_config_t* 结构体中：

- `gpio_array`：包含 GPIO 编号的数组。
- `array_size`：`gpio_array` 的元素个数。
- `flags`：用于控制 GPIO 捆绑包行为的标志。
  - `in_en` 和 `out_en` 用于选择是否开启输入输出功能（这两个功能可以同时开启）。
  - `in_invert` 和 `out_invert` 用于选择是否反转 GPIO 信号。

以下代码展示了如何安装只有输出功能的 GPIO 捆绑包：

```
// 配置 GPIO
const int bundleA_gpios[] = {0, 1};
gpio_config_t io_conf = {
    .mode = GPIO_MODE_OUTPUT,
};
for (int i = 0; i < sizeof(bundleA_gpios) / sizeof(bundleA_gpios[0]); i++) {
    io_conf.pin_bit_mask = 1ULL << bundleA_gpios[i];
    gpio_config(&io_conf);
}
// 创建 bundleA，仅输出
dedic_gpio_bundle_handle_t bundleA = NULL;
dedic_gpio_bundle_config_t bundleA_config = {
    .gpio_array = bundleA_gpios,
    .array_size = sizeof(bundleA_gpios) / sizeof(bundleA_gpios[0]),
    .flags = {
        .out_en = 1,
    },
};
ESP_ERROR_CHECK(dedic_gpio_new_bundle(&bundleA_config, &bundleA));
```

如需卸载 GPIO 捆绑包，可调用 *dedic_gpio_del_bundle()*。

备注： *dedic_gpio_new_bundle()* 不包含任何 GPIO pad 配置（例如上拉/下拉、驱动能力、输出/输入使能）。因此，在安装专用 GPIO 捆绑包之前，必须使用 GPIO 驱动程序 API（如 *gpio_config()*）单

独配置 GPIO。更多关于 GPIO 驱动的信息，请参考*GPIO API 参考*。

GPIO 捆绑包操作

操作	函数
以掩码的方式指定 GPIO 捆绑包的输出	*dedic_gpio_bundle_write()*
读取 GPIO 捆绑包实际输出的电平值	*dedic_gpio_bundle_read_out()*
读取 GPIO 捆绑包中输入的电平值	*dedic_gpio_bundle_read_in()*

备注： 由于函数调用的开销和内部涉及的位操作，使用上述函数可能无法获得较高的 GPIO 翻转速度。用户可以尝试*通过编写汇编代码操作 GPIO* 来减少开销，但应自行注意线程安全。

通过编写汇编代码操作 GPIO

高阶用户可以通过编写汇编代码或调用 CPU 低层 API 来操作 GPIO。常见步骤为：

1. 分配一个 GPIO 捆绑包：*dedic_gpio_new_bundle()*
2. 查询该包占用的掩码：*dedic_gpio_get_out_mask()* 和/或*dedic_gpio_get_in_mask()*
3. 调用 CPU LL apis（如 *cpu_ll_write_dedic_gpio_mask*）或使用该掩码编写汇编代码
4. 切换 IO 的最快捷方式是使用专用的“设置/清除”指令：

有关支持的专用 GPIO 指令的详细信息，请参考 **ESP32-P4 技术参考手册 > ESP-RISC-V CPU** [PDF]。

一些专用的 CPU 指令也包含在 `hal/dedic_gpio_cpu_ll.h` 中，作为辅助内联函数。

备注： 由于自定义指令在不同目标上可能会有不同的格式，在应用程序中编写汇编代码可能会让代码难以在不同的芯片架构之间移植。

应用示例

- 通过汇编代码使用专用的 CPU 指令来操作 GPIO 以模拟 UART/I2C/SPI 协议（Bit Banging）peripherals/dedicated_gpio.

API 参考

Header File

- components/esp_driver_gpio/include/driver/dedic_gpio.h
- This header file can be included with:

```
#include "driver/dedic_gpio.h"
```

- This header file is a part of the API provided by the `esp_driver_gpio` component. To declare that your component depends on `esp_driver_gpio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_gpio
```

or

```
PRIV_REQUIRES esp_driver_gpio
```

Functions

esp_err_t **dedic_gpio_get_out_mask**(*dedic_gpio_bundle_handle_t* bundle, uint32_t *mask)

Get allocated channel mask.

备注: Each bundle should have at least one mask (in or/and out), based on bundle configuration.

备注: With the returned mask, user can directly invoke LL function like "dedic_gpio_cpu_ll_write_mask" or write assembly code with dedicated GPIO instructions, to get better performance on GPIO manipulation.

参数

- **bundle** -- **[in]** Handle of GPIO bundle that returned from "dedic_gpio_new_bundle"
- **mask** -- **[out]** Returned mask value for on specific direction (in or out)

返回

- ESP_OK: Get channel mask successfully
- ESP_ERR_INVALID_ARG: Get channel mask failed because of invalid argument
- ESP_FAIL: Get channel mask failed because of other error

esp_err_t **dedic_gpio_get_in_mask**(*dedic_gpio_bundle_handle_t* bundle, uint32_t *mask)

esp_err_t **dedic_gpio_get_out_offset**(*dedic_gpio_bundle_handle_t* bundle, uint32_t *offset)

Get the channel offset of the GPIO bundle.

A GPIO bundle maps the GPIOS of a particular direction to a consecutive set of channels within a particular GPIO bank of a particular CPU. This function returns the offset to the bundle's first channel of a particular direction within the bank.

参数

- **bundle** -- **[in]** Handle of GPIO bundle that returned from "dedic_gpio_new_bundle"
- **offset** -- **[out]** Offset value to the first channel of a specific direction (in or out)

返回

- ESP_OK: Get channel offset successfully
- ESP_ERR_INVALID_ARG: Get channel offset failed because of invalid argument
- ESP_FAIL: Get channel offset failed because of other error

esp_err_t **dedic_gpio_get_in_offset**(*dedic_gpio_bundle_handle_t* bundle, uint32_t *offset)

esp_err_t **dedic_gpio_new_bundle**(const *dedic_gpio_bundle_config_t* *config, *dedic_gpio_bundle_handle_t* *ret_bundle)

Create GPIO bundle and return the handle.

备注: One has to enable at least input or output mode in "config" parameter.

参数

- **config** -- **[in]** Configuration of GPIO bundle
- **ret_bundle** -- **[out]** Returned handle of the new created GPIO bundle

返回

- ESP_OK: Create GPIO bundle successfully
- ESP_ERR_INVALID_ARG: Create GPIO bundle failed because of invalid argument
- ESP_ERR_NO_MEM: Create GPIO bundle failed because of no capable memory
- ESP_ERR_NOT_FOUND: Create GPIO bundle failed because of no enough continuous dedicated channels
- ESP_FAIL: Create GPIO bundle failed because of other error

esp_err_t **dedic_gpio_del_bundle**(*dedic_gpio_bundle_handle_t* bundle)

Destroy GPIO bundle.

参数 **bundle** -- **[in]** Handle of GPIO bundle that returned from "dedic_gpio_new_bundle"
返回
- ESP_OK: Destroy GPIO bundle successfully
- ESP_ERR_INVALID_ARG: Destroy GPIO bundle failed because of invalid argument
- ESP_FAIL: Destroy GPIO bundle failed because of other error

void **dedic_gpio_bundle_write**(*dedic_gpio_bundle_handle_t* bundle, uint32_t mask, uint32_t value)

Write value to GPIO bundle.

备注: The mask is seen from the view of GPIO bundle. For example, bundleA contains [GPIO10, GPIO12, GPIO17], to set GPIO17 individually, the mask should be 0x04.

备注: For performance reasons, this function doesn't check the validity of any parameters, and is placed in IRAM.

参数
- **bundle** -- **[in]** Handle of GPIO bundle that returned from "dedic_gpio_new_bundle"
- **mask** -- **[in]** Mask of the GPIOs to be written in the given bundle
- **value** -- **[in]** Value to write to given GPIO bundle, low bit represents low member in the bundle

uint32_t **dedic_gpio_bundle_read_out**(*dedic_gpio_bundle_handle_t* bundle)

Read the value that output from the given GPIO bundle.

备注: For performance reasons, this function doesn't check the validity of any parameters, and is placed in IRAM.

参数 **bundle** -- **[in]** Handle of GPIO bundle that returned from "dedic_gpio_new_bundle"
返回 Value that output from the GPIO bundle, low bit represents low member in the bundle

uint32_t **dedic_gpio_bundle_read_in**(*dedic_gpio_bundle_handle_t* bundle)

Read the value that input to the given GPIO bundle.

备注: For performance reasons, this function doesn't check the validity of any parameters, and is placed in IRAM.

参数 **bundle** -- **[in]** Handle of GPIO bundle that returned from "dedic_gpio_new_bundle"
返回 Value that input to the GPIO bundle, low bit represents low member in the bundle

Structures

struct **dedic_gpio_bundle_config_t**

Type of Dedicated GPIO bundle configuration.

Public Members

const int ***gpio_array**

Array of GPIO numbers, gpio_array[0] ~ gpio_array[size-1] <=> low_dedic_channel_num ~ high_dedic_channel_num

size_t **array_size**

Number of GPIOs in gpio_array

unsigned int **in_en**

Enable input

unsigned int **in_invert**

Invert input signal

unsigned int **out_en**

Enable output

unsigned int **out_invert**

Invert output signal

struct *dedic_gpio_bundle_config_t*::[anonymous] **flags**

Flags to control specific behaviour of GPIO bundle

Type Definitions

typedef struct dedic_gpio_bundle_t ***dedic_gpio_bundle_handle_t**

Type of Dedicated GPIO bundle.

2.5.10 哈希消息认证码 (HMAC)

哈希消息认证码 (HMAC) 是一种安全的身份认证技术，支持使用预共享的密钥验证消息的真实性和完整性。利用烧录在 eFuse 块中的密钥，HMAC 可以为生成 SHA256-HMAC 提供硬件加速。

更多有关应用操作流程，及 HMAC 计算过程的详细信息，请参阅 **ESP32-P4 技术参考手册 > HMAC 加速器 (HMAC)** [PDF]。

通用应用方案

现有 A、B 双方，他们都需要验证对方发送消息的真实性和完整性。那么在开始发送消息前，双方应通过安全通道交换密钥。

要验证来自 A 的信息，B 可遵循以下步骤：

- A 计算要发送的消息的 HMAC。
- A 将消息及消息的 HMAC 一并发送给 B。
- B 自行计算所接收消息的 HMAC。
- B 检查接收到的 HMAC 是否与计算得出的 HMAC 匹配。

若两个 HMAC 匹配，消息为真。

但 HMAC 本身还可以应用于更多场景，如支持 HMAC 的挑战-应答协议，或作为更多安全模块（详见下文）的密钥输入等。

ESP32-P4 上的 HMAC

在 ESP32-P4 HMAC 模块的 eFuse 中会烧录一个密钥，可将该 eFuse 密钥设置为禁止所有外部资源访问，避免密钥泄露。

此外，在 ESP32-P4 上的 HMAC 有以下三种应用场景：

1. HMAC 支持软件使用
2. HMAC 用作数字签名 (DS) 的密钥
3. HMAC 用于启用软禁用的 JTAG 接口

第一种应用场景称为 **上行**模式，后两种应用场景称为 **下行**模式。

HMAC 的 eFuse 密钥 在 ESP32-P4 中，有六个物理 eFuse 块可用作 HMAC 的密钥，分别是块 4 到块 9。在 API 中，枚举类型 *hmac_key_id_t* 将这些块映射为 `HMAC_KEY0` ~ `HMAC_KEY5`。

每个密钥都有相应的 eFuse 参数 **密钥功能 (key purpose)**，决定密钥应用于 HMAC 的哪种应用场景。

密钥功能	应用场景
8	HMAC 支持软件使用
7	HMAC 用作数字签名 (DS) 的密钥
6	HMAC 启用软禁用的 JTAG 接口
5	HMAC 既用作数字签名 (DS) 的密钥，又用于启用 JTAG 接口

这样一来，可以确保密钥用于原定场景。

要计算 HMAC，软件必须同时提供包含密钥的密钥块 ID，以及 **密钥功能**（详情请参阅 **ESP32-P4 技术参考手册 > eFuse 控制器 (eFuse)** [PDF]）。

在进行 HMAC 密钥计算前，HMAC 会验证软件所提供密钥块的功能。在软件所提供 ID 的对应密钥块中，eFuse 存储了密钥块的功能。只有当软件所提供密钥块的功能与 eFuse 中存储的密钥块功能匹配，才会继续进行计算。

HMAC 支持软件使用 密钥功能值：8

在此情况下，HMAC 支持软件使用，如验证消息真实性等。

API *esp_hmac_calculate()* 用于计算 HMAC。输入参数包括消息、消息长度以及包含密钥的 eFuse 密钥块 ID，且该密钥块的 eFuse 密钥功能设置为上行模式。

HMAC 用作数字签名 (DS) 的密钥 密钥功能值：7、5

HMAC 可用作密钥派生函数，解码数字签名模块使用的私钥参数。在此情况下，硬件使用标准信息进行计算。在 HMAC 部分只需提供 eFuse 密钥块和功能；而在数字签名模块则还需要一些额外参数。

无论是密钥还是实际的 HMAC，都不会暴露在 HMAC 和数字签名模块之外。对 HMAC 的计算，以及将其传递给数字签名模块的过程，均在内部进行。

详情请参阅 **ESP32-P4 技术参考手册 > 数字签名 (DS)** [PDF]。

HMAC 启用 JTAG 接口 密钥功能值：6、5

HMAC 的第三种应用场景是将其作为密钥，启用软禁用的 JTAG 接口。

重新启用 JTAG 接口的步骤如下：

第一步：设置

1. 生成一个 256 位的 HMAC 密钥，用于重新启用 JTAG。
2. 将步骤 1 获得的密钥写入 eFuse 块，且 eFuse 块的密钥功能参数应为 HMAC_DOWN_ALL (5) 或 HMAC_DOWN_JTAG (6)。为此，可以使用固件中的 `esp_efuse_write_key()` 函数，或使用主机上的 `espefuse.py` 完成操作。

3. 使用 `esp_efuse_set_read_protect()` 将 eFuse 密钥块配置为读保护，防止软件读取写入到 eFuse 密钥块中的 HMAC 密钥值。
4. 在烧录到 ESP32-P4 上时，将特定的位或位组设置为 `soft JTAG disable`。这样可以永久禁用 JTAG 接口，除非软件提供正确的密钥值进行验证。

备 注: API **esp_efuse_write_field_cnt(ESP_EFUSE_SOFT_DIS_JTAG, ESP_EFUSE_SOFT_DIS_JTAG[0]->bit_count)** 支持在 ESP32-P4 上烧录 `soft JTAG disable` 位。

备注: 置位 `DIS_PAD_JTAG` eFuse 时，JTAG 处于永久禁用状态，`SOFT_DIS_JTAG` 功能将失效。

启用 JTAG

1. 以 eFuse 中的密钥和 32 个 `0x00` 字节为输入，经过 HMAC-SHA256 函数处理，得到的函数输出结果即重新启用 JTAG 的密钥。
2. 从固件调用*esp_hmac_jtag_enable()* 函数时，传递上一步获取的密钥值。
3. 要在固件中重新禁用 JTAG，可以重置系统，或调用*esp_hmac_jtag_disable()*。

关于如何暂时禁用以及重新启用 JTAG 的完整示例，请参考 security/hmac_soft_jtag。

更多有关详情，请参阅 **ESP32-P4 技术参考手册 > HMAC 加速器 (HMAC)** [PDF]。

应用示例

以下为针对特定应用场景的实例代码，可用于设置 eFuse 密钥，并将其用于计算支持软件使用的 HMAC。

`esp_efuse_write_key` 可以设置 eFuse 中的物理密钥块 4，并设置其功能。`ESP_EFUSE_KEY_PURPOSE_HMAC_UP` (8) 表明，该密钥仅适用于生成支持软件使用的 HMAC。

```
#include "esp_efuse.h"

const uint8_t key_data[32] = { ... };

esp_err_t status = esp_efuse_write_key(EFUSE_BLK_KEY4,
                    ESP_EFUSE_KEY_PURPOSE_HMAC_UP,
                    key_data, sizeof(key_data));

if (status == ESP_OK) {
    // 密钥写入成功
} else {
    // 密钥写入失败，可能已写入过
}
```

接下来可以使用已存储的密钥来计算 HMAC，供软件使用。

```
#include "esp_hmac.h"

uint8_t hmac[32];

const char *message = "Hello, HMAC!";
const size_t msg_len = 12;

esp_err_t result = esp_hmac_calculate(HMAC_KEY4, message, msg_len, hmac);

if (result == ESP_OK) {
    // HMAC 已写入 hmac 数组
} else {
    // 计算 HMAC 失败
}
```

API 参考

Header File

- components/esp_hw_support/include/esp_hmac.h
- This header file can be included with:

```
#include "esp_hmac.h"
```

Functions

esp_err_t **esp_hmac_calculate**(*hmac_key_id_t* key_id, const void *message, size_t message_len, uint8_t *hmac)

Calculate the HMAC of a given message.

Calculate the HMAC `hmac` of a given message `message` with length `message_len`. SHA256 is used for the calculation.

备注： Uses the HMAC peripheral in ”upstream” mode.

参数

- **key_id** -- Determines which of the 6 key blocks in the efuses should be used for the HMAC calcuation. The corresponding purpose field of the key block in the efuse must be set to the HMAC upstream purpose value.
- **message** -- the message for which to calculate the HMAC
- **message_len** -- message length return ESP_ERR_INVALID_STATE if unsuccessful
- **hmac** -- **[out]** the hmac result; the buffer behind the provided pointer must be a writeable buffer of 32 bytes

返回

- ESP_OK, if the calculation was successful,
- ESP_ERR_INVALID_ARG if message or hmac is a nullptr or if key_id out of range
- ESP_FAIL, if the hmac calculation failed

esp_err_t **esp_hmac_jtag_enable**(*hmac_key_id_t* key_id, const uint8_t *token)

Use HMAC peripheral in Downstream mode to re-enable the JTAG, if it is not permanently disabled by HW. In downstream mode, HMAC calculations performed by peripheral are used internally and not provided back to user.

备注： Return value of the API does not indicate the JTAG status.

参数

- **key_id** -- Determines which of the 6 key blocks in the efuses should be used for the HMAC calculation. The corresponding purpose field of the key block in the efuse must be set to HMAC downstream purpose.
- **token** -- Pre calculated HMAC value of the 32-byte 0x00 using SHA-256 and the known private HMAC key. The key is already programmed to a eFuse key block. The key block number is provided as the first parameter to this function.

返回

- ESP_OK, if the key_purpose of the key_id matches to HMAC downstread mode, The API returns success even if calculated HMAC does not match with the provided token. However, The JTAG will be re-enabled only if the calculated HMAC value matches with provided token, otherwise JTAG will remain disabled.
- ESP_FAIL, if the key_purpose of the key_id is not set to HMAC downstream purpose or JTAG is permanently disabled by EFUSE_HARD_DIS_JTAG eFuse parameter.
- ESP_ERR_INVALID_ARG, invalid input arguments

esp_err_t **esp_hmac_jtag_disable**(void)

Disable the JTAG which might be enabled using the HMAC downstream mode. This function just clears the result generated by calling esp_hmac_jtag_enable() API.

返回

- ESP_OK return ESP_OK after writing the HMAC_SET_INVALIDATE_JTAG_REG with value 1.

Enumerations

enum **hmac_key_id_t**

The possible efuse keys for the HMAC peripheral

Values:

enumerator **HMAC_KEY0**

enumerator **HMAC_KEY1**

enumerator **HMAC_KEY2**

enumerator **HMAC_KEY3**

enumerator **HMAC_KEY4**

enumerator **HMAC_KEY5**

enumerator **HMAC_KEY_MAX**

2.5.11 数字签名 (DS)

数字签名 (DS) 模块利用 RSA 硬件加速器为信息签名。HMAC 作为密钥派生函数，使用 eFuse 作为输入密钥，输出一组加密参数。随后，数字签名模块利用这组预加密的参数，计算出签名。以上过程均在硬件中完成，因此在计算签名时，软件无法获取 RSA 参数的解密密钥，也无法获取 HMAC 密钥派生函数的输入密钥。

签名计算所涉及的硬件信息以及所用寄存器的有关信息，请参阅 **ESP32-P4 技术参考手册 > 数字签名 (DS)** [PDF]。

私钥参数

RSA 签名的私钥参数存储在 flash 中。为防止发生未经授权的访问，这些参数都进行了 AES 加密。此时，HMAC 模块被作为密钥派生函数，用于计算 AES 加密了的私钥参数。同时，HMAC 模块本身使用的来自 eFuse 密钥块的密钥也具有读取保护，可以防止发生未经授权的访问。

调用签名计算时，软件只需指定使用的 eFuse 密钥、相应的 eFuse 密钥用途、加密的 RSA 参数位置以及待签名的数据或信息。

密钥生成

在使用 DS 外设前，需首先创建并存储 HMAC 密钥和 RSA 私钥，此步骤可在 ESP32-P4 上通过软件完成，也可在主机上进行。在 ESP-IDF 中，可以使用*esp_efuse_write_block()* 设置 HMAC 密钥，并使用*esp_hmac_calculate()* 对 RSA 私钥参数进行加密。

计算并组装私钥参数的详细信息，请参阅 **ESP32-P4 技术参考手册 > 数字签名 (DS)** [PDF]。

在 ESP-IDF 中进行数字签名计算

在 ESP-IDF 中进行数字签名计算的工作流程，以及所用寄存器的有关信息，请参阅 **ESP32-P4 技术参考手册 > 数字签名 (DS)** [PDF]。

要进行数字签名计算，需要准备以下三个参数：

1. 用作 HMAC 密钥的 eFuse 密钥块 ID
2. 加密私钥参数的位置
3. 待签名的数据或信息

由于签名计算需要一些时间，ESP-IDF 提供了两种可用的 API。第一种是*esp_ds_sign()*，调用此 API 后，程序会在计算完成前保持阻塞状态。如果在计算过程中，软件需要执行其他操作，则可以调用*esp_ds_start_sign()*，用另一种方式启动签名计算，然后周期性地调用*esp_ds_is_busy()*，检查计算是否已完成。一旦计算完成，即可调用*esp_ds_finish_sign()* 来获取签名结果。

API *esp_ds_sign()* 和*esp_ds_start_sign()* 会借助 DS 外设计算明文 RSA 签名。RSA 签名需要转换成合适的格式，以供进一步使用。例如，MbedTLS SSL 栈支持 PKCS#1 格式，使用 API `esp_ds_rsa_sign()` 可以直接获得 PKCS#1 v1.5 格式的签名，该 API 内部调用了*esp_ds_start_sign()* 函数，并将签名转换成 PKCS#1 v1.5 格式。

备注： 此处只是最基本的 DS 构造块，其消息必须是固定长度。为在任意消息上创建签名，通常会将实际消息的哈希值作为输入，并将其填充到所需长度。乐鑫计划在未来提供一个 API 来实现这个功能。

TLS 连接所需的 DS 外设配置

在 ESP32-P4 芯片上使用 TLS 连接之前，必须先配置 DS 外设，具体步骤如下：

1) 随机生成一个 256 位的值，作为 *初始化向量* (IV)。
2) 随机生成一个 256 位的值，作为 `HMAC_KEY`。
3) 使用客户端私钥 (RAS) 和上述步骤生成的参数，计算加密的私钥参数。
4) 将 256 位的 `HMAC_KEY` 烧录到 eFuse 中，只支持 DS 外设读取。

更多细节，请参阅 **ESP32-P4 技术参考手册 > 数字签名 (DS)** [PDF]。

如果要以开发为目的配置 DS 外设，可以使用 esp-secure-cert-tool。

配置完 DS 外设后获取的加密私钥参数需要保存在 flash 中并传递给 DS 外设，DS 外设将使用这些参数完成数字签名。随后，应用程序需要从 flash 中读取 DS 数据，这可以通过 esp_secure_cert_mgr 组件提供的 API 完成。更多细节，请参阅 component/README。

在 esp_tls 仓库内部，*ESP-TLS* 负责完成初始化 DS 外设、执行数字签名的过程。更多细节，请参阅*ESP-TLS 的数字签名*。

如 *ESP-TLS* 文档所述，应用程序只需将加密私钥参数作为 *ds_data* 传递给 esp_tls 上下文，esp_tls 仓库内部就会执行所有必要操作，以初始化 DS 外设，并执行数字签名。

使用 DS 外设进行 SSL 双向认证

示例 protocols/mqtt/ssl_ds 展示了如何使用 DS 外设进行 SSL 双向认证。在示例中，使用了 *mqtt_client*（通过 *ESP-MQTT* 实现），通过 SSL 传输连接到代理服务器 `test.mosquitto.org`，并进行 SSL 双向认证。SSL 部分在内部使用 *ESP-TLS* 完成。更多细节，请参阅 protocols/mqtt/ssl_ds/README.md。

API 参考

Header File

- components/esp_hw_support/include/esp_ds.h
- This header file can be included with:

```
#include "esp_ds.h"
```

Functions

esp_err_t **esp_ds_sign**(const void *message, const *esp_ds_data_t* *data, *hmac_key_id_t* key_id, void *signature)

Sign the message with a hardware key from specific key slot. The function calculates a plain RSA signature with help of the DS peripheral. The RSA encryption operation is as follows: Z = XY mod M where, Z is the signature, X is the input message, Y and M are the RSA private key parameters.

This function is a wrapper around `esp_ds_finish_sign()` and `esp_ds_start_sign()`, so do not use them in parallel. It blocks until the signing is finished and then returns the signature.

备注： Please see note section of `esp_ds_start_sign()` for more details about the input parameters.

参数

- **message** -- the message to be signed; its length should be (data->rsa_length + 1)*4 bytes, and those bytes must be in little endian format. It is your responsibility to apply your hash function and padding before calling this function, if required. (e.g. message = padding(hash(inputMsg)))
- **data** -- the encrypted signing key data (AES encrypted RSA key + IV)
- **key_id** -- the HMAC key ID determining the HMAC key of the HMAC which will be used to decrypt the signing key data
- **signature** -- the destination of the signature, should be (data->rsa_length + 1)*4 bytes long

返回

- ESP_OK if successful, the signature was written to the parameter `signature`.
- ESP_ERR_INVALID_ARG if one of the parameters is NULL or data->rsa_length is too long or 0
- ESP_ERR_HW_CRYPTO_DS_HMAC_FAIL if there was an HMAC failure during retrieval of the decryption key
- ESP_ERR_NO_MEM if there hasn't been enough memory to allocate the context object
- ESP_ERR_HW_CRYPTO_DS_INVALID_KEY if there's a problem with passing the HMAC key to the DS component
- ESP_ERR_HW_CRYPTO_DS_INVALID_DIGEST if the message digest didn't match; the signature is invalid.
- ESP_ERR_HW_CRYPTO_DS_INVALID_PADDING if the message padding is incorrect, the signature can be read though since the message digest matches.

esp_err_t **esp_ds_start_sign**(const void *message, const *esp_ds_data_t* *data, *hmac_key_id_t* key_id, *esp_ds_context_t* **esp_ds_ctx)

Start the signing process.

This function yields a context object which needs to be passed to `esp_ds_finish_sign()` to finish the signing process. The function calculates a plain RSA signature with help of the DS peripheral. The RSA encryption operation is as follows: Z = XY mod M where, Z is the signature, X is the input message, Y and M are the RSA private key parameters.

备注： This function locks the HMAC, SHA, AES and RSA components, so the user has to ensure to call `esp_ds_finish_sign()` in a timely manner. The numbers Y, M, Rb which are a part of esp_ds_data_t

should be provided in little endian format and should be of length equal to the RSA private key bit length The message length in bits should also be equal to the RSA private key bit length. No padding is applied to the message automatically, Please ensure the message is appropriate padded before calling the API.

参数

- **message** -- the message to be signed; its length should be (data->rsa_length + 1)*4 bytes, and those bytes must be in little endian format. It is your responsibility to apply your hash function and padding before calling this function, if required. (e.g. message = padding(hash(inputMsg)))
- **data** -- the encrypted signing key data (AES encrypted RSA key + IV)
- **key_id** -- the HMAC key ID determining the HMAC key of the HMAC which will be used to decrypt the signing key data
- **esp_ds_ctx** -- the context object which is needed for finishing the signing process later

返回

- ESP_OK if successful, the ds operation was started now and has to be finished with `esp_ds_finish_sign()`
- ESP_ERR_INVALID_ARG if one of the parameters is NULL or data->rsa_length is too long or 0
- ESP_ERR_HW_CRYPTO_DS_HMAC_FAIL if there was an HMAC failure during retrieval of the decryption key
- ESP_ERR_NO_MEM if there hasn't been enough memory to allocate the context object
- ESP_ERR_HW_CRYPTO_DS_INVALID_KEY if there's a problem with passing the HMAC key to the DS component

bool **esp_ds_is_busy**(void)

Return true if the DS peripheral is busy, otherwise false.

备注: Only valid if `esp_ds_start_sign()` was called before.

---

esp_err_t **esp_ds_finish_sign**(void *signature, *esp_ds_context_t* *esp_ds_ctx)

Finish the signing process.

参数

- **signature** -- the destination of the signature, should be (data->rsa_length + 1)*4 bytes long, the resultant signature bytes shall be written in little endian format.
- **esp_ds_ctx** -- the context object retreived by `esp_ds_start_sign()`

返回

- ESP_OK if successful, the ds operation has been finished and the result is written to signature.
- ESP_ERR_INVALID_ARG if one of the parameters is NULL
- ESP_ERR_HW_CRYPTO_DS_INVALID_DIGEST if the message digest didn't match; the signature is invalid. This means that the encrypted RSA key parameters are invalid, indicating that they may have been tampered with or indicating a flash error, etc.
- ESP_ERR_HW_CRYPTO_DS_INVALID_PADDING if the message padding is incorrect, the signature can be read though since the message digest matches (see TRM for more details).

esp_err_t **esp_ds_encrypt_params**(*esp_ds_data_t* *data, const void *iv, const *esp_ds_p_data_t* *p_data, const void *key)

Encrypt the private key parameters.

The encryption is a prerequisite step before any signature operation can be done. It is not strictly necessary to use this encryption function, the encryption could also happen on an external device.

备注: The numbers Y, M, Rb which are a part of esp_ds_data_t should be provided in little endian format and should be of length equal to the RSA private key bit length The message length in bits should also be equal to

the RSA private key bit length. No padding is applied to the message automatically, Please ensure the message is appropriate padded before calling the API.

参数

- **data** -- Output buffer to store encrypted data, suitable for later use generating signatures.
- **iv** -- Pointer to 16 byte IV buffer, will be copied into 'data'. Should be randomly generated bytes each time.
- **p_data** -- Pointer to input plaintext key data. The expectation is this data will be deleted after this process is done and 'data' is stored.
- **key** -- Pointer to 32 bytes of key data. Type determined by key_type parameter. The expectation is the corresponding HMAC key will be stored to efuse and then permanently erased.

返回

- ESP_OK if successful, the ds operation has been finished and the result is written to signature.
- ESP_ERR_INVALID_ARG if one of the parameters is NULL or p_data->rsa_length is too long

Structures

struct **esp_digital_signature_data**

Encrypted private key data. Recommended to store in flash in this format.

备注： This struct has to match to one from the ROM code! This documentation is mostly taken from there.

Public Members

esp_digital_signature_length_t **rsa_length**

RSA LENGTH register parameters (number of words in RSA key & operands, minus one).

This value must match the length field encrypted and stored in 'c', or invalid results will be returned. (The DS peripheral will always use the value in 'c', not this value, so an attacker can't alter the DS peripheral results this way, it will just truncate or extend the message and the resulting signature in software.)

备注： In IDF, the enum type length is the same as of type unsigned, so they can be used interchangably. See the ROM code for the original declaration of struct `ets_ds_data_t`.

uint32_t **iv**[ESP_DS_IV_BIT_LEN / 32]

IV value used to encrypt 'c'

uint8_t **c**[ESP_DS_C_LEN]

Encrypted Digital Signature parameters. Result of AES-CBC encryption of plaintext values. Includes an encrypted message digest.

struct **esp_ds_p_data_t**

Plaintext parameters used by Digital Signature.

This is only used for encrypting the RSA parameters by calling esp_ds_encrypt_params(). Afterwards, the result can be stored in flash or in other persistent memory. The encryption is a prerequisite step before any signature operation can be done.

备注： Y, M, Rb, & M_Prime must all be in little endian format.

Public Members

uint32_t **Y**[ESP_DS_SIGNATURE_MAX_BIT_LEN / 32]

RSA exponent.

uint32_t **M**[ESP_DS_SIGNATURE_MAX_BIT_LEN / 32]

RSA modulus.

uint32_t **Rb**[ESP_DS_SIGNATURE_MAX_BIT_LEN / 32]

RSA r inverse operand.

uint32_t **M_prime**

RSA M prime operand.

uint32_t **length**

RSA length in words (32 bit)

Macros

ESP_DS_IV_BIT_LEN

ESP_DS_IV_LEN

ESP_DS_SIGNATURE_MAX_BIT_LEN

ESP_DS_SIGNATURE_MD_BIT_LEN

ESP_DS_SIGNATURE_M_PRIME_BIT_LEN

ESP_DS_SIGNATURE_L_BIT_LEN

ESP_DS_SIGNATURE_PADDING_BIT_LEN

ESP_DS_C_LEN

Type Definitions

typedef struct esp_ds_context **esp_ds_context_t**

typedef struct *esp_digital_signature_data* **esp_ds_data_t**

Encrypted private key data. Recommended to store in flash in this format.

备注： This struct has to match to one from the ROM code! This documentation is mostly taken from there.

Enumerations

enum **esp_digital_signature_length_t**

Values:

enumerator **ESP_DS_RSA_1024**

enumerator **ESP_DS_RSA_2048**

enumerator **ESP_DS_RSA_3072**

enumerator **ESP_DS_RSA_4096**

2.5.12 Inter-Integrated Circuit (I2C)

Introduction

I2C is a serial, synchronous, multi-device, half-duplex communication protocol that allows co-existence of multiple masters and slaves on the same bus. I2C uses two bidirectional open-drain lines: serial data line (SDA) and serial clock line (SCL), pulled up by resistors.

ESP32-P4 has 2 I2C controller (also called port), responsible for handling communication on the I2C bus. A single I2C controller can be a master or a slave.

Additionally, the ESP32-P4 chip has 1 low-power (LP) I2C controller. It is the cut-down version of regular I2C. Usually, the LP I2C controller only support basic I2C functionality with a much smaller RAM size, and does not support slave mode. For a full list of difference between HP I2C and LP I2C, please refer to the *ESP32-P4 Technical Reference Manual* > *I2C Controller (I2C)* > *Features* [PDF].

You can use LP I2C peripheral when HP I2C is not sufficient for users' usage. But please note again the LP I2C does not support all HP I2C functions. Please read docs before you use it.

Typically, an I2C slave device has a 7-bit address or 10-bit address. ESP32-P4 supports both I2C Standard-mode (Sm) and Fast-mode (Fm) which can go up to 100KHz and 400KHz respectively.

警告： The clock frequency of SCL in master mode should not be larger than 400 KHz

备注： The frequency of SCL is influenced by both the pull-up resistor and the wire capacitance. Therefore, users are strongly recommended to choose appropriate pull-up resistors to make the frequency accurate. The recommended value for pull-up resistors usually ranges from 1K Ohms to 10K Ohms.

Keep in mind that the higher the frequency, the smaller the pull-up resistor should be (but not less than 1 KOhms). Indeed, large resistors will decline the current, which will increase the clock switching time and reduce the frequency. We usually recommend a range of 2 KOhms to 5 KOhms, but users may also need to make some adjustments depending on their current draw requirements.

I2C Clock Configuration

- `i2c_clock_source_t::I2C_CLK_SRC_DEFAULT`: Default I2C source clock.
- `i2c_clock_source_t::I2C_CLK_SRC_XTAL`: External crystal for I2C clock source.
- `i2c_clock_source_t::I2C_CLK_RC_FAST`: Internal 20MHz rc oscillator for I2C clock source.

I2C File Structure

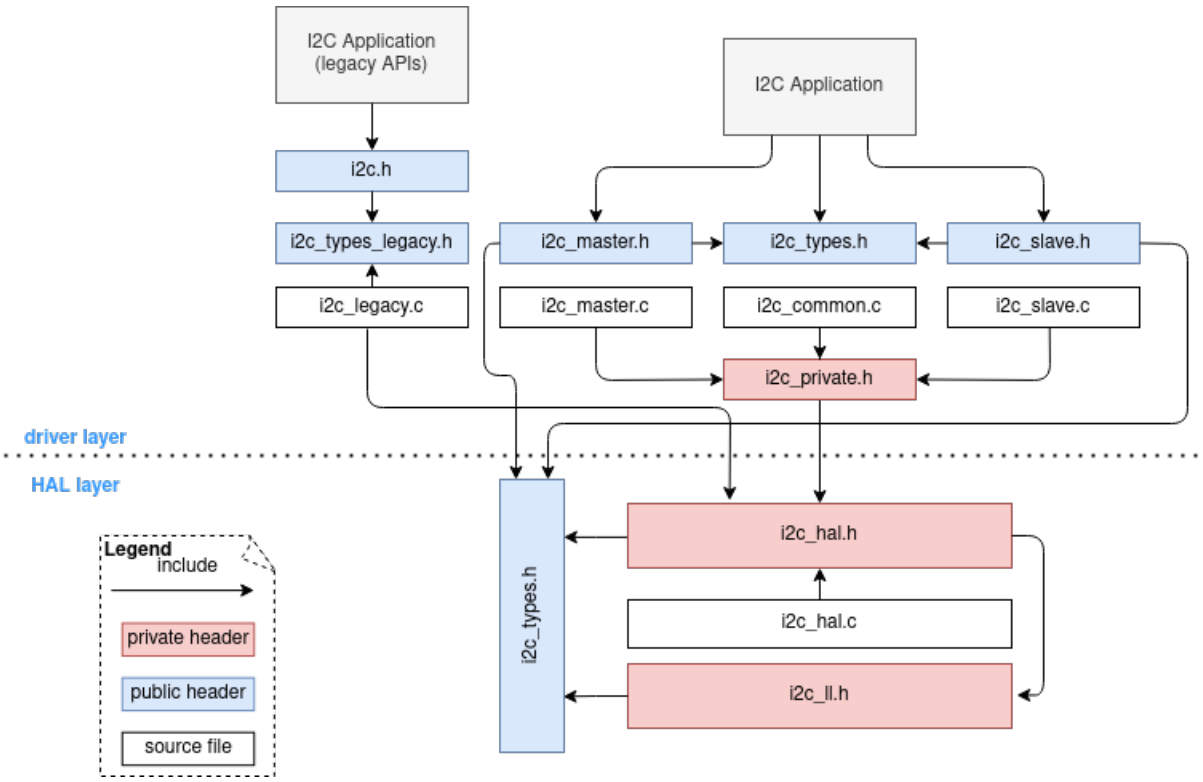


图 5: I2C file structure

Public headers that need to be included in the I2C application

- `i2c.h`: The header file of legacy I2C APIs (for apps using legacy driver).
- `i2c_master.h`: The header file that provides standard communication mode specific APIs (for apps using new driver with master mode).
- `i2c_slave.h`: The header file that provides standard communication mode specific APIs (for apps using new driver with slave mode).

备注： The legacy driver can't coexist with the new driver. Include `i2c.h` to use the legacy driver or the other two headers to use the new driver. Please keep in mind that the legacy driver is now deprecated and will be removed in future.

Public headers that have been included in the headers above

- `i2c_types_legacy.h`: The legacy public types that only used in the legacy driver.
- `i2c_types.h`: The header file that provides public types.

Functional Overview

The I2C driver offers following services:

- *Resource Allocation* - covers how to allocate I2C bus with properly set of configurations. It also covers how to recycle the resources when they finished working.
- *I2C Master Controller* - covers behavior of I2C master controller. Introduce data transmit, data receive, and data transmit and receive.
- *I2C Slave Controller* - covers behavior of I2C slave controller. Involve data transmit and data receive.
- *Power Management* - describes how different source clock will affect power consumption.
- *IRAM Safe* - describes tips on how to make the I2C interrupt work better along with a disabled cache.

- *Thread Safety* - lists which APIs are guaranteed to be thread safe by the driver.
- *Kconfig Options* - lists the supported Kconfig options that can bring different effects to the driver.

Resource Allocation Both I2C master bus and I2C slave bus, when supported, are represented by `i2c_bus_handle_t` in the driver. The available ports are managed in a resource pool that allocates a free port on request.

Install I2C master bus and device The I2C master is designed based on bus-device model. So `i2c_master_bus_config_t` and `i2c_device_config_t` are required separately to allocate the I2C master bus instance and I2C device instance.

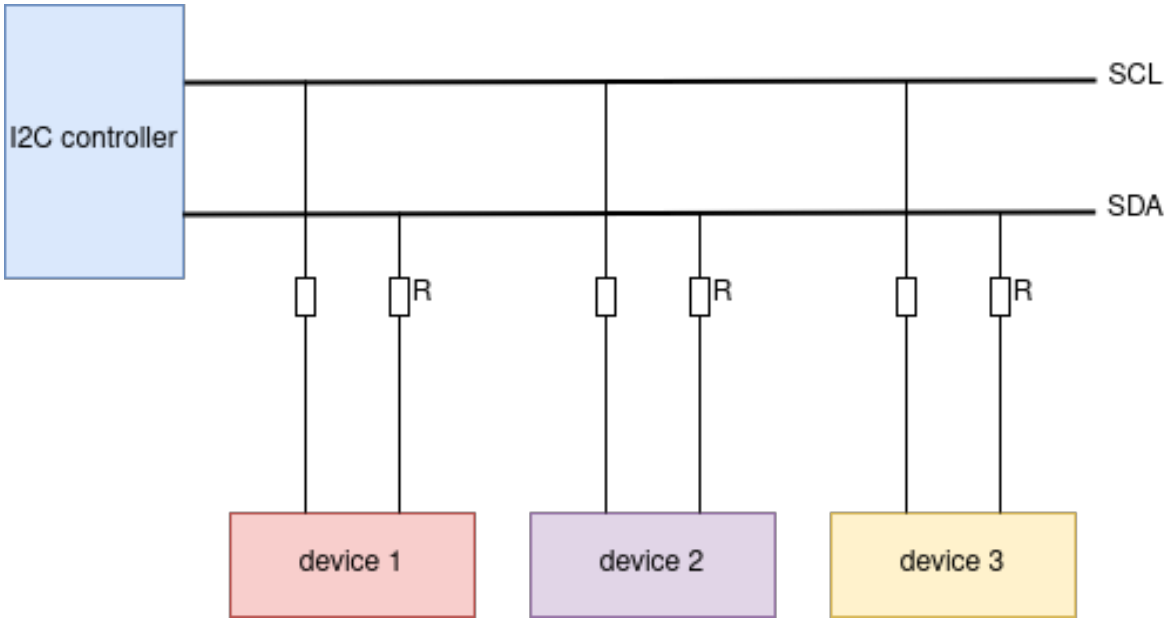


图 6: I2C master bus-device module

I2C master bus requires the configuration that specified by `i2c_master_bus_config_t`:

- `i2c_master_bus_config_t::i2c_port` sets the I2C port used by the controller.
- `i2c_master_bus_config_t::sda_io_num` sets the GPIO number for the serial data bus (SDA).
- `i2c_master_bus_config_t::scl_io_num` sets the GPIO number for the serial clock bus (SCL).
- `i2c_master_bus_config_t::clk_source` selects the source clock for I2C bus. The available clocks are listed in `i2c_clock_source_t`. For the effect on power consumption of different clock source, please refer to *Power Management* section.
- `i2c_master_bus_config_t::glitch_ignore_cnt` sets the glitch period of master bus, if the glitch period on the line is less than this value, it can be filtered out, typically value is 7.
- `i2c_master_bus_config_t::intr_priority` Set the priority of the interrupt. If set to `0` , then the driver will use a interrupt with low or medium priority (priority level may be one of 1,2 or 3), otherwise use the priority indicated by `i2c_master_bus_config_t::intr_priority` Please use the number form (1,2,3) , not the bitmask form ((1«1),(1«2),(1«3)).
- `i2c_master_bus_config_t::trans_queue_depth` Depth of internal transfer queue. Only valid in asynchronous transaction.
- `i2c_master_bus_config_t::enable_internal_pullup` Enable internal pullups. Note: This is not strong enough to pullup buses under high-speed frequency. A suitable external pullup is recommended.

If the configurations in `i2c_master_bus_config_t` is specified, users can call `i2c_new_master_bus()` to allocate and initialize an I2C master bus. This function will return an I2C bus handle if it runs correctly. Specifically, when there are no more I2C port available, this function will return `ESP_ERR_NOT_FOUND` error.

I2C master device requires the configuration that specified by `i2c_device_config_t`:

- *i2c_device_config_t::dev_addr_length* configure the address bit length of the slave device. User can choose from enumerator *I2C_ADDR_BIT_LEN_7* or *I2C_ADDR_BIT_LEN_10* (if supported).
- *i2c_device_config_t::device_address* I2C device raw address. Please parse the device address to this member directly. For example, the device address is 0x28, then parse 0x28 to *i2c_device_config_t::device_address*, don't carry a write/read bit.
- *i2c_device_config_t::scl_speed_hz* set the scl line frequency of this device.
- *i2c_device_config_t::scl_wait_us*. SCL await time (in us). Usually this value should not be very small because slave stretch will happen in pretty long time. (It's possible even stretch for 12ms). Set 0 means use default reg value.

Once the *i2c_device_config_t* structure is populated with mandatory parameters, users can call *i2c_master_bus_add_device()* to allocate an I2C device instance and mounted to the master bus then. This function will return an I2C device handle if it runs correctly. Specifically, when the I2C bus is not initialized properly, calling this function will result in a *ESP_ERR_INVALID_ARG* error.

```
#include "driver/i2c_master.h"

i2c_master_bus_config_t i2c_mst_config = {
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .i2c_port = TEST_I2C_PORT,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .glitch_ignore_cnt = 7,
    .flags.enable_internal_pullup = true,
};

i2c_master_bus_handle_t bus_handle;
ESP_ERROR_CHECK(i2c_new_master_bus(&i2c_mst_config, &bus_handle));

i2c_device_config_t dev_cfg = {
    .dev_addr_length = I2C_ADDR_BIT_LEN_7,
    .device_address = 0x58,
    .scl_speed_hz = 100000,
};

i2c_master_dev_handle_t dev_handle;
ESP_ERROR_CHECK(i2c_master_bus_add_device(bus_handle, &dev_cfg, &dev_handle));
```

Get I2C master handle via port Given that i2c master handle has been initialized in some module(e.g. an audio module), an another module(e.g. a video module) is not convenient to reused the handle. We have a helper function, *i2c_master_get_bus_handle()* for getting the initialized handle via port. However, please make sure the handle has already been initialized ahead of time. Otherwise an error would be reported.

```
// Source File 1
#include "driver/i2c_master.h"
i2c_master_bus_handle_t bus_handle;
i2c_master_bus_config_t i2c_mst_config = {
    ... // same as others
};
ESP_ERROR_CHECK(i2c_new_master_bus(&i2c_mst_config, &bus_handle));

// Source File 2
#include "driver/i2c_master.h"
i2c_master_bus_handle_t handle;
ESP_ERROR_CHECK(i2c_master_get_bus_handle(0, &handle));
```

Install I2C master bus with LP I2C Peripheral Install I2C master bus with LP I2C peripheral is almost as same as how HP I2C peripheral is installed. However, there are still some difference user should take focus on including IOs, clock sources, i2c port number, etc. Following code will show you how to install I2C master bus with LP_I2C.

```
#include "driver/i2c_master.h"

i2c_master_bus_config_t i2c_mst_config = {
    .clk_source = LP_I2C_SCLK_DEFAULT,    // clock source for LP i2c, might␣
↪different from HP I2C
    .i2c_port = LP_I2C_NUM_0,             // Assign to LP I2C port
    .scl_io_num = 7,                      // SCL IO number. Please refer to␣
↪technical reference manual
    .sda_io_num = 6,                      // SDA IO number. Please refer to␣
↪technical reference manual
    .glitch_ignore_cnt = 7,
    .flags.enable_internal_pullup = true,
};

i2c_master_bus_handle_t bus_handle;
ESP_ERROR_CHECK(i2c_new_master_bus(&i2c_mst_config, &bus_handle));

i2c_device_config_t dev_cfg = {
    .dev_addr_length = I2C_ADDR_BIT_LEN_7,
    .device_address = 0x58,
    .scl_speed_hz = 100000,
};

i2c_master_dev_handle_t dev_handle;
ESP_ERROR_CHECK(i2c_master_bus_add_device(bus_handle, &dev_cfg, &dev_handle));
```

Uninstall I2C master bus and device If a previously installed I2C bus or device is no longer needed, it's recommended to recycle the resource by calling *i2c_master_bus_rm_device()* or *i2c_del_master_bus()*, so that to release the underlying hardware.

Install I2C slave device I2C slave requires the configuration that specified by *i2c_slave_config_t*:

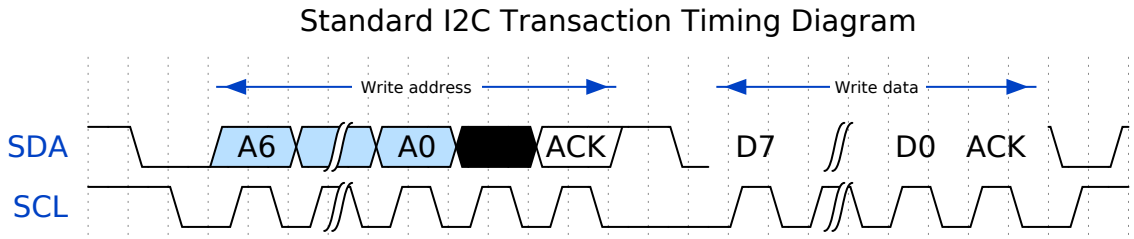
- *i2c_slave_config_t::i2c_port* sets the I2C port used by the controller.
- *i2c_slave_config_t::sda_io_num* sets the GPIO number for serial data bus (SDA).
- *i2c_slave_config_t::scl_io_num* sets the GPIO number for serial clock bus (SCL).
- *i2c_slave_config_t::clk_source* selects the source clock for I2C bus. The available clocks are listed in *i2c_clock_source_t*. For the effect on power consumption of different clock source, please refer to *Power Management* section.
- *i2c_slave_config_t::send_buf_depth* sets the sending buffer length.
- *i2c_slave_config_t::slave_addr* sets the slave address
- *i2c_master_bus_config_t::intr_priority* Set the priority of the interrupt. If set to `0` , then the driver will use a interrupt with low or medium priority (priority level may be one of 1,2 or 3), otherwise use the priority indicated by *i2c_master_bus_config_t::intr_priority* Please use the number form (1,2,3) , not the bitmask form ((1«1),(1«2),(1«3)). Please pay attention that once the interrupt priority is set, it cannot be changed until *i2c_del_master_bus()* is called.
- *i2c_slave_config_t::addr_bit_len* sets true if you need the slave to have a 10-bit address.
- *i2c_slave_config_t::access_ram_en* Set true to enable the non-fifo mode. Thus the I2C data fifo can be used as RAM, and double addressing will be synchronised opened.
- *i2c_slave_config_t::slave_unmatch_en* Set true to enable the slave unmatch interrupt. If master send command address cannot match the slave address, and unmatch interrupt will be triggered.

Once the *i2c_slave_config_t* structure is populated with mandatory parameters, users can call *i2c_new_slave_device()* to allocate and initialize an I2C master bus. This function will return an I2C bus handle if it runs correctly. Specifically, when there are no more I2C port available, this function will return *ESP_ERR_NOT_FOUND* error.

```
i2c_slave_config_t i2c_slv_config = {
    .addr_bit_len = I2C_ADDR_BIT_LEN_7,
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .i2c_port = TEST_I2C_PORT,
    .send_buf_depth = 256,
    .scl_io_num = I2C_SLAVE_SCL_IO,
    .sda_io_num = I2C_SLAVE_SDA_IO,
    .slave_addr = 0x58,
};

i2c_slave_dev_handle_t slave_handle;
ESP_ERROR_CHECK(i2c_new_slave_device(&i2c_slv_config, &slave_handle));
```

Uninstall I2C slave device If a previously installed I2C bus is no longer needed, it's recommended to recycle the resource by calling *i2c_del_slave_device()*, so that to release the underlying hardware.

I2C Master Controller After installing the i2c master driver by *i2c_new_master_bus()*, ESP32-P4 is ready to communicate with other I2C devices. I2C APIs allow the standard transactions. Like the wave as follows:



I2C Master Write After installing I2C master bus successfully, you can simply call *i2c_master_transmit()* to write data to the slave device. The principle of this function can be explained by following chart.

In order to organize the process, the driver uses a command link, that should be populated with a sequence of commands and then passed to I2C controller for execution.

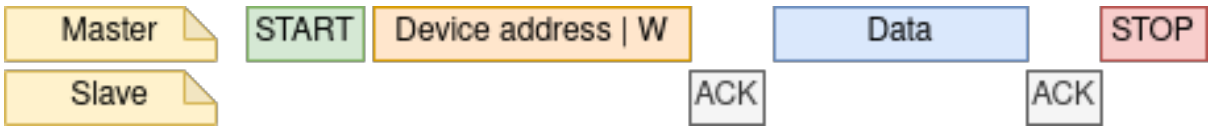


图 7: I2C master write to slave

Simple example for writing data to slave:

```
#define DATA_LENGTH 100
i2c_master_bus_config_t i2c_mst_config = {
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .i2c_port = I2C_PORT_NUM_0,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .glitch_ignore_cnt = 7,
};
i2c_master_bus_handle_t bus_handle;

ESP_ERROR_CHECK(i2c_new_master_bus(&i2c_mst_config, &bus_handle));

i2c_device_config_t dev_cfg = {
```

(下页继续)

(续上页)

```
    .dev_addr_length = I2C_ADDR_BIT_LEN_7,
    .device_address = 0x58,
    .scl_speed_hz = 100000,
};

i2c_master_dev_handle_t dev_handle;
ESP_ERROR_CHECK(i2c_master_bus_add_device(bus_handle, &dev_cfg, &dev_handle));

ESP_ERROR_CHECK(i2c_master_transmit(dev_handle, data_wr, DATA_LENGTH, -1));
```

I2C Master Read After installing I2C master bus successfully, you can simply call *i2c_master_receive()* to read data from the slave device. The principle of this function can be explained by following chart.

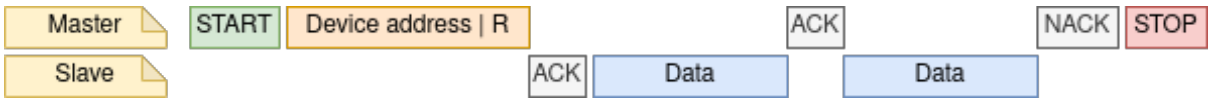


图 8: I2C master read from slave

Simple example for reading data from slave:

```
#define DATA_LENGTH 100
i2c_master_bus_config_t i2c_mst_config = {
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .i2c_port = I2C_PORT_NUM_0,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .glitch_ignore_cnt = 7,
};
i2c_master_bus_handle_t bus_handle;

ESP_ERROR_CHECK(i2c_new_master_bus(&i2c_mst_config, &bus_handle));

i2c_device_config_t dev_cfg = {
    .dev_addr_length = I2C_ADDR_BIT_LEN_7,
    .device_address = 0x58,
    .scl_speed_hz = 100000,
};

i2c_master_dev_handle_t dev_handle;
ESP_ERROR_CHECK(i2c_master_bus_add_device(bus_handle, &dev_cfg, &dev_handle));

i2c_master_receive(dev_handle, data_rd, DATA_LENGTH, -1);
```

I2C Master Write and Read Some I2C device needs write configurations before reading data from it, therefore, an interface called *i2c_master_transmit_receive()* can help. The principle of this function can be explained by following chart.

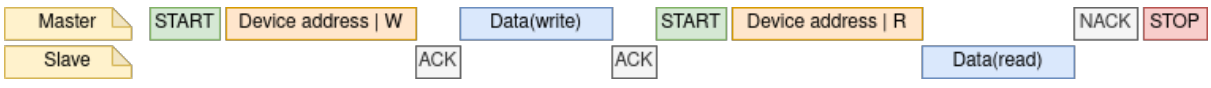


图 9: I2C master write to slave and read from slave

Simple example for writing and reading from slave:

```
i2c_device_config_t dev_cfg = {
    .dev_addr_length = I2C_ADDR_BIT_LEN_7,
```

(下页继续)

(续上页)

```
    .device_address = 0x58,
    .scl_speed_hz = 100000,
};

i2c_master_dev_handle_t dev_handle;
ESP_ERROR_CHECK(i2c_master_bus_add_device(I2C_PORT_NUM_0, &dev_cfg, &dev_handle));
uint8_t buf[20] = {0x20};
uint8_t buffer[2];
ESP_ERROR_CHECK(i2c_master_transmit_receive(dev_handle, buf, sizeof(buf), buffer,␣
↪2, -1));
```

I2C Master Probe I2C driver can use *i2c_master_probe()* to detect whether the specific device has been connected on I2C bus. If this function return ESP_OK, that means the device has been detected.

重要: Pull-ups must be connected to the SCL and SDA pins when this function is called. If you get *ESP_ERR_TIMEOUT* while *xfer_timeout_ms* was parsed correctly, you should check the pull-up resistors. If you do not have proper resistors nearby, setting *flags.enable_internal_pullup* as true is also acceptable.

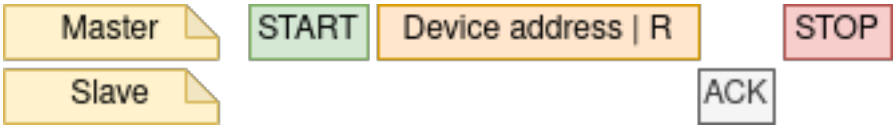


图 10: I2C master probe

Simple example for probing an I2C device:

```
i2c_master_bus_config_t i2c_mst_config_1 = {
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .i2c_port = TEST_I2C_PORT,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .glitch_ignore_cnt = 7,
    .flags.enable_internal_pullup = true,
};
i2c_master_bus_handle_t bus_handle;

ESP_ERROR_CHECK(i2c_new_master_bus(&i2c_mst_config_1, &bus_handle));
ESP_ERROR_CHECK(i2c_master_probe(bus_handle, 0x22, -1));
ESP_ERROR_CHECK(i2c_del_master_bus(bus_handle));
```

I2C Slave Controller After installing the i2c slave driver by *i2c_new_slave_device()*, ESP32-P4 is ready to communicate with other I2C master as a slave.

I2C Slave Write The send buffer of the I2C slave is used as a FIFO to store the data to be sent. The data will queue up until the master requests them. You can call *i2c_slave_transmit()* to transfer data.

Simple example for writing data to FIFO:

```
uint8_t *data_wr = (uint8_t *) malloc(DATA_LENGTH);

i2c_slave_config_t i2c_slv_config = {
    .addr_bit_len = I2C_ADDR_BIT_LEN_7,    // 7-bit address
    .clk_source = I2C_CLK_SRC_DEFAULT,     // set the clock source
    .i2c_port = 0,                         // set I2C port number
```

(下页继续)

(续上页)

```
    .send_buf_depth = 256,                  // set tx buffer length
    .scl_io_num = 2,                        // SCL gpio number
    .sda_io_num = 1,                        // SDA gpio number
    .slave_addr = 0x58,                     // slave address
};

i2c_bus_handle_t i2c_bus_handle;
ESP_ERROR_CHECK(i2c_new_slave_device(&i2c_slv_config, &i2c_bus_handle));
for (int i = 0; i < DATA_LENGTH; i++) {
    data_wr[i] = i;
}

ESP_ERROR_CHECK(i2c_slave_transmit(i2c_bus_handle, data_wr, DATA_LENGTH, 10000));
```

I2C Slave Read Whenever the master writes data to the slave, the slave will automatically store data in the receive buffer. This allows the slave application to call the function *i2c_slave_receive()* as its own discretion. As *i2c_slave_receive()* is designed as a non-blocking interface. So the user needs to register callback *i2c_slave_register_event_callbacks()* to know when the receive has finished.

```
static IRAM_ATTR bool i2c_slave_rx_done_callback(i2c_slave_dev_handle_t channel,␣
↪const i2c_slave_rx_done_event_data_t *edata, void *user_data)
{
    BaseType_t high_task_wakeup = pdFALSE;
    QueueHandle_t receive_queue = (QueueHandle_t)user_data;
    xQueueSendFromISR(receive_queue, edata, &high_task_wakeup);
    return high_task_wakeup == pdTRUE;
}

uint8_t *data_rd = (uint8_t *) malloc(DATA_LENGTH);
uint32_t size_rd = 0;

i2c_slave_config_t i2c_slv_config = {
    .addr_bit_len = I2C_ADDR_BIT_LEN_7,
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .i2c_port = TEST_I2C_PORT,
    .send_buf_depth = 256,
    .scl_io_num = I2C_SLAVE_SCL_IO,
    .sda_io_num = I2C_SLAVE_SDA_IO,
    .slave_addr = 0x58,
};

i2c_slave_dev_handle_t slave_handle;
ESP_ERROR_CHECK(i2c_new_slave_device(&i2c_slv_config, &slave_handle));

s_receive_queue = xQueueCreate(1, sizeof(i2c_slave_rx_done_event_data_t));
i2c_slave_event_callbacks_t cbs = {
    .on_recv_done = i2c_slave_rx_done_callback,
};
ESP_ERROR_CHECK(i2c_slave_register_event_callbacks(slave_handle, &cbs, s_receive_
↪queue));

i2c_slave_rx_done_event_data_t rx_data;
ESP_ERROR_CHECK(i2c_slave_receive(slave_handle, data_rd, DATA_LENGTH));
xQueueReceive(s_receive_queue, &rx_data, pdMS_TO_TICKS(10000));
// Receive done.
```

Put Data In I2C Slave RAM I2C slave fifo mentioned above can be used as RAM, which means user can access the RAM directly via address fields. For example, writing data to the 3rd ram block with following graph. Before using this, please note that *i2c_slave_config_t::access_ram_en* needs to be set to true.

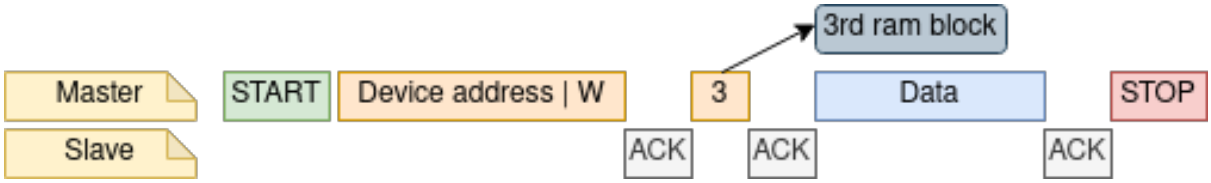


图 11: Put data in I2C slave RAM

```
uint8_t data_rd[DATA_LENGTH_RAM] = {0};

i2c_slave_config_t i2c_slv_config = {
    .addr_bit_len = I2C_ADDR_BIT_LEN_7,
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .i2c_port = TEST_I2C_PORT,
    .send_buf_depth = 256,
    .scl_io_num = I2C_SLAVE_SCL_IO,
    .sda_io_num = I2C_SLAVE_SDA_IO,
    .slave_addr = 0x58,
    .flags.access_ram_en = true,
};

// Master write to slave.

i2c_slave_dev_handle_t slave_handle;
ESP_ERROR_CHECK(i2c_new_slave_device(&i2c_slv_config, &slave_handle));
ESP_ERROR_CHECK(i2c_slave_read_ram(slave_handle, 0x5, data_rd, DATA_LENGTH_RAM));
ESP_ERROR_CHECK(i2c_del_slave_device(slave_handle));
```

Get Data From I2C Slave RAM Data can be stored in the RAM with a specific offset by the slave controller, and the master can read this data directly via the RAM address. For example, if the data is stored in 3rd ram block, master can read this data by following graph. Before using this, please note that *i2c_slave_config_t::access_ram_en* needs to be set to true.

图 12: Get data from I2C slave RAM

```
uint8_t data_wr[DATA_LENGTH_RAM] = {0};

i2c_slave_config_t i2c_slv_config = {
    .addr_bit_len = I2C_ADDR_BIT_LEN_7,
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .i2c_port = TEST_I2C_PORT,
    .send_buf_depth = 256,
    .scl_io_num = I2C_SLAVE_SCL_IO,
    .sda_io_num = I2C_SLAVE_SDA_IO,
    .slave_addr = 0x58,
    .flags.access_ram_en = true,
};

i2c_slave_dev_handle_t slave_handle;
ESP_ERROR_CHECK(i2c_new_slave_device(&i2c_slv_config, &slave_handle));
ESP_ERROR_CHECK(i2c_slave_write_ram(slave_handle, 0x2, data_wr, DATA_LENGTH_RAM));
ESP_ERROR_CHECK(i2c_del_slave_device(slave_handle));
```

Register Event Callbacks

I2C master callbacks When an I2C master bus triggers an interrupt, a specific event will be generated and notify the CPU. If you have some functions that need to be called when those events occurred, you can hook your functions to the ISR (Interrupt Service Routine) by calling *i2c_master_register_event_callbacks()*. Since the registered callback functions are called in the interrupt context, user should ensure the callback function doesn't attempt to block (e.g. by making sure that only FreeRTOS APIs with `ISR` suffix are called from within the function). The callback functions are required to return a boolean value, to tell the ISR whether a high priority task is woke up by it.

I2C master event callbacks are listed in the *i2c_master_event_callbacks_t*.

Although I2C is a synchronous communication protocol, we also support asynchronous behavior by registering above callback. In this way, I2C APIs will be non-blocking interface. But note that on the same bus, only one device can adopt asynchronous operation.

重要: I2C master asynchronous transaction is still an experimental feature. (The issue is when asynchronous transaction is very large, it will cause memory problem.)

- `i2c_master_event_callbacks_t::on_recv_done` sets a callback function for master "transaction-done" event. The function prototype is declared in *i2c_master_callback_t*.

I2C slave callbacks When an I2C slave bus triggers an interrupt, a specific event will be generated and notify the CPU. If you have some function that needs to be called when those events occurred, you can hook your function to the ISR (Interrupt Service Routine) by calling *i2c_slave_register_event_callbacks()*. Since the registered callback functions are called in the interrupt context, user should ensure the callback function doesn't attempt to block (e.g. by making sure that only FreeRTOS APIs with `ISR` suffix are called from within the function). The callback function has a boolean return value, to tell the caller whether a high priority task is woke up by it.

I2C slave event callbacks are listed in the *i2c_slave_event_callbacks_t*.

- *i2c_slave_event_callbacks_t::on_recv_done* sets a callback function for "receive-done" event. The function prototype is declared in *i2c_slave_received_callback_t*.

Power Management If the controller clock source is selected to *I2C_CLK_SRC_XTAL*, then the driver won't install power management lock for it, which is more suitable for a low power application as long as the source clock can still provide sufficient resolution.

IRAM Safe By default, the I2C interrupt will be deferred when the Cache is disabled for reasons like writing/erasing Flash. Thus the event callback functions will not get executed in time, which is not expected in a real-time application.

There's a Kconfig option *CONFIG_I2C_ISR_IRAM_SAFE* that will:

1. Enable the interrupt being serviced even when cache is disabled
2. Place all functions that used by the ISR into IRAM
3. Place driver object into DRAM (in case it's mapped to PSRAM by accident)

This will allow the interrupt to run while the cache is disabled but will come at the cost of increased IRAM consumption.

Thread Safety The factory function *i2c_new_master_bus()* and *i2c_new_slave_device()* are guaranteed to be thread safe by the driver, which means, user can call them from different RTOS tasks without protection by extra locks. Other public I2C APIs are not thread safe. which means the user should avoid calling them from multiple tasks, if user strongly needs to call them in multiple tasks, please add extra lock.

Kconfig Options

- *CONFIG_I2C_ISR_IRAM_SAFE* controls whether the default ISR handler can work when cache is disabled, see also *IRAM Safe* for more information.
- *CONFIG_I2C_ENABLE_DEBUG_LOG* is used to enable the debug log at the cost of increased firmware binary size.

API Reference

Header File

- components/esp_driver_i2c/include/driver/i2c_master.h
- This header file can be included with:

```
#include "driver/i2c_master.h"
```

- This header file is a part of the API provided by the `esp_driver_i2c` component. To declare that your component depends on `esp_driver_i2c`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_i2c
```

or

```
PRIV_REQUIRES esp_driver_i2c
```

Functions

esp_err_t **i2c_new_master_bus**(const *i2c_master_bus_config_t* *bus_config, *i2c_master_bus_handle_t* *ret_bus_handle)

Allocate an I2C master bus.

参数

- **bus_config** -- **[in]** I2C master bus configuration.
- **ret_bus_handle** -- **[out]** I2C bus handle

返回

- ESP_OK: I2C master bus initialized successfully.
- ESP_ERR_INVALID_ARG: I2C bus initialization failed because of invalid argument.
- ESP_ERR_NO_MEM: Create I2C bus failed because of out of memory.
- ESP_ERR_NOT_FOUND: No more free bus.

esp_err_t **i2c_master_bus_add_device**(*i2c_master_bus_handle_t* bus_handle, const *i2c_device_config_t* *dev_config, *i2c_master_dev_handle_t* *ret_handle)

Add I2C master BUS device.

参数

- **bus_handle** -- **[in]** I2C bus handle.
- **dev_config** -- **[in]** device config.
- **ret_handle** -- **[out]** device handle.

返回

- ESP_OK: Create I2C master device successfully.
- ESP_ERR_INVALID_ARG: I2C bus initialization failed because of invalid argument.
- ESP_ERR_NO_MEM: Create I2C bus failed because of out of memory.

esp_err_t **i2c_del_master_bus**(*i2c_master_bus_handle_t* bus_handle)

Deinitialize the I2C master bus and delete the handle.

参数 **bus_handle** -- **[in]** I2C bus handle.

返回

- ESP_OK: Delete I2C bus success, otherwise, failed.
- Otherwise: Some module delete failed.

esp_err_t **i2c_master_bus_rm_device**(*i2c_master_dev_handle_t* handle)

I2C master bus delete device.

参数 **handle** -- i2c device handle

返回

- ESP_OK: If device is successfully deleted.

esp_err_t **i2c_master_transmit**(*i2c_master_dev_handle_t* i2c_dev, const uint8_t *write_buffer, size_t write_size, int xfer_timeout_ms)

Perform a write transaction on the I2C bus. The transaction will be undergoing until it finishes or it reaches the timeout provided.

备注: If a callback was registered with `i2c_master_register_event_callbacks`, the transaction will be asynchronous, and thus, this function will return directly, without blocking. You will get finish information from callback. Besides, data buffer should always be completely prepared when callback is registered, otherwise, the data will get corrupt.

参数

- **i2c_dev** -- **[in]** I2C master device handle that created by `i2c_master_bus_add_device`.
- **write_buffer** -- **[in]** Data bytes to send on the I2C bus.
- **write_size** -- **[in]** Size, in bytes, of the write buffer.
- **xfer_timeout_ms** -- **[in]** Wait timeout, in ms. Note: -1 means wait forever.

返回

- ESP_OK: I2C master transmit success
- ESP_ERR_INVALID_ARG: I2C master transmit parameter invalid.
- ESP_ERR_TIMEOUT: Operation timeout(larger than xfer_timeout_ms) because the bus is busy or hardware crash.

esp_err_t **i2c_master_transmit_receive**(*i2c_master_dev_handle_t* i2c_dev, const uint8_t *write_buffer, size_t write_size, uint8_t *read_buffer, size_t read_size, int xfer_timeout_ms)

Perform a write-read transaction on the I2C bus. The transaction will be undergoing until it finishes or it reaches the timeout provided.

备注: If a callback was registered with `i2c_master_register_event_callbacks`, the transaction will be asynchronous, and thus, this function will return directly, without blocking. You will get finish information from callback. Besides, data buffer should always be completely prepared when callback is registered, otherwise, the data will get corrupt.

参数

- **i2c_dev** -- **[in]** I2C master device handle that created by `i2c_master_bus_add_device`.
- **write_buffer** -- **[in]** Data bytes to send on the I2C bus.
- **write_size** -- **[in]** Size, in bytes, of the write buffer.
- **read_buffer** -- **[out]** Data bytes received from i2c bus.
- **read_size** -- **[in]** Size, in bytes, of the read buffer.
- **xfer_timeout_ms** -- **[in]** Wait timeout, in ms. Note: -1 means wait forever.

返回

- ESP_OK: I2C master transmit-receive success
- ESP_ERR_INVALID_ARG: I2C master transmit parameter invalid.
- ESP_ERR_TIMEOUT: Operation timeout(larger than xfer_timeout_ms) because the bus is busy or hardware crash.

esp_err_t **i2c_master_receive**(*i2c_master_dev_handle_t* i2c_dev, uint8_t *read_buffer, size_t read_size, int xfer_timeout_ms)

Perform a read transaction on the I2C bus. The transaction will be undergoing until it finishes or it reaches the timeout provided.

备注: If a callback was registered with `i2c_master_register_event_callbacks`, the transaction will be asynchronous, and thus, this function will return directly, without blocking. You will get finish information from callback. Besides, data buffer should always be completely prepared when callback is registered, otherwise, the data will get corrupt.

参数

- **i2c_dev** -- **[in]** I2C master device handle that created by `i2c_master_bus_add_device`.
- **read_buffer** -- **[out]** Data bytes received from i2c bus.
- **read_size** -- **[in]** Size, in bytes, of the read buffer.
- **xfer_timeout_ms** -- **[in]** Wait timeout, in ms. Note: -1 means wait forever.

返回

- ESP_OK: I2C master receive success
- ESP_ERR_INVALID_ARG: I2C master receive parameter invalid.
- ESP_ERR_TIMEOUT: Operation timeout(larger than xfer_timeout_ms) because the bus is busy or hardware crash.

esp_err_t **i2c_master_probe**(*i2c_master_bus_handle_t* bus_handle, uint16_t address, int xfer_timeout_ms)

Probe I2C address, if address is correct and ACK is received, this function will return ESP_OK.

Attention Pull-ups must be connected to the SCL and SDA pins when this function is called. If you get `ESP_ERR_TIMEOUT whilexfer_timeout_mswas parsed correctly, you should check the pull-up resistors. If you do not have proper resistors nearby.` flags.enable_internal_pullup‘ is also acceptable.

备注: The principle of this function is to sent device address with a write command. If the device on your I2C bus, there would be an ACK signal and function returns `ESP_OK`. If the device is not on your I2C bus, there would be a NACK signal and function returns `ESP_ERR_NOT_FOUND`. `ESP_ERR_TIMEOUT` is not an expected failure, which indicated that the i2c probe not works properly, usually caused by pull-up resistors not be connected properly. Suggestion check data on SDA/SCL line to see whether there is ACK/NACK signal is on line when i2c probe function fails.

备注: There are lots of I2C devices all over the world, we assume that not all I2C device support the behavior like `device_address+nack/ack`. So, if the on line data is strange and no ack/nack got respond. Please check the device datasheet.

参数

- **bus_handle** -- **[in]** I2C master device handle that created by `i2c_master_bus_add_device`.
- **address** -- **[in]** I2C device address that you want to probe.
- **xfer_timeout_ms** -- **[in]** Wait timeout, in ms. Note: -1 means wait forever (Not recommended in this function).

返回

- ESP_OK: I2C device probe successfully
- ESP_ERR_NOT_FOUND: I2C probe failed, doesn't find the device with specific address you gave.

- ESP_ERR_TIMEOUT: Operation timeout(larger than xfer_timeout_ms) because the bus is busy or hardware crash.

esp_err_t **i2c_master_register_event_callbacks**(*i2c_master_dev_handle_t* i2c_dev, const *i2c_master_event_callbacks_t* *cbs, void *user_data)

Register I2C transaction callbacks for a master device.

备注： User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

备注： When CONFIG_I2C_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. The `user_data` should also reside in SRAM.

备注： If the callback is used for helping asynchronous transaction. On the same bus, only one device can be used for performing asynchronous operation.

参数

- **i2c_dev** -- **[in]** I2C master device handle that created by `i2c_master_bus_add_device`.
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set I2C transaction callbacks successfully
- ESP_ERR_INVALID_ARG: Set I2C transaction callbacks failed because of invalid argument
- ESP_FAIL: Set I2C transaction callbacks failed because of other error

esp_err_t **i2c_master_bus_reset**(*i2c_master_bus_handle_t* bus_handle)

Reset the I2C master bus.

参数 **bus_handle** -- I2C bus handle.

返回

- ESP_OK: Reset succeed.
- ESP_ERR_INVALID_ARG: I2C master bus handle is not initialized.
- Otherwise: Reset failed.

esp_err_t **i2c_master_bus_wait_all_done**(*i2c_master_bus_handle_t* bus_handle, int timeout_ms)

Wait for all pending I2C transactions done.

参数

- **bus_handle** -- **[in]** I2C bus handle
- **timeout_ms** -- **[in]** Wait timeout, in ms. Specially, -1 means to wait forever.

返回

- ESP_OK: Flush transactions successfully
- ESP_ERR_INVALID_ARG: Flush transactions failed because of invalid argument
- ESP_ERR_TIMEOUT: Flush transactions failed because of timeout
- ESP_FAIL: Flush transactions failed because of other error

esp_err_t **i2c_master_get_bus_handle**(*i2c_port_num_t* port_num, *i2c_master_bus_handle_t* *ret_handle)

Retrieves the I2C master bus handle for a specified I2C port number.

This function retrieves the I2C master bus handle for the given I2C port number. Please make sure the handle has already been initialized, and this function would simply returns the existing handle. Note that the returned handle still can't be used concurrently

参数

- **port_num** -- I2C port number for which the handle is to be retrieved.
- **ret_handle** -- Pointer to a variable where the retrieved handle will be stored.

返回

- ESP_OK: Success. The handle is retrieved successfully.
- ESP_ERR_INVALID_ARG: Invalid argument, such as invalid port number
- ESP_ERR_INVALID_STATE: Invalid state, such as the I2C port is not initialized.

Structures

struct **i2c_master_bus_config_t**

I2C master bus specific configurations.

Public Members

i2c_port_num_t **i2c_port**

I2C port number, `-1` for auto selecting, (not include LP I2C instance)

gpio_num_t **sda_io_num**

GPIO number of I2C SDA signal, pulled-up internally

gpio_num_t **scl_io_num**

GPIO number of I2C SCL signal, pulled-up internally

i2c_clock_source_t **clk_source**

Clock source of I2C master bus

lp_i2c_clock_source_t **lp_source_clk**

LP_UART source clock selection

uint8_t **glitch_ignore_cnt**

If the glitch period on the line is less than this value, it can be filtered out, typically value is 7 (unit: I2C module clock cycle)

int **intr_priority**

I2C interrupt priority, if set to 0, driver will select the default priority (1,2,3).

size_t **trans_queue_depth**

Depth of internal transfer queue, increase this value can support more transfers pending in the background, only valid in asynchronous transaction. (Typically max_device_num * per_transaction)

uint32_t **enable_internal_pullup**

Enable internal pullups. Note: This is not strong enough to pullup buses under high-speed frequency. Recommend proper external pull-up if possible

struct *i2c_master_bus_config_t*::[anonymous] **flags**

I2C master config flags

struct **i2c_device_config_t**

I2C device configuration.

Public Members

i2c_addr_bit_len_t **dev_addr_length**

Select the address length of the slave device.

uint16_t **device_address**

I2C device raw address. (The 7/10 bit address without read/write bit)

uint32_t **scl_speed_hz**

I2C SCL line frequency.

uint32_t **scl_wait_us**

Timeout value. (unit: us). Please note this value should not be so small that it can handle stretch/disturbance properly. If 0 is set, that means use the default reg value

uint32_t **disable_ack_check**

Disable ACK check. If this is set false, that means ack check is enabled, the transaction will be stopped and API returns error when nack is detected.

struct *i2c_device_config_t*::[anonymous] **flags**

I2C device config flags

struct **i2c_master_event_callbacks_t**

Group of I2C master callbacks, can be used to get status during transaction or doing other small things. But take care potential concurrency issues.

备注： The callbacks are all running under ISR context

备注： When CONFIG_I2C_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

Public Members

i2c_master_callback_t **on_trans_done**

I2C master transaction finish callback

Header File

- components/esp_driver_i2c/include/driver/i2c_slave.h
- This header file can be included with:

```
#include "driver/i2c_slave.h"
```

- This header file is a part of the API provided by the `esp_driver_i2c` component. To declare that your component depends on `esp_driver_i2c`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_i2c
```

or

```
PRIV_REQUIRES esp_driver_i2c
```

Functions

esp_err_t **i2c_new_slave_device**(const *i2c_slave_config_t* *slave_config, *i2c_slave_dev_handle_t* *ret_handle)

Initialize an I2C slave device.

参数

- **slave_config** -- **[in]** I2C slave device configurations
- **ret_handle** -- **[out]** Return a generic I2C device handle

返回

- ESP_OK: I2C slave device initialized successfully
- ESP_ERR_INVALID_ARG: I2C device initialization failed because of invalid argument.
- ESP_ERR_NO_MEM: Create I2C device failed because of out of memory.

esp_err_t **i2c_del_slave_device**(*i2c_slave_dev_handle_t* i2c_slave)

Deinitialize the I2C slave device.

参数 **i2c_slave** -- **[in]** I2C slave device handle that created by `i2c_new_slave_device`.

返回

- ESP_OK: Delete I2C device successfully.
- ESP_ERR_INVALID_ARG: I2C device initialization failed because of invalid argument.

esp_err_t **i2c_slave_receive**(*i2c_slave_dev_handle_t* i2c_slave, uint8_t *data, size_t buffer_size)

Read bytes from I2C internal buffer. Start a job to receive I2C data.

备 注: This function is non-blocking, it initiates a new receive job and then returns. User should check the received data from the `on_recv_done` callback that registered by `i2c_slave_register_event_callbacks()`.

参数

- **i2c_slave** -- **[in]** I2C slave device handle that created by `i2c_new_slave_device`.
- **data** -- **[out]** Buffer to store data from I2C fifo. Should be valid until `on_recv_done` is triggered.
- **buffer_size** -- **[in]** Buffer size of data that provided by users.

返回

- ESP_OK: I2C slave receive success.
- ESP_ERR_INVALID_ARG: I2C slave receive parameter invalid.
- ESP_ERR_NOT_SUPPORTED: This function should be work in fifo mode, but I2C_SLAVE_NONFIFO mode is configured

esp_err_t **i2c_slave_transmit**(*i2c_slave_dev_handle_t* i2c_slave, const uint8_t *data, int size, int xfer_timeout_ms)

Write bytes to internal ringbuffer of the I2C slave data. When the TX fifo empty, the ISR will fill the hardware FIFO with the internal ringbuffer's data.

备注: If you connect this slave device to some master device, the data transaction direction is from slave device to master device.

参数

- **i2c_slave** -- **[in]** I2C slave device handle that created by `i2c_new_slave_device`.
- **data** -- **[in]** Buffer to write to slave fifo, can pickup by master. Can be freed after this function returns. Equal or larger than `size`.
- **size** -- **[in]** In bytes, of `data` buffer.
- **xfer_timeout_ms** -- **[in]** Wait timeout, in ms. Note: -1 means wait forever.

返回

- ESP_OK: I2C slave transmit success.
- ESP_ERR_INVALID_ARG: I2C slave transmit parameter invalid.
- ESP_ERR_TIMEOUT: Operation timeout(larger than xfer_timeout_ms) because the device is busy or hardware crash.
- ESP_ERR_NOT_SUPPORTED: This function should be work in fifo mode, but I2C_SLAVE_NONFIFO mode is configured

esp_err_t **i2c_slave_register_event_callbacks**(*i2c_slave_dev_handle_t* i2c_slave, const *i2c_slave_event_callbacks_t* *cbs, void *user_data)

Set I2C slave event callbacks for I2C slave channel.

备注： User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

备注： When CONFIG_I2C_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. The `user_data` should also reside in SRAM.

参数

- **i2c_slave** -- **[in]** I2C slave device handle that created by `i2c_new_slave_device`.
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set I2C transaction callbacks successfully
- ESP_ERR_INVALID_ARG: Set I2C transaction callbacks failed because of invalid argument
- ESP_FAIL: Set I2C transaction callbacks failed because of other error

esp_err_t **i2c_slave_read_ram**(*i2c_slave_dev_handle_t* i2c_slave, uint8_t ram_address, uint8_t *data, size_t receive_size)

Read bytes from I2C internal ram. This can be only used when `access_ram_en` in configuration structure set to true.

参数

- **i2c_slave** -- **[in]** I2C slave device handle that created by `i2c_new_slave_device`.
- **ram_address** -- **[in]** The offset of RAM (Cannot larger than I2C RAM memory)
- **data** -- **[out]** Buffer to store data read from I2C ram.
- **receive_size** -- **[in]** Received size from RAM.

返回

- ESP_OK: I2C slave transmit success.
- ESP_ERR_INVALID_ARG: I2C slave transmit parameter invalid.
- ESP_ERR_NOT_SUPPORTED: This function should be work in non-fifo mode, but I2C_SLAVE_FIFO mode is configured

esp_err_t **i2c_slave_write_ram**(*i2c_slave_dev_handle_t* i2c_slave, uint8_t ram_address, const uint8_t *data, size_t size)

Write bytes to I2C internal ram. This can be only used when `access_ram_en` in configuration structure set to true.

参数

- **`i2c_slave`** -- **[in]** I2C slave device handle that created by `i2c_new_slave_device`.
- **`ram_address`** -- **[in]** The offset of RAM (Cannot larger than I2C RAM memory)
- **`data`** -- **[in]** Buffer to fill.
- **`size`** -- **[in]** Received size from RAM.

返回

- ESP_OK: I2C slave transmit success.
- ESP_ERR_INVALID_ARG: I2C slave transmit parameter invalid.
- ESP_ERR_INVALID_SIZE: Write size is larger than
- ESP_ERR_NOT_SUPPORTED: This function should be work in non-fifo mode, but I2C_SLAVE_FIFO mode is configured

Structures

struct **`i2c_slave_config_t`**

I2C slave specific configurations.

Public Members

i2c_port_num_t **`i2c_port`**

I2C port number, `-1` for auto selecting

gpio_num_t **`sda_io_num`**

SDA IO number used by I2C bus

gpio_num_t **`scl_io_num`**

SCL IO number used by I2C bus

i2c_clock_source_t **`clk_source`**

Clock source of I2C bus.

uint32_t **`send_buf_depth`**

Depth of internal transfer ringbuffer, increase this value can support more transfers pending in the background

uint16_t **`slave_addr`**

I2C slave address

i2c_addr_bit_len_t **`addr_bit_len`**

I2C slave address in bit length

int **`intr_priority`**

I2C interrupt priority, if set to 0, driver will select the default priority (1,2,3).

uint32_t **`broadcast_en`**

I2C slave enable broadcast

uint32_t **access_ram_en**

Can get access to I2C RAM directly

uint32_t **slave_unmatch_en**

Can trigger unmatch interrupt when slave address does not match what master sends

struct *i2c_slave_config_t*::[anonymous] **flags**

I2C slave config flags

struct **i2c_slave_event_callbacks_t**

Group of I2C slave callbacks (e.g. get i2c slave stretch cause). But take care of potential concurrency issues.

备注: The callbacks are all running under ISR context

备注: When CONFIG_I2C_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

Public Members

i2c_slave_received_callback_t **on_recv_done**

I2C slave receive done callback

Header File

- components/esp_driver_i2c/include/driver/i2c_types.h
- This header file can be included with:

```
#include "driver/i2c_types.h"
```

- This header file is a part of the API provided by the `esp_driver_i2c` component. To declare that your component depends on `esp_driver_i2c`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_i2c
```

or

```
PRIV_REQUIRES esp_driver_i2c
```

Structures

struct **i2c_master_event_data_t**

Data type used in I2C event callback.

Public Members

i2c_master_event_t **event**

The I2C hardware event that I2C callback is called.

struct **i2c_slave_rx_done_event_data_t**

Event structure used in I2C slave.

Public Members

uint8_t ***buffer***

Pointer for buffer received in callback.

Type Definitions

typedef int **i2c_port_num_t**

I2C port number.

typedef struct i2c_master_bus_t ***i2c_master_bus_handle_t**

Type of I2C master bus handle.

typedef struct i2c_master_dev_t ***i2c_master_dev_handle_t**

Type of I2C master bus device handle.

typedef struct i2c_slave_dev_t ***i2c_slave_dev_handle_t**

Type of I2C slave device handle.

typedef bool (***i2c_master_callback_t**)(*i2c_master_dev_handle_t* i2c_dev, const *i2c_master_event_data_t* *evt_data, void *arg)

An callback for I2C transaction.

Param i2c_dev [in] Handle for I2C device.
Param evt_data [out] I2C capture event data, fed by driver
Param arg [in] User data, set in `i2c_master_register_event_callbacks()`
Return Whether a high priority task has been waken up by this function

typedef bool (***i2c_slave_received_callback_t**)(*i2c_slave_dev_handle_t* i2c_slave, const *i2c_slave_rx_done_event_data_t* *evt_data, void *arg)

Callback signature for I2C slave.

Param i2c_slave [in] Handle for I2C slave.
Param evt_data [out] I2C capture event data, fed by driver
Param arg [in] User data, set in `i2c_slave_register_event_callbacks()`
Return Whether a high priority task has been waken up by this function

Enumerations

enum **i2c_master_status_t**

Enumeration for I2C fsm status.

Values:

enumerator **I2C_STATUS_READ**

read status for current master command

enumerator **I2C_STATUS_WRITE**

write status for current master command

enumerator **I2C_STATUS_START**

Start status for current master command

enumerator **I2C_STATUS_STOP**

stop status for current master command

enumerator **I2C_STATUS_IDLE**

idle status for current master command

enumerator **I2C_STATUS_ACK_ERROR**

ack error status for current master command

enumerator **I2C_STATUS_DONE**

I2C command done

enumerator **I2C_STATUS_TIMEOUT**

I2C bus status error, and operation timeout

enum **i2c_master_event_t**

Enumeration for I2C event.

Values:

enumerator **I2C_EVENT_ALIVE**

i2c bus in alive status.

enumerator **I2C_EVENT_DONE**

i2c bus transaction done

enumerator **I2C_EVENT_NACK**

i2c bus nack

enumerator **I2C_EVENT_TIMEOUT**

i2c bus timeout

Header File

- components/hal/include/hal/i2c_types.h
- This header file can be included with:

```
#include "hal/i2c_types.h"
```

Structures

struct **i2c_hal_clk_config_t**

Data structure for calculating I2C bus timing.

Public Members

uint16_t **clkm_div**

I2C core clock divider

uint16_t **scl_low**

I2C scl low period

uint16_t **scl_high**

I2C scl high period

uint16_t **scl_wait_high**

I2C scl wait_high period

uint16_t **sda_hold**

I2C scl low period

uint16_t **sda_sample**

I2C sda sample time

uint16_t **setup**

I2C start and stop condition setup period

uint16_t **hold**

I2C start and stop condition hold period

uint16_t **tout**

I2C bus timeout period

Type Definitions

typedef *soc_periph_i2c_clk_src_t* **i2c_clock_source_t**

I2C group clock source.

typedef *soc_periph_lp_i2c_clk_src_t* **lp_i2c_clock_source_t**

LP_UART source clock.

Enumerations

enum **i2c_port_t**

I2C port number, can be I2C_NUM_0 ~ (I2C_NUM_MAX-1).

Values:

enumerator **I2C_NUM_0**

I2C port 0

enumerator **I2C_NUM_1**

I2C port 1

enumerator **LP_I2C_NUM_0**

enumerator **I2C_NUM_MAX**

I2C port max

enum **i2c_addr_bit_len_t**

Enumeration for I2C device address bit length.

Values:

enumerator **I2C_ADDR_BIT_LEN_7**

i2c address bit length 7

enumerator **I2C_ADDR_BIT_LEN_10**

i2c address bit length 10

enum **i2c_mode_t**

Values:

enumerator **I2C_MODE_SLAVE**

I2C slave mode

enumerator **I2C_MODE_MASTER**

I2C master mode

enumerator **I2C_MODE_MAX**

enum **i2c_rw_t**

Values:

enumerator **I2C_MASTER_WRITE**

I2C write data

enumerator **I2C_MASTER_READ**

I2C read data

enum **i2c_trans_mode_t**

Values:

enumerator **I2C_DATA_MODE_MSB_FIRST**

I2C data msb first

enumerator **I2C_DATA_MODE_LSB_FIRST**

I2C data lsb first

enumerator **I2C_DATA_MODE_MAX**

enum **i2c_addr_mode_t**

Values:

enumerator **I2C_ADDR_BIT_7**

I2C 7bit address for slave mode

enumerator **I2C_ADDR_BIT_10**

I2C 10bit address for slave mode

enumerator **I2C_ADDR_BIT_MAX**

enum **i2c_ack_type_t**

Values:

enumerator **I2C_MASTER_ACK**

I2C ack for each byte read

enumerator **I2C_MASTER_NACK**

I2C nack for each byte read

enumerator **I2C_MASTER_LAST_NACK**

I2C nack for the last byte

enumerator **I2C_MASTER_ACK_MAX**

enum **i2c_slave_stretch_cause_t**

Enum for I2C slave stretch causes.

Values:

enumerator **I2C_SLAVE_STRETCH_CAUSE_ADDRESS_MATCH**

Stretching SCL low when the slave is read by the master and the address just matched

enumerator **I2C_SLAVE_STRETCH_CAUSE_TX_EMPTY**

Stretching SCL low when TX FIFO is empty in slave mode

enumerator **I2C_SLAVE_STRETCH_CAUSE_RX_FULL**

Stretching SCL low when RX FIFO is full in slave mode

enumerator **I2C_SLAVE_STRETCH_CAUSE_SENDING_ACK**

Stretching SCL low when slave sending ACK

2.5.13 I2S

简介

I2S（Inter-IC Sound，集成电路内置音频总线）是一种同步串行通信协议，通常用于在两个数字音频设备之间传输音频数据。

ESP32-P4 包含 1 个 I2S 外设。通过配置这些外设，可以借助 I2S 驱动来输入和输出采样数据。

标准或 TDM 通信模式下的 I2S 总线包含以下几条线路：

- **MCLK**：主时钟线。该信号线可选，具体取决于从机，主要用于向 I2S 从机提供参考时钟。
- **BCLK**：位时钟线。用于数据线的位时钟。
- **WS**：字（声道）选择线。通常用于识别声道（除 PDM 模式外）。

- **DIN/DOUT**：串行数据输入/输出线。如果 DIN 和 DOUT 被配置到相同的 GPIO，数据将在内部回环。

PDM 通信模式下的 I2S 总线包含以下几条线路：

- **CLK**：PDM 时钟线。
- **DIN/DOUT**：串行数据输入/输出线。

每个 I2S 控制器都具备以下功能，可由 I2S 驱动进行配置：

- 可用作系统主机或从机
- 可用作发射器或接收器
- DMA 控制器支持流数据采样，CPU 无需单独复制每个采样数据

每个控制器都有独立的 RX 和 TX 通道，连接到不同 GPIO 管脚，能够在不同的时钟和声道配置下工作。注意，尽管在一个控制器上 TX 通道和 RX 通道的内部 MCLK 相互独立，但输出的 MCLK 信号只能连接到一个通道。如果需要两个互相独立的 MCLK 输出，必须将其分配到不同的 I2S 控制器上。

I2S 文件结构

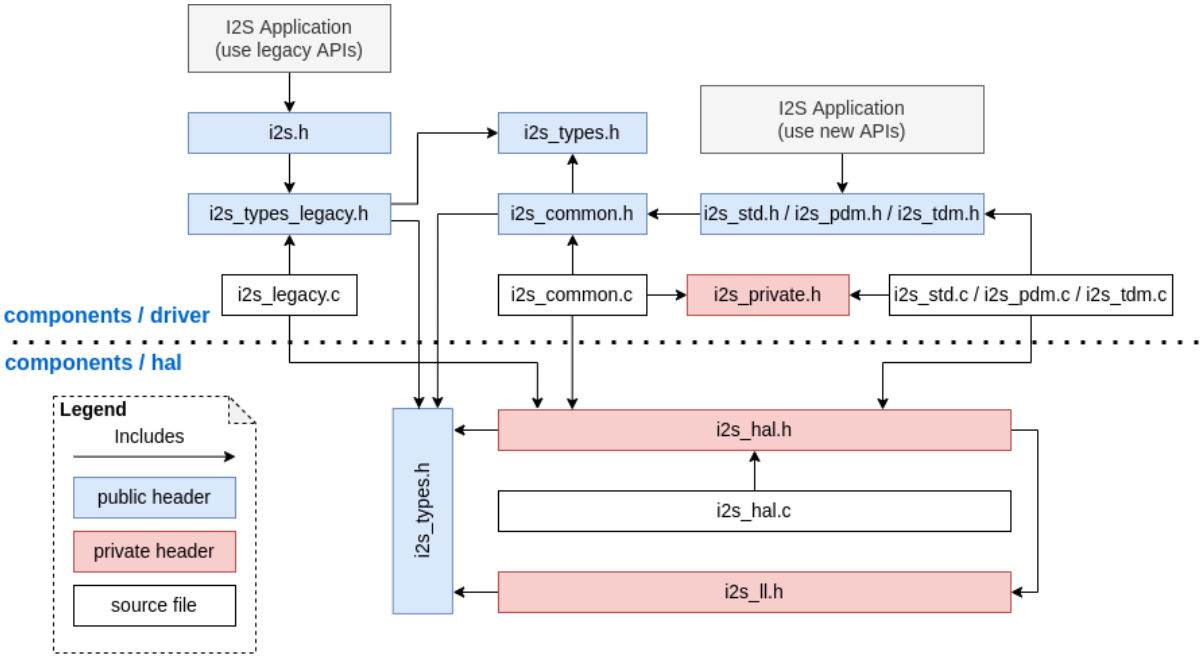


图 13: I2S 文件结构

需要包含在 I2S 应用中的公共头文件如下所示：

- `i2s.h`：提供原有 I2S API（用于使用原有驱动的应用）。
- `i2s_std.h`：提供标准通信模式的 API（用于使用标准模式的新驱动程序的应用）。
- `i2s_pdm.h`：提供 PDM 通信模式的 API（用于使用 PDM 模式的新驱动程序的应用）。
- `i2s_tdm.h`：提供 TDM 通信模式的 API（用于使用 TDM 模式的新驱动的应用）。

备注： 原有驱动与新驱动无法共存。包含 `i2s.h` 以使用原有驱动，或包含其他三个头文件以使用新驱动。原有驱动未来可能会被删除。

已包含在上述头文件中的公共头文件如下所示：

- `i2s_types_legacy.h`：提供只在原有驱动中使用的原有公共类型。
- `i2s_types.h`：提供公共类型。
- `i2s_common.h`：提供所有通信模式通用的 API。

I2S 时钟

时钟源

- `i2s_clock_src_t::I2S_CLK_SRC_DEFAULT`：默认 PLL 时钟。
- `i2s_clock_src_t::I2S_CLK_SRC_PLL_160M`：160 MHz PLL 时钟。
- `i2s_clock_src_t::I2S_CLK_SRC_APLL`：音频 PLL 时钟，在高采样率应用中比 `I2S_CLK_SRC_PLL_160M` 更精确。其频率可根据采样率进行配置，但如果 APLL 已经被 EMAC 或其他通道占用，则无法更改 APLL 频率，驱动程序将尝试在原有 APLL 频率下工作。如果原有 APLL 频率无法满足 I2S 的需求，时钟配置将失败。

时钟术语

- **采样率**：单声道每秒采样数据数量。
- **SCLK**：源时钟频率，即时钟源的频率。
- **MCLK**：主时钟频率，BCLK 由其产生。MCLK 信号通常作为参考时钟，用于同步 I2S 主机和从机之间的 BCLK 和 WS。
- **BCLK**：位时钟频率，一个 BCLK 时钟周期代表数据管脚上的一个数据位。通过 *i2s_std_slot_config_t::slot_bit_width* 配置的通道位宽即为一个声道中的 BCLK 时钟周期数量，因此一个声道中可以有 8/16/24/32 个 BCLK 时钟周期。
- **LRCK / WS**：左/右时钟或字选择时钟。在非 PDM 模式下，其频率等于采样率。

备注： 通常，MCLK 应该同时是采样率和 BCLK 的倍数。字段 *i2s_std_clk_config_t::mclk_multiple* 表示 MCLK 相对于采样率的倍数。在大多数情况下，将其设置为 `I2S_MCLK_MULTIPLE_256` 即可。但如果 `slot_bit_width` 被设置为 `I2S_SLOT_BIT_WIDTH_24BIT`，为了保证 MCLK 是 BCLK 的整数倍，应该将 *i2s_std_clk_config_t::mclk_multiple* 设置为能被 3 整除的倍数，如 `I2S_MCLK_MULTIPLE_384`，否则 WS 会不精准。

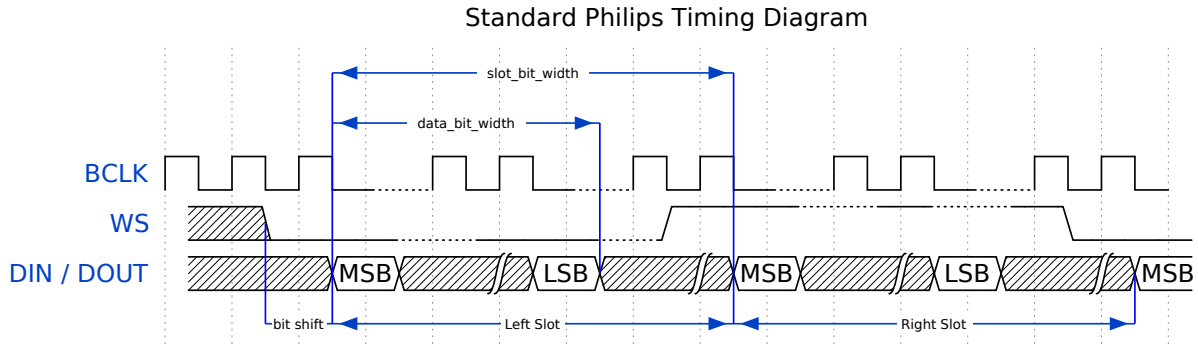
I2S 通信模式

模式概览

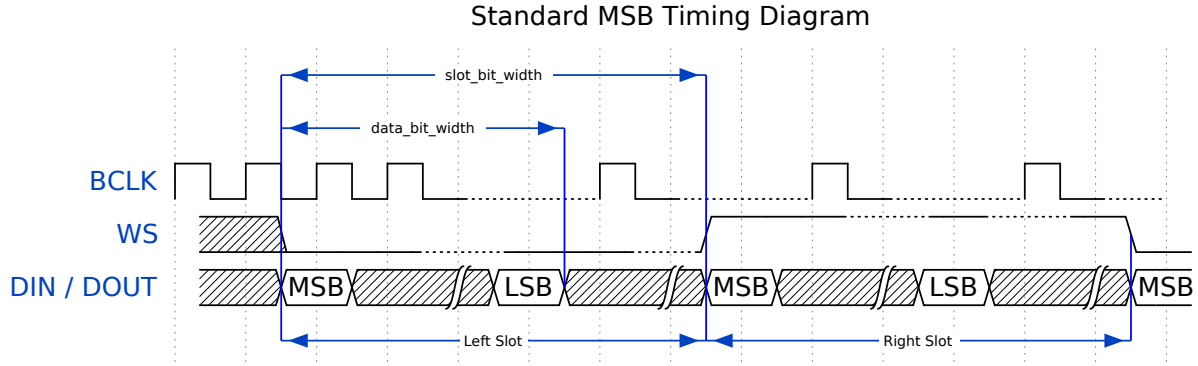
芯片	I2S 标准	PDM TX	PDM RX	TDM	ADC/DAC	LCD/摄像头
ESP32	I2S 0/1	I2S 0	I2S 0	无	I2S 0	I2S 0
ESP32-S2	I2S 0	无	无	无	无	I2S 0
ESP32-C3	I2S 0	I2S 0	无	I2S 0	无	无
ESP32-C6	I2S 0	I2S 0	无	I2S 0	无	无
ESP32-S3	I2S 0/1	I2S 0	I2S 0	I2S 0/1	无	无
ESP32-H2	I2S 0	I2S 0	无	I2S 0	无	无
ESP32-P4	I2S 0~2	I2S 0	I2S 0	I2S 0~2	无	无

标准模式 标准模式中有且仅有左右两个声道，驱动中将声道称为 slot。这些声道可以支持 8/16/24/32 位宽的采样数据，声道的通信格式主要包括以下几种：

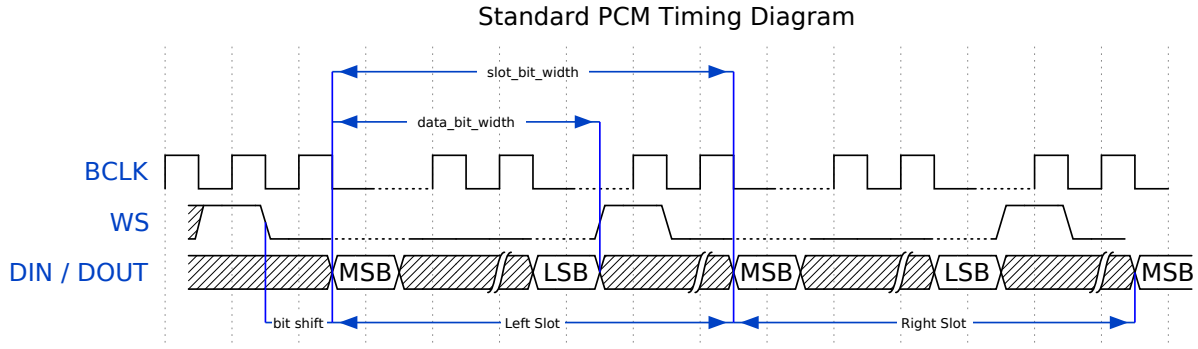
- **Philips 格式**：数据信号与 WS 信号相比有一个位的位移。WS 信号的占空比为 50%。



- **MSB 格式**：与 Philips 格式基本相同，但其数据没有位移。

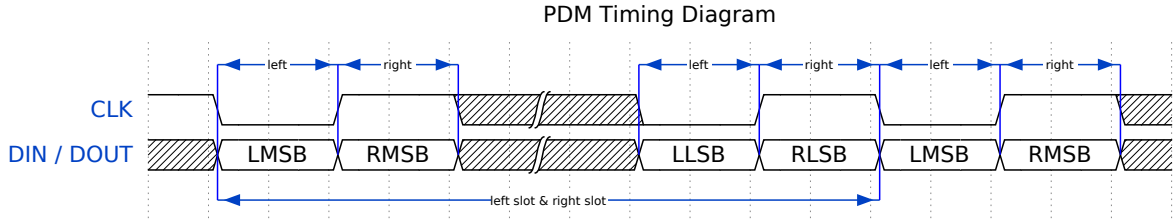


- **PCM 帧同步**：数据有一个位的位移，同时 WS 信号变成脉冲，持续一个 BCLK 周期。



PDM 模式 (TX) 在 PDM（Pulse-density Modulation，脉冲密度调制）模式下，TX 通道可以将 PCM 数据转换为 PDM 格式，该格式始终有左右两个声道。PDM TX 只在 I2S0 中受支持，且只支持 16 位宽的采样数据。PDM TX 至少需要一个 CLK 管脚用于时钟信号，一个 DOUT 管脚用于数据信号（即下图中的 WS 和 SD 信号。BCK 信号为内部位采样时钟，在 PDM 设备之间不需要）。PDM 模式允许用户配置上采样参数*i2s_pdm_tx_clk_config_t::up_sample_fp* 和*i2s_pdm_tx_clk_config_t::up_sample_fs*，上采样率可以通过公式 `up_sample_rate = i2s_pdm_tx_clk_config_t::up_sample_fp / i2s_pdm_tx_clk_config_t::up_sample_fs` 来计算。在 PDM TX 中有以下两种上采样模式：

- **固定时钟频率模式**：在这种模式下，上采样率将根据采样率的变化而变化。设置 `fp = 960`、`fs = sample_rate / 100`，则 CLK 管脚上的时钟频率 (Fpdm) 将固定为 `128 * 48 KHz = 6.144 MHz`。注意此频率不等于采样率 (Fpcm)。
- **固定上采样率模式**：在这种模式下，上采样率固定为 2。设置 `fp = 960`、`fs = 480`，则 CLK 管脚上的时钟频率 (Fpdm) 将为 `128 * sample_rate`。



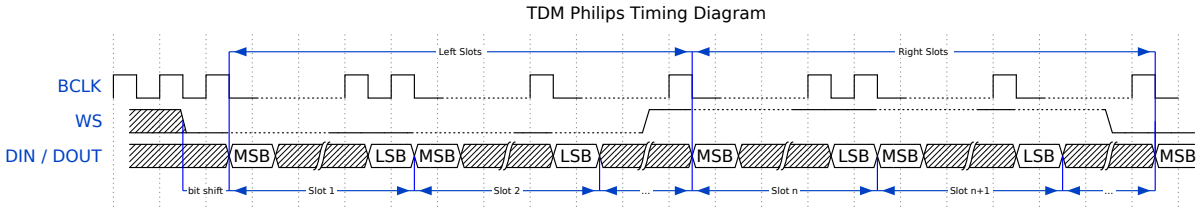
PDM 模式 (RX) 在 PDM（Pulse-density Modulation，脉冲密度调制）模式下，RX 通道可以接收 PDM 格式的数据并将数据转换成 PCM 格式。PDM RX 只在 I2S0 中受支持，且只支持 16 位宽的采样数据。PDM RX 至少需要一个 CLK 管脚用于时钟信号，一个 DIN 管脚用于数据信号。此模式允许用户配置下采样参数 *i2s_pdm_rx_clk_config_t::dn_sample_mode*。在 PDM RX 中有以下两种下采样模式：

- *i2s_pdm_dsr_t::I2S_PDM_DSR_8S*：在这种模式下，WS 管脚的时钟频率 (Fpdm) 将为 `sample_rate (Fpcm) * 64`。
- *i2s_pdm_dsr_t::I2S_PDM_DSR_16S*：在这种模式下，WS 管脚的时钟频率 (Fpdm) 将为 `sample_rate (Fpcm) * 128`。

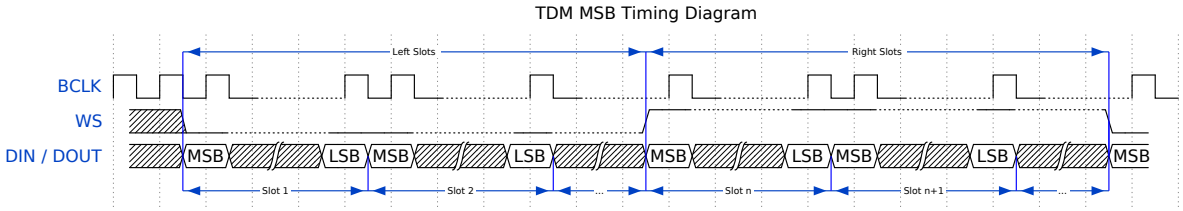
TDM 模式 TDM（Time Division Multiplexing，时分多路复用）模式最多支持 16 个声道，可通过 *i2s_tdm_slot_config_t::slot_mask* 启用通道。

该模式下无论启用多少声道，都支持任意数据位宽，也即一个帧中最多可以有 `32 位宽 * 16 个声道 = 512 位`的数据。

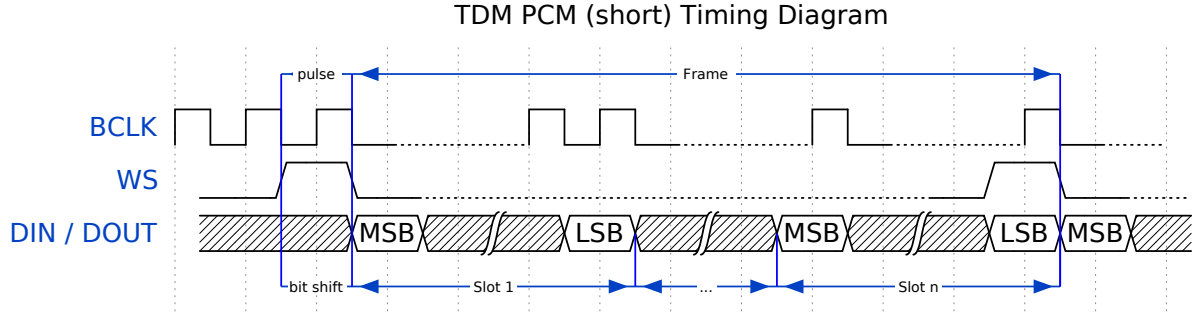
- **Philips 格式**：数据信号与 WS 信号相比有一个位的位移。无论一帧中包含多少个声道，WS 信号的占空比将始终保持为 50%。



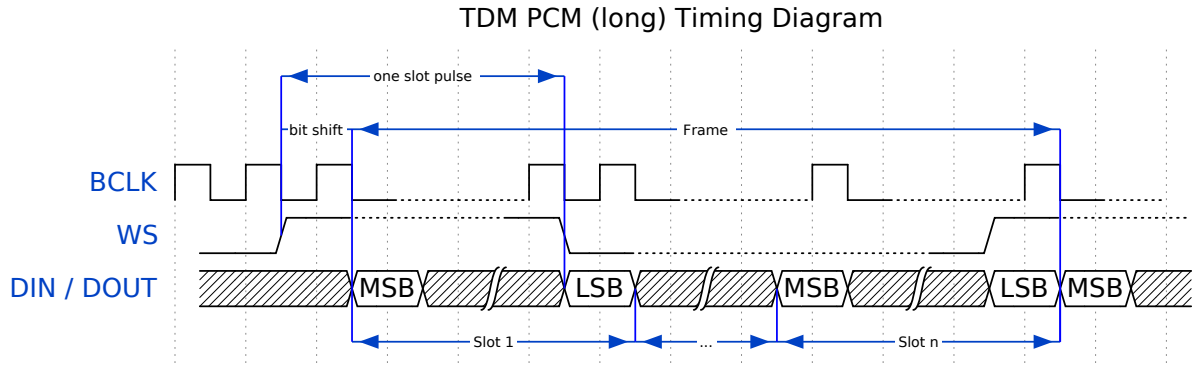
- **MSB 格式**：与 Philips 格式基本相同，但数据没有位移。



- **PCM 短帧同步**：数据有一个位的位移，同时 WS 信号变为脉冲，每帧持续一个 BCLK 周期。



- **PCM 长帧同步**：数据有一个位的位移，同时 WS 信号将在每一帧持续一个声道的宽度。例如，如果启用了四个声道，那么 WS 的占空比将是 25%，如果启用了五个声道，则为 20%。



功能概览

I2S 驱动提供以下服务：

资源管理 I2S 驱动中的资源可分为三个级别：

- 平台级资源：当前芯片中所有 I2S 控制器的资源。
- 控制器级资源：一个 I2S 控制器的资源。
- 通道级资源：一个 I2S 控制器 TX 或 RX 通道的资源。

公开的 API 都是通道级别的 API，通道句柄 *i2s_chan_handle_t* 可以帮助用户管理特定通道下的资源，而无需考虑其他两个级别的资源。高级别资源为私有资源，由驱动自动管理。用户可以调用 *i2s_new_channel()* 来分配通道句柄，或调用 *i2s_del_channel()* 来删除该句柄。

电源管理 电源管理启用（即开启 *CONFIG_PM_ENABLE*）时，系统将在进入 Light-sleep 前调整或停止 I2S 时钟源，这可能会影响 I2S 信号，从而导致传输或接收的数据无效。

I2S 驱动可以获取电源管理锁，从而防止系统设置更改或时钟源被禁用。时钟源为 APB 时，锁的类型将被设置为 *esp_pm_lock_type_t::ESP_PM_APB_FREQ_MAX*。时钟源为 APLL（若支持）时，锁的类型将被设置为 *esp_pm_lock_type_t::ESP_PM_NO_LIGHT_SLEEP*。用户通过 I2S 读写时（即调用 *i2s_channel_read()* 或 *i2s_channel_write()*），驱动程序将获取电源管理锁，并在读写完成后释放锁。

有限状态机 I2S 通道有三种状态，分别为 `registered`（已注册）、`ready`（准备就绪）和 `running`（运行中），它们的关系如下图所示：

图中的 <mode> 可用相应的 I2S 通信模式来代替，如 `std` 代表标准的双声道模式。更多关于通信模式的信息，请参考 *I2S 通信模式* 小节。

数据传输 I2S 的数据传输（包括数据发送和接收）由 DMA 实现。在传输数据之前，请调用 *i2s_channel_enable()* 来启用特定的通道。发送或接收的数据达到 DMA 缓冲区的大小时，将触发 `I2S_OUT_EOF` 或 `I2S_IN_SUC_EOF` 中断。注意，DMA 缓冲区的大小不等于 *i2s_chan_config_t::dma_frame_num*，这里的一帧是指一个 WS 周期内的所有采样数据。因此，`dma_buffer_size = dma_frame_num * slot_num * slot_bit_width / 8`。传输数据时，可以调用 *i2s_channel_write()* 来输入数据，并把数据从源缓冲区复制到 DMA TX 缓冲区等待传输完成。此过程将重复进行，直到发送的字节数达到配置的大小。接收数据时，用户可以调用函数 *i2s_channel_read()* 来等待接收包含 DMA 缓冲区地址的消息队列，从而将数据从 DMA RX 缓冲区复制到目标缓冲区。

i2s_channel_write() 和 *i2s_channel_read()* 都是阻塞函数，在源缓冲区的数据发送完毕前，或是整个目标缓冲区都被加载数据占用时，它们会一直保持等待状态。在等待时间

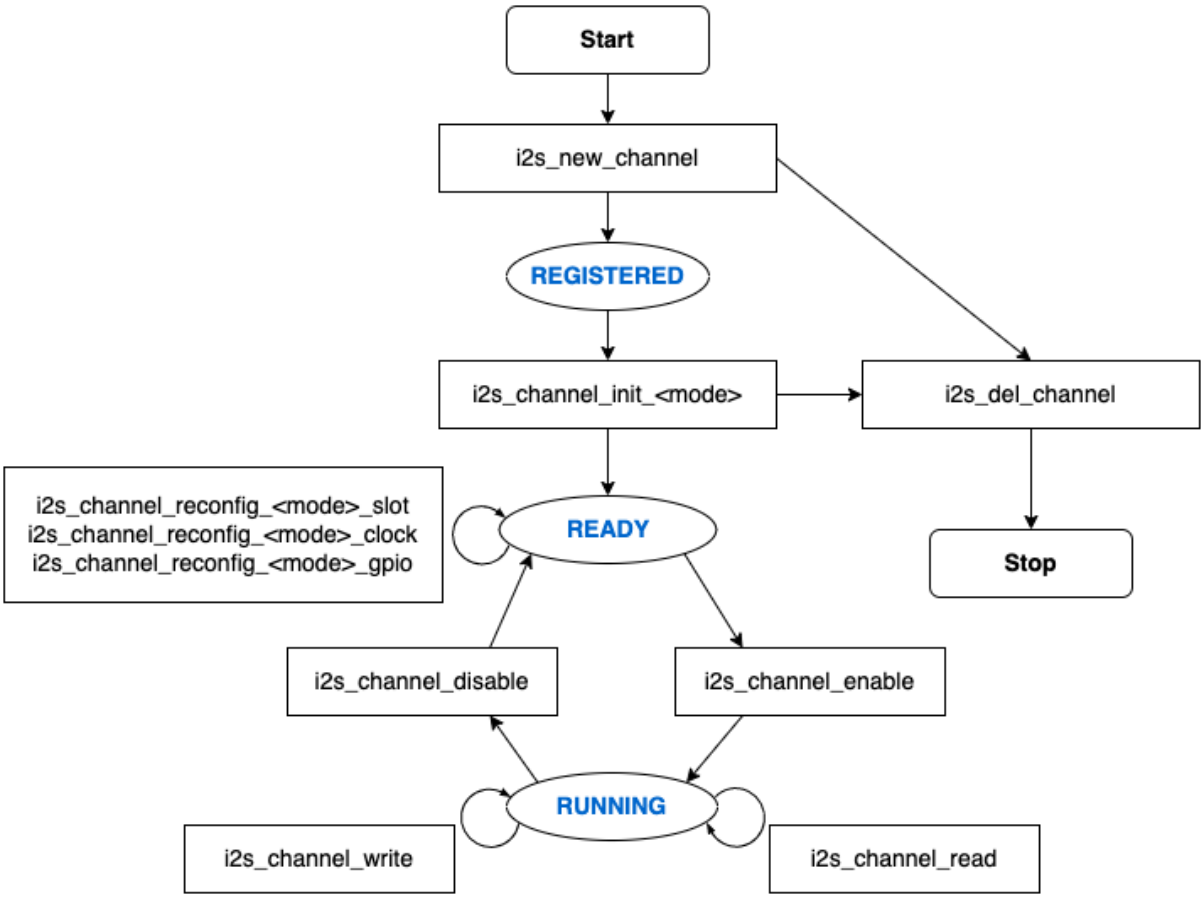


图 14: I2S 有限状态机

达到最大阻塞时间时，返回 ESP_ERR_TIMEOUT 错误。要实现异步发送或接收数据，可以通过*i2s_channel_register_event_callback()* 注册回调，随即便可在回调函数中直接访问 DMA 缓冲区，无需通过这两个阻塞函数来发送或接收数据。但请注意，该回调是一个中断回调，不要在该回调中添加复杂的逻辑、进行浮点运算或调用不可重入函数。

配置 用户可以通过调用相应函数（即 i2s_channel_init_std_mode()、i2s_channel_init_pdm_rx_mode()、i2s_channel_init_pdm_tx_mode() 或 i2s_channel_init_tdm_mode()）将通道初始化为特定模式。如果初始化后需要更新配置，必须先调用*i2s_channel_disable()* 以确保通道已经停止运行，然后再调用相应的'reconfig'函数，例如*i2s_channel_reconfig_std_slot()*、*i2s_channel_reconfig_std_clock()* 和*i2s_channel_reconfig_std_gpio()*。

IRAM 安全 默认情况下，由于写入或擦除 flash 等原因导致 cache 被禁用时，I2S 中断将产生延迟，无法及时执行 EOF 中断。

在实时应用中，可通过启用 Kconfig 选项*CONFIG_I2S_ISR_IRAM_SAFE* 来避免此种情况发生，启用后：

1. 即使在 cache 被禁用的情况下，中断仍可继续运行。
2. 驱动程序将存放进 DRAM 中（以防其意外映射到 PSRAM 中）。

启用该选项可以保证 cache 禁用时的中断运行，但会相应增加 IRAM 占用。

线程安全 驱动程序可保证所有公开的 I2S API 的线程安全，使用时，可以直接从不同的 RTOS 任务中调用此类 API，无需额外锁保护。注意，I2S 驱动使用 mutex 锁来保证线程安全，因此不允许在 ISR 中使用这些 API。

Kconfig 选项

- *CONFIG_I2S_ISR_IRAM_SAFE* 控制默认 ISR 处理程序能否在禁用 cache 的情况下工作。更多信息可参考 *IRAM 安全*。
- *CONFIG_I2S_SUPPRESS_DEPRECATE_WARN* 控制是否在使用原有 I2S 驱动时关闭警告信息。
- *CONFIG_I2S_ENABLE_DEBUG_LOG* 用于启用调试日志输出。启用该选项将增加固件的二进制文件大小。

应用实例

I2S 驱动例程请参考 peripherals/i2s 目录。以下为每种模式的简单用法：

标准 TX/RX 模式的应用 不同声道的通信格式可通过以下标准模式的辅助宏来生成。如上所述，在标准模式下有三种格式，辅助宏分别为：

- *I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG*
- *I2S_STD_PCM_SLOT_DEFAULT_CONFIG*
- *I2S_STD_MSB_SLOT_DEFAULT_CONFIG*

时钟配置的辅助宏为：

- *I2S_STD_CLK_DEFAULT_CONFIG*。

请参考标准模式 了解 STD API 的相关信息。更多细节请参考 esp_driver_i2s/include/driver/i2s_std.h。

STD TX 模式 以 16 位数据位宽为例，如果 uint16_t 写缓冲区中的数据如下所示：

数据 0	数据 1	数据 2	数据 3	数据 4	数据 5	数据 6	数据 7	...
0x0001	0x0002	0x0003	0x0004	0x0005	0x0006	0x0007	0x0008	...

下表展示了在不同*i2s_std_slot_config_t::slot_mode*和*i2s_std_slot_config_t::slot_mask*设置下线路上的真实数据。

数据位宽	声道模式	声道掩码	WS 低电平	WS 高电平	WS 低电平	WS 高电平	WS 低电平	WS 高电平	WS 低电平	WS 高电平
16 位	单声道	左	0x0001	0x0000	0x0002	0x0000	0x0003	0x0000	0x0004	0x0000
		右	0x0000	0x0001	0x0000	0x0002	0x0000	0x0003	0x0000	0x0004
		左右	0x0001	0x0001	0x0002	0x0002	0x0003	0x0003	0x0004	0x0004
	立体声	左	0x0001	0x0000	0x0003	0x0000	0x0005	0x0000	0x0007	0x0000
		右	0x0000	0x0002	0x0000	0x0004	0x0000	0x0006	0x0000	0x0008
		左右	0x0001	0x0002	0x0003	0x0004	0x0005	0x0006	0x0007	0x0008

备注: 数据位宽为 8 位和 32 位时，缓冲区的类型最好为 `uint8_t` 和 `uint32_t`。但需注意，数据位宽为 24 位时，数据缓冲区应该以 3 字节对齐，即每 3 个字节代表一个 24 位数据，另外，*i2s_chan_config_t::dma_frame_num*、*i2s_std_clk_config_t::mclk_multiple*和写缓冲区的大小应该为 3 的倍数，否则线路上的数据或采样率可能会不准确。

```
#include "driver/i2s_std.h"
#include "driver/gpio.h"

i2s_chan_handle_t tx_handle;
/* 通过辅助宏获取默认的通道配置
 * 这个辅助宏在 'i2s_common.h' 中定义，由所有 I2S 通信模式共享
 * 它可以帮助指定 I2S 角色和端口 ID */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE_
↪MASTER);
/* 分配新的 TX 通道并获取该通道的句柄 */
i2s_new_channel(&chan_cfg, &tx_handle, NULL);

/* 进行配置，可以通过宏生成声道配置和时钟配置
 * 这两个辅助宏在 'i2s_std.h' 中定义，只能用于 STD 模式
 * 它们可以帮助初始化或更新声道和时钟配置 */
i2s_std_config_t std_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(48000),
    .slot_cfg = I2S_STD_MSB_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_32BIT, I2S_SLOT_
↪MODE_STEREO),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .din = I2S_GPIO_UNUSED,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
/* 初始化通道 */
i2s_channel_init_std_mode(tx_handle, &std_cfg);

/* 在写入数据之前，先启用 TX 通道 */
i2s_channel_enable(tx_handle);
i2s_channel_write(tx_handle, src_buf, bytes_to_write, bytes_written, ticks_to_
↪wait);

/* 如果需要更新声道或时钟配置
```

(下页继续)

(续上页)

```
 * 需要在更新前先禁用通道 */
// i2s_channel_disable(tx_handle);
// std_cfg.slot_cfg.slot_mode = I2S_SLOT_MODE_MONO; // 默认为立体声
// i2s_channel_reconfig_std_slot(tx_handle, &std_cfg.slot_cfg);
// std_cfg.clk_cfg.sample_rate_hz = 96000;
// i2s_channel_reconfig_std_clock(tx_handle, &std_cfg.clk_cfg);

/* 删除通道之前必须先禁用通道 */
i2s_channel_disable(tx_handle);
/* 如果不再需要句柄，删除该句柄以释放通道资源 */
i2s_del_channel(tx_handle);
```

STD RX 模式 例如，当数据位宽为 16 时，如线路上的数据如下所示：

WS 低电平	WS 高电平	WS 低电平	WS 高电平	WS 低电平	WS 高电平	WS 低电平	WS 高电平	...
0x0001	0x0002	0x0003	0x0004	0x0005	0x0006	0x0007	0x0008	...

不同*i2s_std_slot_config_t::slot_mode* 和*i2s_std_slot_config_t::slot_mask* 配置下缓冲区中收到的数据如下所示。

数据位宽	声道模式	声道掩码	数据 0	数据 1	数据 2	数据 3	数据 4	数据 5	数据 6	数据 7
16 位	单声道	左	0x0001	0x0003	0x0005	0x0007	0x0009	0x000b	0x000d	0x000f
		右	0x0002	0x0004	0x0006	0x0008	0x000a	0x000c	0x000e	0x0010
	立体声	任意	0x0001	0x0002	0x0003	0x0004	0x0005	0x0006	0x0007	0x0008

备注: 8 位、24 位和 32 位与 16 位的情况类似，接收缓冲区的数据位宽与线路上的数据位宽相等。此外需注意，数据位宽为 24 位时，*i2s_chan_config_t::dma_frame_num*、*i2s_std_clk_config_t::mclk_multiple* 和接收缓冲区的大小应该为 3 的倍数，否则线路上的数据或采样率可能会不准确。

```
#include "driver/i2s_std.h"
#include "driver/gpio.h"

i2s_chan_handle_t rx_handle;
/* 通过辅助宏获取默认的通道配置
 * 这个辅助宏在 'i2s_common.h' 中定义，由所有 I2S 通信模式共享
 * 它可以帮助指定 I2S 角色和端口 ID */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE_
↪MASTER);
/* 分配新的 TX 通道并获取该通道的句柄 */
i2s_new_channel(&chan_cfg, NULL, &rx_handle);

/* 进行配置，可以通过宏生成声道配置和时钟配置
 * 这两个辅助宏在 'i2s_std.h' 中定义，只能用于 STD 模式
 * 它们可以帮助初始化或更新声道和时钟配置 */
i2s_std_config_t std_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(48000),
    .slot_cfg = I2S_STD_MSB_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_32BIT, I2S_SLOT_
↪MODE_STEREO),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
```

(下页继续)

(续上页)

```
        .dout = I2S_GPIO_UNUSED,
        .din = GPIO_NUM_19,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
/* 初始化通道 */
i2s_channel_init_std_mode(rx_handle, &std_cfg);

/* 在读取数据之前，先启动 RX 通道 */
i2s_channel_enable(rx_handle);
i2s_channel_read(rx_handle, desc_buf, bytes_to_read, bytes_read, ticks_to_wait);

/* 删除通道之前必须先禁用通道 */
i2s_channel_disable(rx_handle);
/* 如果不再需要句柄，删除该句柄以释放通道资源 */
i2s_del_channel(rx_handle);
```

PDM TX 模式的应用 针对 TX 通道的 PDM 模式，声道配置的辅助宏为：

- *I2S_PDM_TX_SLOT_DEFAULT_CONFIG*

时钟配置的辅助宏为：

- *I2S_PDM_TX_CLK_DEFAULT_CONFIG*

PDM TX API 的相关信息，可参考*PDM 模式*。更多细节请参阅 esp_driver_i2s/include/driver/i2s_pdm.h。

PDM 数据位宽固定为 16 位。如果 `int16_t` 写缓冲区中的数据如下：

数据 0	数据 1	数据 2	数据 3	数据 4	数据 5	数据 6	数据 7	...
0x0001	0x0002	0x0003	0x0004	0x0005	0x0006	0x0007	0x0008	...

下表展示了不同*i2s_pdm_tx_slot_config_t::slot_mode*和 `i2s_pdm_tx_slot_config_t::slot_mask` 设置下线路上的真实数据。为方便理解，已将线路上的数据格式由 PDM 转为 PCM。

线路模式	声道模式	线路	左	右	左	右	左	右	左	右
单线 Codec	单声道	dout	0x0001	0x0000	0x0002	0x0000	0x0003	0x0000	0x0004	0x0000
	立体声	dout	0x0001	0x0002	0x0003	0x0004	0x0005	0x0006	0x0007	0x0008
单线 DAC	单声道	dout	0x0001	0x0001	0x0002	0x0002	0x0003	0x0003	0x0004	0x0004
双线 DAC	单声道	dout	0x0002	0x0002	0x0004	0x0004	0x0006	0x0006	0x0008	0x0008
		dout2	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
	立体声	dout	0x0002	0x0002	0x0004	0x0004	0x0006	0x0006	0x0008	0x0008
		dout2	0x0001	0x0001	0x0003	0x0003	0x0005	0x0005	0x0007	0x0007

备注: PDM TX 模式有三种线路模式，分别为 `I2S_PDM_TX_ONE_LINE_CODEC`、`I2S_PDM_TX_ONE_LINE_DAC` 和 `I2S_PDM_TX_TWO_LINE_DAC`。单线 Codec 用于需要时钟信号的 PDM 编解码器，PDM 编解码器可以通过时钟电平来区分左右声道。另外两种模式可通过低通滤波器直接驱动功率放大器，而无需时钟信号，所以有两条线路来区分左右声道。此外，对于单线 Codec 的单声道模式，可以通过在 GPIO 配置中设置时钟反转标志，强制将声道改变为右声道。

```
#include "driver/i2s_pdm.h"
#include "driver/gpio.h"

/* 分配 I2S TX 通道 */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_0, I2S_ROLE_
↪MASTER);
i2s_new_channel(&chan_cfg, &tx_handle, NULL);

/* 初始化通道为 PDM TX 模式 */
i2s_pdm_tx_config_t pdm_tx_cfg = {
    .clk_cfg = I2S_PDM_TX_CLK_DEFAULT_CONFIG(36000),
    .slot_cfg = I2S_PDM_TX_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_
↪MODE_MONO),
    .gpio_cfg = {
        .clk = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .invert_flags = {
            .clk_inv = false,
        },
    },
};
i2s_channel_init_pdm_tx_mode(tx_handle, &pdm_tx_cfg);

...
```

PDM RX 模式的应用 针对 RX 通道的 PDM 模式，声道配置的辅助宏为：

- *I2S_PDM_RX_SLOT_DEFAULT_CONFIG*

时钟配置的辅助宏为：

- *I2S_PDM_RX_CLK_DEFAULT_CONFIG*

PDM RX API 的相关信息，可参考*PDM 模式*。更多细节请参阅 esp_driver_i2s/include/driver/i2s_pdm.h。

PDM 数据位宽固定为 16 位。如果线路上的数据如下所示。为方便理解，已将线路上的数据格式由 PDM 转为 PCM。

左	右	左	右	左	右	左	右	...
0x0001	0x0002	0x0003	0x0004	0x0005	0x0006	0x0007	0x0008	...

下表展示了不同*i2s_pdm_rx_slot_config_t::slot_mode*和*i2s_pdm_rx_slot_config_t::slot_mask*设置下'int16_t' 缓冲区接收的数据。

```
#include "driver/i2s_pdm.h"
#include "driver/gpio.h"

i2s_chan_handle_t rx_handle;

/* 分配 I2S RX 通道 */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_0, I2S_ROLE_
↪MASTER);
i2s_new_channel(&chan_cfg, NULL, &rx_handle);

/* 初始化通道为 PDM RX 模式 */
i2s_pdm_rx_config_t pdm_rx_cfg = {
    .clk_cfg = I2S_PDM_RX_CLK_DEFAULT_CONFIG(36000),
    .slot_cfg = I2S_PDM_RX_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_
↪MODE_MONO),
    .gpio_cfg = {
        .clk = GPIO_NUM_5,
```

(下页继续)

(续上页)

```
        .din = GPIO_NUM_19,
        .invert_flags = {
            .clk_inv = false,
        },
    },
};
i2s_channel_init_pdm_rx_mode(rx_handle, &pdm_rx_cfg);

...
```

TDM TX/RX 模式的应用 可以通过以下 TDM 模式的辅助宏生成不同的声道通信格式。如上所述，TDM 模式有四种格式，它们的辅助宏分别为：

- *I2S_TDM_PHILIPS_SLOT_DEFAULT_CONFIG*
- *I2S_TDM_MSB_SLOT_DEFAULT_CONFIG*
- *I2S_TDM_PCM_SHORT_SLOT_DEFAULT_CONFIG*
- *I2S_TDM_PCM_LONG_SLOT_DEFAULT_CONFIG*

时钟配置的辅助宏为：

- *I2S_TDM_CLK_DEFAULT_CONFIG*

有关 TDM API 的信息，请参阅*TDM 模式*。更多细节请参阅 esp_driver_i2s/include/driver/i2s_tdm.h。

备注： 在为从机配置时钟时，由于硬件限制，请注意*i2s_tdm_clk_config_t::bclk_div* 不应小于 8，增加此字段的值可以减少从机发送数据的延迟。使用高采样率时，数据可能会延迟一个 BCLK 周期以上，这将导致数据错位。可以通过缓慢增加*i2s_tdm_clk_config_t::bclk_div* 的值来进行校正。

由于*i2s_tdm_clk_config_t::bclk_div* 是 MCLK 基于 BCLK 的除数，增加该值也可以提高 MCLK 频率。因此，如果 MCLK 频率太高，将会无法从源时钟分频，此时时钟计算可能会失败，也就是说*i2s_tdm_clk_config_t::bclk_div* 不是越大越好。

TDM TX 模式

```
#include "driver/i2s_tdm.h"
#include "driver/gpio.h"

/* 分配 I2S TX 通道 */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE_
↪MASTER);
i2s_new_channel(&chan_cfg, &tx_handle, NULL);

/* 初始化通道为 TDM 模式 */
i2s_tdm_config_t tdm_cfg = {
    .clk_cfg = I2S_TDM_CLK_DEFAULT_CONFIG(44100),
    .slot_cfg = I2S_TDM_MSB_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_
↪MODE_STEREO,
                I2S_TDM_SLOT0 | I2S_TDM_SLOT1 | I2S_TDM_SLOT2 | I2S_TDM_SLOT3),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .din = I2S_GPIO_UNUSED,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
```

(下页继续)

(续上页)

```
    },
};
i2s_channel_init_tdm_mode(tx_handle, &tdm_cfg);

...
```

TDM RX 模式

```
#include "driver/i2s_tdm.h"
#include "driver/gpio.h"

/* 将通道模式设置为 TDM */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_CONFIG(I2S_ROLE_MASTER, I2S_COMM_MODE_TDM,
↪ &i2s_pin);
i2s_new_channel(&chan_cfg, NULL, &rx_handle);

/* 初始化通道为 TDM 模式 */
i2s_tdm_config_t tdm_cfg = {
    .clk_cfg = I2S_TDM_CLK_DEFAULT_CONFIG(44100),
    .slot_cfg = I2S_TDM_MSB_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_
↪MODE_STEREO,
                I2S_TDM_SLOT0 | I2S_TDM_SLOT1 | I2S_TDM_SLOT2 | I2S_TDM_SLOT3),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
        .dout = I2S_GPIO_UNUSED,
        .din = GPIO_NUM_18,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
i2s_channel_init_tdm_mode(rx_handle, &tdm_cfg);
...
```

全双工 全双工模式可以在 I2S 端口中同时注册 TX 和 RX 通道，同时通道共享 BCLK 和 WS 信号。目前，STD 和 TDM 通信模式支持以下方式的全双工通信，但不支持 PDM 全双工模式，因为 PDM 模式下 TX 和 RX 通道的时钟不同。

请注意，一个句柄只能代表一个通道，因此仍然需要对 TX 和 RX 通道逐个进行声道和时钟配置。

以下示例展示了如何分配两个全双工通道：

```
#include "driver/i2s_std.h"
#include "driver/gpio.h"

i2s_chan_handle_t tx_handle;
i2s_chan_handle_t rx_handle;

/* 分配两个 I2S 通道 */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE_
↪MASTER);
/* 同时分配给 TX 和 RX 通道，使其进入全双工模式。 */
i2s_new_channel(&chan_cfg, &tx_handle, &rx_handle);

/* 配置两个通道，因为在全双工模式下，TX 和 RX 通道必须相同。 */
```

(下页继续)

(续上页)

```
i2s_std_config_t std_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(32000),
    .slot_cfg = I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_
↪SLOT_MODE_STEREO),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .din = GPIO_NUM_19,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
i2s_channel_init_std_mode(tx_handle, &std_cfg);
i2s_channel_init_std_mode(rx_handle, &std_cfg);

i2s_channel_enable(tx_handle);
i2s_channel_enable(rx_handle);

...
```

单工模式 在单工模式下分配通道，应该为每个通道调用*i2s_new_channel()*。ESP32-P4 上，TX/RX 通道的时钟和 GPIO 管脚相互独立，因此可以配置为不同的模式和时钟，并且能够在单工模式下共存于同一个 I2S 端口中。对于 PDM 模式，用户可以通过在同一个 I2S 端口上注册 PDM TX 单工和 PDM RX 单工来实现 PDM 双工。但在这种情况下，PDM TX/RX 可能会使用不同的时钟，因此在配置 GPIO 管脚和时钟时需多加注意。

以下为单工模式的示例。请注意，如果 TX 和 RX 通道来自同一个控制器，则 TX 和 RX 通道的内部 MCLK 信号虽然是分开的，但输出的 MCLK 信号只能绑定到其中一个通道。如果两个通道都初始化了 MCLK，则该信号会绑定到后初始化的通道。

```
#include "driver/i2s_std.h"
#include "driver/gpio.h"

i2s_chan_handle_t tx_handle;
i2s_chan_handle_t rx_handle;
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_0, I2S_ROLE_
↪MASTER);
i2s_new_channel(&chan_cfg, &tx_handle, NULL);
i2s_std_config_t std_tx_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(48000),
    .slot_cfg = I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_
↪SLOT_MODE_STEREO),
    .gpio_cfg = {
        .mclk = GPIO_NUM_0,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .din = I2S_GPIO_UNUSED,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
```

(下页继续)

(续上页)

```
/* 初始化通道 */
i2s_channel_init_std_mode(tx_handle, &std_tx_cfg);
i2s_channel_enable(tx_handle);

/* 如果没有找到其他可用的 I2S 设备，RX 通道将被注册在另一个 I2S 上
 * 并返回 ESP_ERR_NOT_FOUND */
i2s_new_channel(&chan_cfg, NULL, &rx_handle); // RX 和 TX 通道都将注册在 I2S0␣
↪上，但配置可以不同
i2s_std_config_t std_rx_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(16000),
    .slot_cfg = I2S_STD_MSB_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_32BIT, I2S_SLOT_
↪MODE_STEREO),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_6,
        .ws = GPIO_NUM_7,
        .dout = I2S_GPIO_UNUSED,
        .din = GPIO_NUM_19,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
i2s_channel_init_std_mode(rx_handle, &std_rx_cfg);
i2s_channel_enable(rx_handle);
```

应用注意事项

防止数据丢失 对于需要高频采样率的应用，数据的巨大吞吐量可能会导致数据丢失。用户可以通过注册 ISR 回调函数来接收事件队列中的数据丢失事件：

```
static IRAM_ATTR bool i2s_rx_queue_overflow_callback(i2s_chan_handle_t␣
↪handle, i2s_event_data_t *event, void *user_ctx)
{
    // 处理 RX 队列溢出事件 ...
    return false;
}

i2s_event_callbacks_t cbs = {
    .on_recv = NULL,
    .on_recv_q_ovf = i2s_rx_queue_overflow_callback,
    .on_sent = NULL,
    .on_send_q_ovf = NULL,
};
TEST_ESP_OK(i2s_channel_register_event_callback(rx_handle, &cbs, NULL));
```

请按照以下步骤操作，以防止数据丢失：

1. 确定中断间隔。通常来说，当发生数据丢失时，为减少中断次数，中断间隔应该越久越好。因此，在保证 DMA 缓冲区大小不超过最大值 4092 的前提下，应使 `dma_frame_num` 尽可能大。具体转换关系如下：

```
interrupt_interval(unit: sec) = dma_frame_num / sample_rate
dma_buffer_size = dma_frame_num * slot_num * data_bit_width / 8 <= 4092
```

2. 确定 `dma_desc_num` 的值。`dma_desc_num` 由 `i2s_channel_read` 轮询周期的最大时间决定，所有接收到的数据都应该存储在两个 `i2s_channel_read` 之间。这个周期可以通过计时器或输出 GPIO 信号来计算。具体转换关系如下：

```
dma_desc_num > polling_cycle / interrupt_interval
```

3. 确定接收缓冲区大小。在 i2s_channel_read 中提供的接收缓冲区应当能够容纳所有 DMA 缓冲区中的数据，这意味着它应该大于所有 DMA 缓冲区的总大小:

```
recv_buffer_size > dma_desc_num * dma_buffer_size
```

例如，如果某个 I2S 应用的已知值包括:

```
sample_rate = 144000 Hz
data_bit_width = 32 bits
slot_num = 2
polling_cycle = 10 ms
```

那么可以按照以下公式计算出参数 dma_frame_num、dma_desc_num 和 recv_buf_size:

```
dma_frame_num * slot_num * data_bit_width / 8 = dma_buffer_size <= 4092
dma_frame_num <= 511
interrupt_interval = dma_frame_num / sample_rate = 511 / 144000 = 0.003549 s = 3.
↪549 ms
dma_desc_num > polling_cycle / interrupt_interval = cell(10 / 3.549) = cell(2.818)␣
↪= 3
recv_buffer_size > dma_desc_num * dma_buffer_size = 3 * 4092 = 12276 bytes
```

API 参考

标准模式

Header File

- components/esp_driver_i2s/include/driver/i2s_std.h
- This header file can be included with:

```
#include "driver/i2s_std.h"
```

- This header file is a part of the API provided by the esp_driver_i2s component. To declare that your component depends on esp_driver_i2s, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_i2s
```

or

```
PRIV_REQUIRES esp_driver_i2s
```

Functions

esp_err_t **i2s_channel_init_std_mode**(*i2s_chan_handle_t* handle, const *i2s_std_config_t* *std_cfg)

Initialize I2S channel to standard mode.

备注: Only allowed to be called when the channel state is REGISTERED, (i.e., channel has been allocated, but not initialized) and the state will be updated to READY if initialization success, otherwise the state will return to REGISTERED.

参数

- **handle** -- **[in]** I2S channel handler
- **std_cfg** -- **[in]** Configurations for standard mode, including clock, slot and GPIO The clock configuration can be generated by the helper macro

I2S_STD_CLK_DEFAULT_CONFIG The slot configuration can be generated by the helper macro I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG, I2S_STD_PCM_SLOT_DEFAULT_CONFIG or I2S_STD_MSB_SLOT_DEFAULT_CONFIG

返回

- ESP_OK Initialize successfully
- ESP_ERR_NO_MEM No memory for storing the channel information
- ESP_ERR_INVALID_ARG NULL pointer or invalid configuration
- ESP_ERR_INVALID_STATE This channel is not registered

esp_err_t **i2s_channel_reconfig_std_clock**(*i2s_chan_handle_t* handle, const *i2s_std_clk_config_t* *clk_cfg)

Reconfigure the I2S clock for standard mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. i2s_channel_disable should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to standard mode, i.e., i2s_channel_init_std_mode has been called before reconfiguring

参数

- **handle** -- **[in]** I2S channel handler
- **clk_cfg** -- **[in]** Standard mode clock configuration, can be generated by I2S_STD_CLK_DEFAULT_CONFIG

返回

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not standard mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **i2s_channel_reconfig_std_slot**(*i2s_chan_handle_t* handle, const *i2s_std_slot_config_t* *slot_cfg)

Reconfigure the I2S slot for standard mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. i2s_channel_disable should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to standard mode, i.e., i2s_channel_init_std_mode has been called before reconfiguring

参数

- **handle** -- **[in]** I2S channel handler
- **slot_cfg** -- **[in]** Standard mode slot configuration, can be generated by I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG, I2S_STD_PCM_SLOT_DEFAULT_CONFIG and I2S_STD_MSB_SLOT_DEFAULT_CONFIG.

返回

- ESP_OK Set clock successfully
- ESP_ERR_NO_MEM No memory for DMA buffer
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not standard mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **i2s_channel_reconfig_std_gpio**(*i2s_chan_handle_t* handle, const *i2s_std_gpio_config_t* *gpio_cfg)

Reconfigure the I2S GPIO for standard mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to standard mode, i.e., `i2s_channel_init_std_mode` has been called before reconfiguring

---

参数

- **handle** -- **[in]** I2S channel handler
- **gpio_cfg** -- **[in]** Standard mode GPIO configuration, specified by user

返回

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not standard mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

Structures

struct **i2s_std_slot_config_t**

I2S slot configuration for standard mode.

Public Members

i2s_data_bit_width_t **data_bit_width**

I2S sample data bit width (valid data bits per sample)

i2s_slot_bit_width_t **slot_bit_width**

I2S slot bit width (total bits per slot)

i2s_slot_mode_t **slot_mode**

Set mono or stereo mode with I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO In TX direction, mono means the written buffer contains only one slot data and stereo means the written buffer contains both left and right data

i2s_std_slot_mask_t **slot_mask**

Select the left, right or both slot

uint32_t **ws_width**

WS signal width (i.e. the number of BCLK ticks that WS signal is high)

bool **ws_pol**

WS signal polarity, set true to enable high lever first

bool **bit_shift**

Set to enable bit shift in Philips mode

bool **left_align**

Set to enable left alignment

bool **big_endian**

Set to enable big endian

bool **bit_order_lsb**

Set to enable lsb first

struct **i2s_std_clk_config_t**

I2S clock configuration for standard mode.

Public Members

uint32_t **sample_rate_hz**

I2S sample rate

i2s_clock_src_t **clk_src**

Choose clock source, see `soc_periph_i2s_clk_src_t` for the supported clock sources. selected `I2S_CLK_SRC_EXTERNAL` (if supports) to enable the external source clock input via MCLK pin,

uint32_t **ext_clk_freq_hz**

External clock source frequency in Hz, only take effect when `clk_src = I2S_CLK_SRC_EXTERNAL`, otherwise this field will be ignored, Please make sure the frequency input is equal or greater than BCLK, i.e. `sample_rate_hz * slot_bits * 2`

i2s_mclk_multiple_t **mclk_multiple**

The multiple of MCLK to the sample rate Default is 256 in the helper macro, it can satisfy most of cases, but please set this field a multiple of `3` (like 384) when using 24-bit data width, otherwise the sample rate might be inaccurate

struct **i2s_std_gpio_config_t**

I2S standard mode GPIO pins configuration.

Public Members

gpio_num_t **mclk**

MCK pin, output by default, input if the clock source is selected to `I2S_CLK_SRC_EXTERNAL`

gpio_num_t **bclk**

BCK pin, input in slave role, output in master role

gpio_num_t **ws**

WS pin, input in slave role, output in master role

gpio_num_t **dout**

DATA pin, output

gpio_num_t **din**

DATA pin, input

uint32_t **mclk_inv**

Set 1 to invert the MCLK input/output

uint32_t **bclk_inv**

Set 1 to invert the BCLK input/output

uint32_t **ws_inv**

Set 1 to invert the WS input/output

struct *i2s_std_gpio_config_t*::[anonymous] **invert_flags**

GPIO pin invert flags

struct **i2s_std_config_t**

I2S standard mode major configuration that including clock/slot/GPIO configuration.

Public Members

i2s_std_clk_config_t **clk_cfg**

Standard mode clock configuration, can be generated by macro I2S_STD_CLK_DEFAULT_CONFIG

i2s_std_slot_config_t **slot_cfg**

Standard mode slot configuration, can be generated by macros I2S_STD_[mode]_SLOT_DEFAULT_CONFIG, [mode] can be replaced with PHILIPS/MSB/PCM

i2s_std_gpio_config_t **gpio_cfg**

Standard mode GPIO configuration, specified by user

Macros

I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo)

Philips format in 2 slots.

This file is specified for I2S standard communication mode Features:

- Philips/MSB/PCM are supported in standard mode
- Fixed to 2 slots

参数

- **bits_per_sample** -- I2S data bit width
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

I2S_STD_PCM_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo)

PCM(short) format in 2 slots.

备注: PCM(long) is same as Philips in 2 slots

参数

- **bits_per_sample** -- I2S data bit width

- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

I2S_STD_MSB_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo)

MSB format in 2 slots.

参数

- **bits_per_sample** -- I2S data bit width
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

I2S_STD_CLK_DEFAULT_CONFIG(rate)

I2S default standard clock configuration.

备注： Please set the mclk_multiple to I2S_MCLK_MULTIPLE_384 while using 24 bits data width Otherwise the sample rate might be imprecise since the BCLK division is not a integer

---

参数

- **rate** -- sample rate

PDM 模式

Header File

- components/esp_driver_i2s/include/driver/i2s_pdm.h
- This header file can be included with:

```
#include "driver/i2s_pdm.h"
```

- This header file is a part of the API provided by the `esp_driver_i2s` component. To declare that your component depends on `esp_driver_i2s`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_i2s
```

or

```
PRIV_REQUIRES esp_driver_i2s
```

Functions

esp_err_t **i2s_channel_init_pdm_rx_mode**(*i2s_chan_handle_t* handle, const *i2s_pdm_rx_config_t* *pdm_rx_cfg)

Initialize I2S channel to PDM RX mode.

备注： Only allowed to be called when the channel state is REGISTERED, (i.e., channel has been allocated, but not initialized) and the state will be updated to READY if initialization success, otherwise the state will return to REGISTERED.

参数

- **handle** -- **[in]** I2S RX channel handler
- **pdm_rx_cfg** -- **[in]** Configurations for PDM RX mode, including clock, slot and GPIO The clock configuration can be generated by the helper macro `I2S_PDM_RX_CLK_DEFAULT_CONFIG` The slot configuration can be generated by the helper macro `I2S_PDM_RX_SLOT_DEFAULT_CONFIG`

返回

- ESP_OK Initialize successfully
- ESP_ERR_NO_MEM No memory for storing the channel information
- ESP_ERR_INVALID_ARG NULL pointer or invalid configuration

- ESP_ERR_INVALID_STATE This channel is not registered

esp_err_t **i2s_channel_reconfig_pdm_rx_clock**(*i2s_chan_handle_t* handle, const *i2s_pdm_rx_clk_config_t* *clk_cfg)

Reconfigure the I2S clock for PDM RX mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to PDM RX mode, i.e., `i2s_channel_init_pdm_rx_mode` has been called before reconfiguring

参数

- **handle** -- **[in]** I2S RX channel handler
- **clk_cfg** -- **[in]** PDM RX mode clock configuration, can be generated by `I2S_PDM_RX_CLK_DEFAULT_CONFIG`

返回

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **i2s_channel_reconfig_pdm_rx_slot**(*i2s_chan_handle_t* handle, const *i2s_pdm_rx_slot_config_t* *slot_cfg)

Reconfigure the I2S slot for PDM RX mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to PDM RX mode, i.e., `i2s_channel_init_pdm_rx_mode` has been called before reconfiguring

参数

- **handle** -- **[in]** I2S RX channel handler
- **slot_cfg** -- **[in]** PDM RX mode slot configuration, can be generated by `I2S_PDM_RX_SLOT_DEFAULT_CONFIG`

返回

- ESP_OK Set clock successfully
- ESP_ERR_NO_MEM No memory for DMA buffer
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **i2s_channel_reconfig_pdm_rx_gpio**(*i2s_chan_handle_t* handle, const *i2s_pdm_rx_gpio_config_t* *gpio_cfg)

Reconfigure the I2S GPIO for PDM RX mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to PDM RX mode, i.e., `i2s_channel_init_pdm_rx_mode` has been called before reconfiguring

参数

- **`handle`** -- **[in]** I2S RX channel handler
- **`gpio_cfg`** -- **[in]** PDM RX mode GPIO configuration, specified by user

返回

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **`i2s_channel_init_pdm_tx_mode`**(*i2s_chan_handle_t* handle, const *i2s_pdm_tx_config_t* *pdm_tx_cfg)

Initialize I2S channel to PDM TX mode.

备注: Only allowed to be called when the channel state is REGISTERED, (i.e., channel has been allocated, but not initialized) and the state will be updated to READY if initialization success, otherwise the state will return to REGISTERED.

参数

- **`handle`** -- **[in]** I2S TX channel handler
- **`pdm_tx_cfg`** -- **[in]** Configurations for PDM TX mode, including clock, slot and GPIO The clock configuration can be generated by the helper macro `I2S_PDM_TX_CLK_DEFAULT_CONFIG` The slot configuration can be generated by the helper macro `I2S_PDM_TX_SLOT_DEFAULT_CONFIG`

返回

- ESP_OK Initialize successfully
- ESP_ERR_NO_MEM No memory for storing the channel information
- ESP_ERR_INVALID_ARG NULL pointer or invalid configuration
- ESP_ERR_INVALID_STATE This channel is not registered

esp_err_t **`i2s_channel_reconfig_pdm_tx_clock`**(*i2s_chan_handle_t* handle, const *i2s_pdm_tx_clk_config_t* *clk_cfg)

Reconfigure the I2S clock for PDM TX mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to PDM TX mode, i.e., `i2s_channel_init_pdm_tx_mode` has been called before reconfiguring

参数

- **`handle`** -- **[in]** I2S TX channel handler
- **`clk_cfg`** -- **[in]** PDM TX mode clock configuration, can be generated by `I2S_PDM_TX_CLK_DEFAULT_CONFIG`

返回

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **i2s_channel_reconfig_pdm_tx_slot**(*i2s_chan_handle_t* handle, const *i2s_pdm_tx_slot_config_t* *slot_cfg)

Reconfigure the I2S slot for PDM TX mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to PDM TX mode, i.e., `i2s_channel_init_pdm_tx_mode` has been called before reconfiguring

参数

- **handle** -- **[in]** I2S TX channel handler
- **slot_cfg** -- **[in]** PDM TX mode slot configuration, can be generated by `I2S_PDM_TX_SLOT_DEFAULT_CONFIG`

返回

- ESP_OK Set clock successfully
- ESP_ERR_NO_MEM No memory for DMA buffer
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **i2s_channel_reconfig_pdm_tx_gpio**(*i2s_chan_handle_t* handle, const *i2s_pdm_tx_gpio_config_t* *gpio_cfg)

Reconfigure the I2S GPIO for PDM TX mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to PDM TX mode, i.e., `i2s_channel_init_pdm_tx_mode` has been called before reconfiguring

参数

- **handle** -- **[in]** I2S TX channel handler
- **gpio_cfg** -- **[in]** PDM TX mode GPIO configuration, specified by user

返回

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

Structures

struct **i2s_pdm_rx_slot_config_t**

I2S slot configuration for PDM RX mode.

Public Members

i2s_data_bit_width_t **data_bit_width**

I2S sample data bit width (valid data bits per sample), only support 16 bits for PDM mode

i2s_slot_bit_width_t **slot_bit_width**

I2S slot bit width (total bits per slot) , only support 16 bits for PDM mode

i2s_slot_mode_t **slot_mode**

Set mono or stereo mode with I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

i2s_pdm_slot_mask_t **slot_mask**

Choose the slots to activate

bool **hp_en**

High pass filter enable

float **hp_cut_off_freq_hz**

High pass filter cut-off frequency, range 23.3Hz ~ 185Hz, see cut-off frequency sheet above

uint32_t **amplify_num**

The amplification number of the final conversion result. The data that have converted from PDM to PCM module, will time `amplify_num` additionally to amplify the final result. Note that it's only a multiplier of the digital PCM data, not the gain of the analog signal range 1~15, default 1

struct **i2s_pdm_rx_clk_config_t**

I2S clock configuration for PDM RX mode.

Public Members

uint32_t **sample_rate_hz**

I2S sample rate

i2s_clock_src_t **clk_src**

Choose clock source

i2s_mclk_multiple_t **mclk_multiple**

The multiple of MCLK to the sample rate

i2s_pdm_dsr_t **dn_sample_mode**

Down-sampling rate mode

uint32_t **bclk_div**

The division from MCLK to BCLK. The typical and minimum value is I2S_PDM_RX_BCLK_DIV_MIN. It will be set to I2S_PDM_RX_BCLK_DIV_MIN by default if it is smaller than I2S_PDM_RX_BCLK_DIV_MIN

struct **i2s_pdm_rx_gpio_config_t**

I2S PDM TX mode GPIO pins configuration.

Public Members

gpio_num_t **clk**

PDM clk pin, output

gpio_num_t **din**

DATA pin 0, input

gpio_num_t **dins**[(4)]

DATA pins, input, only take effect when corresponding I2S_PDM_RX_LINEx_SLOT_xxx is enabled in *i2s_pdm_rx_slot_config_t::slot_mask*

uint32_t **clk_inv**

Set 1 to invert the clk output

struct *i2s_pdm_rx_gpio_config_t*::[anonymous] **invert_flags**

GPIO pin invert flags

struct **i2s_pdm_rx_config_t**

I2S PDM RX mode major configuration that including clock/slot/GPIO configuration.

Public Members

i2s_pdm_rx_clk_config_t **clk_cfg**

PDM RX clock configurations, can be generated by macro I2S_PDM_RX_CLK_DEFAULT_CONFIG

i2s_pdm_rx_slot_config_t **slot_cfg**

PDM RX slot configurations, can be generated by macro I2S_PDM_RX_SLOT_DEFAULT_CONFIG

i2s_pdm_rx_gpio_config_t **gpio_cfg**

PDM RX slot configurations, specified by user

struct **i2s_pdm_tx_slot_config_t**

I2S slot configuration for PDM TX mode.

Public Members

i2s_data_bit_width_t **data_bit_width**

I2S sample data bit width (valid data bits per sample), only support 16 bits for PDM mode

i2s_slot_bit_width_t **slot_bit_width**

I2S slot bit width (total bits per slot), only support 16 bits for PDM mode

i2s_slot_mode_t **slot_mode**

Set mono or stereo mode with I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO For PDM TX mode, mono means the data buffer only contains one slot data, Stereo means the data buffer contains two slots data

uint32_t **sd_prescale**

Sigma-delta filter prescale

i2s_pdm_sig_scale_t **sd_scale**

Sigma-delta filter scaling value

i2s_pdm_sig_scale_t **hp_scale**

High pass filter scaling value

i2s_pdm_sig_scale_t **lp_scale**

Low pass filter scaling value

i2s_pdm_sig_scale_t **sinc_scale**

Sinc filter scaling value

i2s_pdm_tx_line_mode_t **line_mode**

PDM TX line mode, one-line codec, one-line dac, two-line dac mode can be selected

bool **hp_en**

High pass filter enable

float **hp_cut_off_freq_hz**

High pass filter cut-off frequency, range 23.3Hz ~ 185Hz, see cut-off frequency sheet above

uint32_t **sd_dither**

Sigma-delta filter dither

uint32_t **sd_dither2**

Sigma-delta filter dither2

struct **i2s_pdm_tx_clk_config_t**

I2S clock configuration for PDM TX mode.

Public Members

uint32_t **sample_rate_hz**

I2S sample rate, not suggest to exceed 48000 Hz, otherwise more glitches and noise may appear

i2s_clock_src_t **clk_src**

Choose clock source

i2s_mclk_multiple_t **mclk_multiple**

The multiple of MCLK to the sample rate

uint32_t **up_sample_fp**

Up-sampling param fp

uint32_t **up_sample_fs**

Up-sampling param fs, not allowed to be greater than 480

uint32_t **bclk_div**

The division from MCLK to BCLK. The minimum value is I2S_PDM_TX_BCLK_DIV_MIN. It will be set to I2S_PDM_TX_BCLK_DIV_MIN by default if it is smaller than I2S_PDM_TX_BCLK_DIV_MIN

struct **i2s_pdm_tx_gpio_config_t**

I2S PDM TX mode GPIO pins configuration.

Public Members

gpio_num_t **clk**

PDM clk pin, output

gpio_num_t **dout**

DATA pin, output

gpio_num_t **dout2**

The second data pin for the DAC dual-line mode, only take effect when the line mode is `I2S_PDM_TX_TWO_LINE_DAC`

uint32_t **clk_inv**

Set 1 to invert the clk output

struct *i2s_pdm_tx_gpio_config_t*::[anonymous] **invert_flags**

GPIO pin invert flags

struct **i2s_pdm_tx_config_t**

I2S PDM TX mode major configuration that including clock/slot/GPIO configuration.

Public Members

i2s_pdm_tx_clk_config_t **clk_cfg**

PDM TX clock configurations, can be generated by macro I2S_PDM_TX_CLK_DEFAULT_CONFIG

i2s_pdm_tx_slot_config_t **slot_cfg**

PDM TX slot configurations, can be generated by macro I2S_PDM_TX_SLOT_DEFAULT_CONFIG

i2s_pdm_tx_gpio_config_t **gpio_cfg**

PDM TX GPIO configurations, specified by user

Macros

I2S_PDM_RX_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo)

PDM format in 2 slots(RX)

This file is specified for I2S PDM communication mode Features:

- Only support PDM TX/RX mode
- Fixed to 2 slots
- Data bit width only support 16 bits

参数

- **bits_per_sample** -- I2S data bit width, only support 16 bits for PDM mode
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

I2S_PDM_RX_CLK_DEFAULT_CONFIG(rate)

I2S default PDM RX clock configuration.

参数

- **rate** -- sample rate

I2S_PDM_TX_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo)

PDM style in 2 slots(TX) for codec line mode.

参数

- **bits_per_sample** -- I2S data bit width, only support 16 bits for PDM mode
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

I2S_PDM_TX_SLOT_DAC_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo)

PDM style in 1 slots(TX) for DAC line mode.

备注: The noise might be different with different configurations, this macro provides a set of configurations that have relatively high SNR (Signal Noise Ratio), you can also adjust them to fit your case.

参数

- **bits_per_sample** -- I2S data bit width, only support 16 bits for PDM mode
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

I2S_PDM_TX_CLK_DEFAULT_CONFIG(rate)

I2S default PDM TX clock configuration for codec line mode.

备注: TX PDM can only be set to the following two up-sampling rate configurations: 1: fp = 960, fs = sample_rate_hz / 100, in this case, Fpdm = 128*48000 2: fp = 960, fs = 480, in this case, Fpdm = 128*Fpcm = 128*sample_rate_hz If the PDM receiver do not care the PDM serial clock, it's recommended set Fpdm = 128*48000. Otherwise, the second configuration should be adopted.

参数

- **rate** -- sample rate (not suggest to exceed 48000 Hz, otherwise more glitches and noise may appear)

I2S_PDM_TX_CLK_DAC_DEFAULT_CONFIG(rate)

I2S default PDM TX clock configuration for DAC line mode.

备注: TX PDM can only be set to the following two up-sampling rate configurations: 1: fp = 960, fs = sample_rate_hz / 100, in this case, Fpdm = 128*48000 2: fp = 960, fs = 480, in this case, Fpdm = 128*Fpcm = 128*sample_rate_hz If the PDM receiver do not care the PDM serial clock, it's recommended set Fpdm = 128*48000. Otherwise, the second configuration should be adopted.

备注: The noise might be different with different configurations, this macro provides a set of configurations that have relatively high SNR (Signal Noise Ratio), you can also adjust them to fit your case.

参数

- **rate** -- sample rate (not suggest to exceed 48000 Hz, otherwise more glitches and noise may appear)

TDM 模式

Header File

- components/esp_driver_i2s/include/driver/i2s_tdm.h
- This header file can be included with:

```
#include "driver/i2s_tdm.h"
```

- This header file is a part of the API provided by the `esp_driver_i2s` component. To declare that your component depends on `esp_driver_i2s`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_i2s
```

or

```
PRIV_REQUIRES esp_driver_i2s
```

Functions

esp_err_t **i2s_channel_init_tdm_mode**(*i2s_chan_handle_t* handle, const *i2s_tdm_config_t* *tdm_cfg)

Initialize I2S channel to TDM mode.

备注: Only allowed to be called when the channel state is REGISTERED, (i.e., channel has been allocated, but not initialized) and the state will be updated to READY if initialization success, otherwise the state will return to REGISTERED.

参数

- **handle** -- **[in]** I2S channel handler
- **tdm_cfg** -- **[in]** Configurations for TDM mode, including clock, slot and GPIO The clock configuration can be generated by the helper macro `I2S_TDM_CLK_DEFAULT_CONFIG` The slot configuration can be generated by the helper macro `I2S_TDM_PHILIPS_SLOT_DEFAULT_CONFIG`, `I2S_TDM_PCM_SHORT_SLOT_DEFAULT_CONFIG`, `I2S_TDM_PCM_LONG_SLOT_DEFAULT_CONFIG` or `I2S_TDM_MSB_SLOT_DEFAULT_CONFIG`

返回

- ESP_OK Initialize successfully
- ESP_ERR_NO_MEM No memory for storing the channel information
- ESP_ERR_INVALID_ARG NULL pointer or invalid configuration
- ESP_ERR_INVALID_STATE This channel is not registered

esp_err_t **i2s_channel_reconfig_tdm_clock**(*i2s_chan_handle_t* handle, const *i2s_tdm_clk_config_t* *clk_cfg)

Reconfigure the I2S clock for TDM mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to TDM mode, i.e., `i2s_channel_init_tdm_mode` has been called before reconfiguring

---

参数

- **handle** -- **[in]** I2S channel handler

- **clk_cfg** -- **[in]** Standard mode clock configuration, can be generated by I2S_TDM_CLK_DEFAULT_CONFIG

返回

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not TDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **i2s_channel_reconfig_tdm_slot**(*i2s_chan_handle_t* handle, const *i2s_tdm_slot_config_t* *slot_cfg)

Reconfigure the I2S slot for TDM mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. i2s_channel_disable should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to TDM mode, i.e., i2s_channel_init_tdm_mode has been called before reconfiguring

参数

- **handle** -- **[in]** I2S channel handler
- **slot_cfg** -- **[in]** Standard mode slot configuration, can be generated by I2S_TDM_PHILIPS_SLOT_DEFAULT_CONFIG, I2S_TDM_PCM_SHORT_SLOT_DEFAULT_CONFIG, I2S_TDM_PCM_LONG_SLOT_DEFAULT_CONFIG or I2S_TDM_MSB_SLOT_DEFAULT_CONFIG.

返回

- ESP_OK Set clock successfully
- ESP_ERR_NO_MEM No memory for DMA buffer
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not TDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

esp_err_t **i2s_channel_reconfig_tdm_gpio**(*i2s_chan_handle_t* handle, const *i2s_tdm_gpio_config_t* *gpio_cfg)

Reconfigure the I2S GPIO for TDM mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won't change the state. i2s_channel_disable should be called before calling this function if I2S has started.

备 注: The input channel handle has to be initialized to TDM mode, i.e., i2s_channel_init_tdm_mode has been called before reconfiguring

参数

- **handle** -- **[in]** I2S channel handler
- **gpio_cfg** -- **[in]** Standard mode GPIO configuration, specified by user

返回

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not TDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

Structures

struct **i2s_tdm_slot_config_t**

I2S slot configuration for TDM mode.

Public Members

i2s_data_bit_width_t **data_bit_width**

I2S sample data bit width (valid data bits per sample)

i2s_slot_bit_width_t **slot_bit_width**

I2S slot bit width (total bits per slot)

i2s_slot_mode_t **slot_mode**

Set mono or stereo mode with I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

i2s_tdm_slot_mask_t **slot_mask**

Slot mask. Activating slots by setting 1 to corresponding bits. When the activated slots is not consecutive, those data in inactivated slots will be ignored

uint32_t **ws_width**

WS signal width (i.e. the number of BCLK ticks that WS signal is high)

bool **ws_pol**

WS signal polarity, set true to enable high lever first

bool **bit_shift**

Set true to enable bit shift in Philips mode

bool **left_align**

Set true to enable left alignment

bool **big_endian**

Set true to enable big endian

bool **bit_order_lsb**

Set true to enable lsb first

bool **skip_mask**

Set true to enable skip mask. If it is enabled, only the data of the enabled channels will be sent, otherwise all data stored in DMA TX buffer will be sent

uint32_t **total_slot**

I2S total number of slots. If it is smaller than the biggest activated channel number, it will be set to this number automatically.

struct **i2s_tdm_clk_config_t**

I2S clock configuration for TDM mode.

Public Members

uint32_t **sample_rate_hz**

I2S sample rate

i2s_clock_src_t **clk_src**

Choose clock source, see `soc_periph_i2s_clk_src_t` for the supported clock sources. selected `I2S_CLK_SRC_EXTERNAL` (if supports) to enable the external source clock inputted via MCLK pin, please make sure the frequency inputted is equal or greater than `sample_rate_hz * mclk_multiple`

uint32_t **ext_clk_freq_hz**

External clock source frequency in Hz, only take effect when `clk_src = I2S_CLK_SRC_EXTERNAL`, otherwise this field will be ignored Please make sure the frequency inputted is equal or greater than BCLK, i.e. `sample_rate_hz * slot_bits * slot_num`

i2s_mclk_multiple_t **mclk_multiple**

The multiple of MCLK to the sample rate, only take effect for master role

uint32_t **bclk_div**

The division from MCLK to BCLK, only take effect for slave role, it shouldn't be smaller than 8. Increase this field when data sent by slave lag behind

struct **i2s_tdm_gpio_config_t**

I2S TDM mode GPIO pins configuration.

Public Members

gpio_num_t **mclk**

MCK pin, output by default, input if the clock source is selected to `I2S_CLK_SRC_EXTERNAL`

gpio_num_t **bclk**

BCK pin, input in slave role, output in master role

gpio_num_t **ws**

WS pin, input in slave role, output in master role

gpio_num_t **dout**

DATA pin, output

gpio_num_t **din**

DATA pin, input

uint32_t **mclk_inv**

Set 1 to invert the MCLK input/output

uint32_t **bclk_inv**

Set 1 to invert the BCLK input/output

uint32_t **ws_inv**

Set 1 to invert the WS input/output

struct *i2s_tdm_gpio_config_t*::[anonymous] **invert_flags**

GPIO pin invert flags

struct **i2s_tdm_config_t**

I2S TDM mode major configuration that including clock/slot/GPIO configuration.

Public Members

i2s_tdm_clk_config_t **clk_cfg**

TDM mode clock configuration, can be generated by macro I2S_TDM_CLK_DEFAULT_CONFIG

i2s_tdm_slot_config_t **slot_cfg**

TDM mode slot configuration, can be generated by macros I2S_TDM_[mode]_SLOT_DEFAULT_CONFIG, [mode] can be replaced with PHILIPS/MSB/PCM_SHORT/PCM_LONG

i2s_tdm_gpio_config_t **gpio_cfg**

TDM mode GPIO configuration, specified by user

Macros

I2S_TDM_AUTO_SLOT_NUM

This file is specified for I2S TDM communication mode Features:

- More than 2 slots

I2S_TDM_AUTO_WS_WIDTH

I2S_TDM_PHILIPS_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo, mask)

Philips format in active slot that enabled by mask.

参数

- **bits_per_sample** -- I2S data bit width
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO
- **mask** -- active slot mask

I2S_TDM_MSB_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo, mask)

MSB format in active slot enabled that by mask.

参数

- **bits_per_sample** -- I2S data bit width
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO
- **mask** -- active slot mask

I2S_TDM_PCM_SHORT_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo, mask)

PCM(short) format in active slot that enabled by mask.

参数

- **bits_per_sample** -- I2S data bit width
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO
- **mask** -- active slot mask

I2S_TDM_PCM_LONG_SLOT_DEFAULT_CONFIG(bits_per_sample, mono_or_stereo, mask)

PCM(long) format in active slot that enabled by mask.

参数

- **bits_per_sample** -- I2S data bit width
- **mono_or_stereo** -- I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO
- **mask** -- active slot mask

I2S_TDM_CLK_DEFAULT_CONFIG(rate)

I2S default TDM clock configuration.

备注: Please set the mclk_multiple to I2S_MCLK_MULTIPLE_384 while the data width in slot configuration is set to 24 bits Otherwise the sample rate might be imprecise since the BCLK division is not a integer

参数

- **rate** -- sample rate

I2S 驱动

Header File

- components/esp_driver_i2s/include/driver/i2s_common.h
- This header file can be included with:

```
#include "driver/i2s_common.h"
```

- This header file is a part of the API provided by the `esp_driver_i2s` component. To declare that your component depends on `esp_driver_i2s`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_i2s
```

or

```
PRIV_REQUIRES esp_driver_i2s
```

Functions

esp_err_t **i2s_new_channel**(const *i2s_chan_config_t* *chan_cfg, *i2s_chan_handle_t* *ret_tx_handle, *i2s_chan_handle_t* *ret_rx_handle)

Allocate new I2S channel(s)

备注: The new created I2S channel handle will be REGISTERED state after it is allocated successfully.

备注: When the port id in channel configuration is I2S_NUM_AUTO, driver will allocate I2S port automatically on one of the I2S controller, otherwise driver will try to allocate the new channel on the selected port.

备注: If both tx_handle and rx_handle are not NULL, it means this I2S controller will work at full-duplex mode, the RX and TX channels will be allocated on a same I2S port in this case. Note that some configurations of TX/RX channel are shared on ESP32 and ESP32S2, so please make sure they are working at same condition and under same status(start/stop). Currently, full-duplex mode can't guarantee TX/RX channels write/read synchronously, they can only share the clock signals for now.

备注： If tx_handle OR rx_handle is NULL, it means this I2S controller will work at simplex mode. For ESP32 and ESP32S2, the whole I2S controller (i.e. both RX and TX channel) will be occupied, even if only one of RX or TX channel is registered. For the other targets, another channel on this controller will still available.

参数

- **chan_cfg** -- **[in]** I2S controller channel configurations
- **ret_tx_handle** -- **[out]** I2S channel handler used for managing the sending channel(optional)
- **ret_rx_handle** -- **[out]** I2S channel handler used for managing the receiving channel(optional)

返回

- ESP_OK Allocate new channel(s) success
- ESP_ERR_NOT_SUPPORTED The communication mode is not supported on the current chip
- ESP_ERR_INVALID_ARG NULL pointer or illegal parameter in *i2s_chan_config_t*
- ESP_ERR_NOT_FOUND No available I2S channel found

esp_err_t **i2s_del_channel**(*i2s_chan_handle_t* handle)

Delete the I2S channel.

备注： Only allowed to be called when the I2S channel is at REGISTERED or READY state (i.e., it should stop before deleting it).

备注： Resource will be free automatically if all channels in one port are deleted

参数 **handle** -- **[in]** I2S channel handler

- ESP_OK Delete successfully
- ESP_ERR_INVALID_ARG NULL pointer

esp_err_t **i2s_channel_get_info**(*i2s_chan_handle_t* handle, *i2s_chan_info_t* *chan_info)

Get I2S channel information.

参数

- **handle** -- **[in]** I2S channel handler
- **chan_info** -- **[out]** I2S channel basic information

返回

- ESP_OK Get I2S channel information success
- ESP_ERR_NOT_FOUND The input handle doesn't match any registered I2S channels, it may not an I2S channel handle or not available any more
- ESP_ERR_INVALID_ARG The input handle or chan_info pointer is NULL

esp_err_t **i2s_channel_enable**(*i2s_chan_handle_t* handle)

Enable the I2S channel.

备注： Only allowed to be called when the channel state is READY, (i.e., channel has been initialized, but not started) the channel will enter RUNNING state once it is enabled successfully.

备注： Enable the channel can start the I2S communication on hardware. It will start outputting BCLK and WS signal. For MCLK signal, it will start to output when initialization is finished

参数 **handle** -- **[in]** I2S channel handler

- ESP_OK Start successfully
- ESP_ERR_INVALID_ARG NULL pointer
- ESP_ERR_INVALID_STATE This channel has not initialized or already started

esp_err_t **i2s_channel_disable**(*i2s_chan_handle_t* handle)

Disable the I2S channel.

备注： Only allowed to be called when the channel state is RUNNING, (i.e., channel has been started) the channel will enter READY state once it is disabled successfully.

备注： Disable the channel can stop the I2S communication on hardware. It will stop BCLK and WS signal but not MCLK signal

---

参数 **handle** -- **[in]** I2S channel handler

返回

- ESP_OK Stop successfully
- ESP_ERR_INVALID_ARG NULL pointer
- ESP_ERR_INVALID_STATE This channel has not stated

esp_err_t **i2s_channel_preload_data**(*i2s_chan_handle_t* tx_handle, const void *src, size_t size, size_t *bytes_loaded)

Preload the data into TX DMA buffer.

备注： Only allowed to be called when the channel state is READY, (i.e., channel has been initialized, but not started)

备注： As the initial DMA buffer has no data inside, it will transmit the empty buffer after enabled the channel, this function is used to preload the data into the DMA buffer, so that the valid data can be transmitted immediately after the channel is enabled.

备注： This function can be called multiple times before enabling the channel, the buffer that loaded later will be concatenated behind the former loaded buffer. But when all the DMA buffers have been loaded, no more data can be preload then, please check the `bytes_loaded` parameter to see how many bytes are loaded successfully, when the `bytes_loaded` is smaller than the `size`, it means the DMA buffers are full.

参数

- **tx_handle** -- **[in]** I2S TX channel handler
- **src** -- **[in]** The pointer of the source buffer to be loaded
- **size** -- **[in]** The source buffer size
- **bytes_loaded** -- **[out]** The bytes that successfully been loaded into the TX DMA buffer

返回

- ESP_OK Load data successful
- ESP_ERR_INVALID_ARG NULL pointer or not TX direction
- ESP_ERR_INVALID_STATE This channel has not stated

esp_err_t **i2s_channel_write**(*i2s_chan_handle_t* handle, const void *src, size_t size, size_t *bytes_written, uint32_t timeout_ms)

I2S write data.

备注： Only allowed to be called when the channel state is RUNNING, (i.e., TX channel has been started and is not writing now) but the RUNNING only stands for the software state, it doesn't mean there is no the signal transporting on line.

参数

- **handle** -- **[in]** I2S channel handler
- **src** -- **[in]** The pointer of sent data buffer
- **size** -- **[in]** Max data buffer length
- **bytes_written** -- **[out]** Byte number that actually be sent, can be NULL if not needed
- **timeout_ms** -- **[in]** Max block time

返回

- ESP_OK Write successfully
- ESP_ERR_INVALID_ARG NULL pointer or this handle is not TX handle
- ESP_ERR_TIMEOUT Writing timeout, no writing event received from ISR within ticks_to_wait
- ESP_ERR_INVALID_STATE I2S is not ready to write

esp_err_t **i2s_channel_read**(*i2s_chan_handle_t* handle, void *dest, size_t size, size_t *bytes_read, uint32_t timeout_ms)

I2S read data.

备注： Only allowed to be called when the channel state is RUNNING but the RUNNING only stands for the software state, it doesn't mean there is no the signal transporting on line.

参数

- **handle** -- **[in]** I2S channel handler
- **dest** -- **[in]** The pointer of receiving data buffer
- **size** -- **[in]** Max data buffer length
- **bytes_read** -- **[out]** Byte number that actually be read, can be NULL if not needed
- **timeout_ms** -- **[in]** Max block time

返回

- ESP_OK Read successfully
- ESP_ERR_INVALID_ARG NULL pointer or this handle is not RX handle
- ESP_ERR_TIMEOUT Reading timeout, no reading event received from ISR within ticks_to_wait
- ESP_ERR_INVALID_STATE I2S is not ready to read

esp_err_t **i2s_channel_register_event_callback**(*i2s_chan_handle_t* handle, const *i2s_event_callbacks_t* *callbacks, void *user_data)

Set event callbacks for I2S channel.

备注： Only allowed to be called when the channel state is REGISTERED / READY, (i.e., before channel starts)

备注： User can deregister a previously registered callback by calling this function and setting the callback member in the `callbacks` structure to NULL.

备注： When CONFIG_I2S_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. The `user_data`

should also reside in SRAM or internal RAM as well.

参数

- **handle** -- **[in]** I2S channel handler
- **callbacks** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK Set event callbacks successfully
- ESP_ERR_INVALID_ARG Set event callbacks failed because of invalid argument
- ESP_ERR_INVALID_STATE Set event callbacks failed because the current channel state is not REGISTERED or READY

Structures

struct **i2s_event_callbacks_t**

Group of I2S callbacks.

备注： The callbacks are all running under ISR environment

备注： When CONFIG_I2S_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

Public Members

i2s_isr_callback_t **on_recv**

Callback of data received event, only for RX channel The event data includes DMA buffer address and size that just finished receiving data

i2s_isr_callback_t **on_recv_q_ovf**

Callback of receiving queue overflowed event, only for RX channel The event data includes buffer size that has been overwritten

i2s_isr_callback_t **on_sent**

Callback of data sent event, only for TX channel The event data includes DMA buffer address and size that just finished sending data

i2s_isr_callback_t **on_send_q_ovf**

Callback of sending queue overflowed event, only for TX channel The event data includes buffer size that has been overwritten

struct **i2s_chan_config_t**

I2S controller channel configuration.

Public Members

i2s_port_t **id**

I2S port id

i2s_role_t **role**

I2S role, I2S_ROLE_MASTER or I2S_ROLE_SLAVE

uint32_t **dma_desc_num**

I2S DMA buffer number, it is also the number of DMA descriptor

uint32_t **dma_frame_num**

I2S frame number in one DMA buffer. One frame means one-time sample data in all slots, it should be the multiple of `3` when the data bit width is 24.

bool **auto_clear**

Alias of `auto_clear_after_cb`

bool **auto_clear_after_cb**

Set to auto clear DMA TX buffer after `on_sent` callback, I2S will always send zero automatically if no data to send. So that user can assign the data to the DMA buffers directly in the callback, and the data won't be cleared after quit the callback.

bool **auto_clear_before_cb**

Set to auto clear DMA TX buffer before `on_sent` callback, I2S will always send zero automatically if no data to send So that user can access data in the callback that just finished to send.

int **intr_priority**

I2S interrupt priority, range [0, 7], if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

struct **i2s_chan_info_t**

I2S channel information.

Public Members

i2s_port_t **id**

I2S port id

i2s_role_t **role**

I2S role, I2S_ROLE_MASTER or I2S_ROLE_SLAVE

i2s_dir_t **dir**

I2S channel direction

i2s_comm_mode_t **mode**

I2S channel communication mode

i2s_chan_handle_t **pair_chan**

I2S pair channel handle in duplex mode, always NULL in simplex mode

uint32_t **total_dma_buf_size**

Total size of all the allocated DMA buffers

- 0 if the channel has not been initialized
- non-zero if the channel has been initialized

Macros

I2S_CHANNEL_DEFAULT_CONFIG(i2s_num, i2s_role)

get default I2S property

I2S_GPIO_UNUSED

Used in i2s_gpio_config_t for signals which are not used

I2S 类型

Header File

- components/esp_driver_i2s/include/driver/i2s_types.h
- This header file can be included with:

```
#include "driver/i2s_types.h"
```

- This header file is a part of the API provided by the `esp_driver_i2s` component. To declare that your component depends on `esp_driver_i2s`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_i2s
```

or

```
PRIV_REQUIRES esp_driver_i2s
```

Structures

struct **i2s_event_data_t**

Event structure used in I2S event queue.

Public Members

void ***data**

(Deprecated) The secondary pointer of DMA buffer that just finished sending or receiving for `on_recv` and `on_sent` callback NULL for `on_recv_q_ovf` and `on_send_q_ovf` callback

void ***dma_buf**

The first level pointer of DMA buffer that just finished sending or receiving for `on_recv` and `on_sent` callback NULL for `on_recv_q_ovf` and `on_send_q_ovf` callback

size_t **size**

The buffer size of DMA buffer when success to send or receive, also the buffer size that dropped when queue overflow. It is related to the dma_frame_num and data_bit_width, typically it is fixed when data_bit_width is not changed.

Type Definitions

typedef struct i2s_channel_obj_t ***i2s_chan_handle_t**

I2S channel object handle, the control unit of the I2S driver

typedef bool (***i2s_isr_callback_t**)(*i2s_chan_handle_t* handle, *i2s_event_data_t* *event, void *user_ctx)

I2S event callback.

Param handle **[in]** I2S channel handle, created from `i2s_new_channel()`

Param event [in] I2S event data
Param user_ctx [in] User registered context, passed from `i2s_channel_register_event_callback()`
Return Whether a high priority task has been waken up by this callback function

Enumerations

enum **i2s_port_t**

I2S controller port number, the max port number is (SOC_I2S_NUM -1).

Values:

enumerator **I2S_NUM_0**

I2S controller port 0

enumerator **I2S_NUM_1**

I2S controller port 1

enumerator **I2S_NUM_2**

I2S controller port 2

enumerator **I2S_NUM_AUTO**

Select whichever port is available

enum **i2s_comm_mode_t**

I2S controller communication mode.

Values:

enumerator **I2S_COMM_MODE_STD**

I2S controller using standard communication mode, support Philips/MSB/PCM format

enumerator **I2S_COMM_MODE_PDM**

I2S controller using PDM communication mode, support PDM output or input

enumerator **I2S_COMM_MODE_TDM**

I2S controller using TDM communication mode, support up to 16 slots per frame

enumerator **I2S_COMM_MODE_NONE**

Unspecified I2S controller mode

enum **i2s_mclk_multiple_t**

The multiple of MCLK to sample rate.

备注: MCLK is the minimum resolution of the I2S clock. Increasing mclk multiple can reduce the clock jitter of BCLK and WS, which is also useful for the codec that don't require MCLK but have strict requirement to BCLK. For the 24-bit slot width, please choose a multiple that can be divided by 3 (i.e. 24-bit compatible).

Values:

enumerator **I2S_MCLK_MULTIPLE_128**

MCLK = sample_rate * 128

enumerator **I2S_MCLK_MULTIPLE_192**

MCLK = sample_rate * 192 (24-bit compatible)

enumerator **I2S_MCLK_MULTIPLE_256**

MCLK = sample_rate * 256

enumerator **I2S_MCLK_MULTIPLE_384**

MCLK = sample_rate * 384 (24-bit compatible)

enumerator **I2S_MCLK_MULTIPLE_512**

MCLK = sample_rate * 512

enumerator **I2S_MCLK_MULTIPLE_576**

MCLK = sample_rate * 576 (24-bit compatible)

enumerator **I2S_MCLK_MULTIPLE_768**

MCLK = sample_rate * 768 (24-bit compatible)

enumerator **I2S_MCLK_MULTIPLE_1024**

MCLK = sample_rate * 1024

enumerator **I2S_MCLK_MULTIPLE_1152**

MCLK = sample_rate * 1152 (24-bit compatible)

Header File

- components/hal/include/hal/i2s_types.h
- This header file can be included with:

```
#include "hal/i2s_types.h"
```

Type Definitions

typedef *soc_periph_i2s_clk_src_t* **i2s_clock_src_t**

I2S clock source

Enumerations

enum **i2s_slot_mode_t**

I2S channel slot mode.

Values:

enumerator **I2S_SLOT_MODE_MONO**

I2S channel slot format mono, transmit same data in all slots for tx mode, only receive the data in the first slots for rx mode.

enumerator **I2S_SLOT_MODE_STEREO**

I2S channel slot format stereo, transmit different data in different slots for tx mode, receive the data in all slots for rx mode.

enum **i2s_dir_t**

I2S channel direction.

Values:

enumerator **I2S_DIR_RX**

I2S channel direction RX

enumerator **I2S_DIR_TX**

I2S channel direction TX

enum **i2s_role_t**

I2S controller role.

Values:

enumerator **I2S_ROLE_MASTER**

I2S controller master role, bclk and ws signal will be set to output

enumerator **I2S_ROLE_SLAVE**

I2S controller slave role, bclk and ws signal will be set to input

enum **i2s_data_bit_width_t**

Available data bit width in one slot.

Values:

enumerator **I2S_DATA_BIT_WIDTH_8BIT**

I2S channel data bit-width: 8

enumerator **I2S_DATA_BIT_WIDTH_16BIT**

I2S channel data bit-width: 16

enumerator **I2S_DATA_BIT_WIDTH_24BIT**

I2S channel data bit-width: 24

enumerator **I2S_DATA_BIT_WIDTH_32BIT**

I2S channel data bit-width: 32

enum **i2s_slot_bit_width_t**

Total slot bit width in one slot.

Values:

enumerator **I2S_SLOT_BIT_WIDTH_AUTO**

I2S channel slot bit-width equals to data bit-width

enumerator **I2S_SLOT_BIT_WIDTH_8BIT**

I2S channel slot bit-width: 8

enumerator **I2S_SLOT_BIT_WIDTH_16BIT**

I2S channel slot bit-width: 16

enumerator **I2S_SLOT_BIT_WIDTH_24BIT**

I2S channel slot bit-width: 24

enumerator **I2S_SLOT_BIT_WIDTH_32BIT**

I2S channel slot bit-width: 32

enum **i2s_pcm_compress_t**

A/U-law decompress or compress configuration.

Values:

enumerator **I2S_PCM_DISABLE**

Disable A/U law decompress or compress

enumerator **I2S_PCM_A_DECOMPRESS**

A-law decompress

enumerator **I2S_PCM_A_COMPRESS**

A-law compress

enumerator **I2S_PCM_U_DECOMPRESS**

U-law decompress

enumerator **I2S_PCM_U_COMPRESS**

U-law compress

enum **i2s_pdm_dsr_t**

I2S PDM RX down-sampling mode.

Values:

enumerator **I2S_PDM_DSR_8S**

downsampling number is 8 for PDM RX mode

enumerator **I2S_PDM_DSR_16S**

downsampling number is 16 for PDM RX mode

enumerator **I2S_PDM_DSR_MAX**

enum **i2s_pdm_sig_scale_t**

pdm tx singnal scaling mode

Values:

enumerator **I2S_PDM_SIG_SCALING_DIV_2**

I2S TX PDM signal scaling: /2

enumerator **I2S_PDM_SIG_SCALING_MUL_1**

I2S TX PDM signal scaling: x1

enumerator **I2S_PDM_SIG_SCALING_MUL_2**

I2S TX PDM signal scaling: x2

enumerator **I2S_PDM_SIG_SCALING_MUL_4**

I2S TX PDM signal scaling: x4

enum **i2s_pdm_tx_line_mode_t**

PDM TX line mode.

备注: For the standard codec mode, PDM pins are connect to a codec which requires both clock signal and data signal For the DAC output mode, PDM data signal can be connected to a power amplifier directly with a low-pass filter, normally, DAC output mode doesn't need the clock signal.

Values:

enumerator **I2S_PDM_TX_ONE_LINE_CODEC**

Standard PDM format output, left and right slot data on a single line

enumerator **I2S_PDM_TX_ONE_LINE_DAC**

PDM DAC format output, left or right slot data on a single line

enumerator **I2S_PDM_TX_TWO_LINE_DAC**

PDM DAC format output, left and right slot data on separated lines

enum **i2s_std_slot_mask_t**

I2S slot select in standard mode.

备注: It has different meanings in tx/rx/mono/stereo mode, and it may have differen behaviors on different targets For the details, please refer to the I2S API reference

---

Values:

enumerator **I2S_STD_SLOT_LEFT**

I2S transmits or receives left slot

enumerator **I2S_STD_SLOT_RIGHT**

I2S transmits or receives right slot

enumerator **I2S_STD_SLOT_BOTH**

I2S transmits or receives both left and right slot

enum **i2s_pdm_slot_mask_t**

I2S slot select in PDM mode.

Values:

enumerator **I2S_PDM_SLOT_RIGHT**

I2S PDM only transmits or receives the PDM device whose 'select' pin is pulled up

enumerator **I2S_PDM_SLOT_LEFT**

I2S PDM only transmits or receives the PDM device whose 'select' pin is pulled down

enumerator **I2S_PDM_SLOT_BOTH**

I2S PDM transmits or receives both two slots

enumerator **I2S_PDM_RX_LINE0_SLOT_RIGHT**

I2S PDM receives the right slot on line 0

enumerator **I2S_PDM_RX_LINE0_SLOT_LEFT**

I2S PDM receives the left slot on line 0

enumerator **I2S_PDM_RX_LINE1_SLOT_RIGHT**

I2S PDM receives the right slot on line 1

enumerator **I2S_PDM_RX_LINE1_SLOT_LEFT**

I2S PDM receives the left slot on line 1

enumerator **I2S_PDM_RX_LINE2_SLOT_RIGHT**

I2S PDM receives the right slot on line 2

enumerator **I2S_PDM_RX_LINE2_SLOT_LEFT**

I2S PDM receives the left slot on line 2

enumerator **I2S_PDM_RX_LINE3_SLOT_RIGHT**

I2S PDM receives the right slot on line 3

enumerator **I2S_PDM_RX_LINE3_SLOT_LEFT**

I2S PDM receives the left slot on line 3

enumerator **I2S_PDM_LINE_SLOT_ALL**

I2S PDM receives all slots

enum **i2s_tdm_slot_mask_t**

tdm slot number

备注： Multiple slots in TDM mode. For TX module, only the active slot send the audio data, the inactive slot send a constant or will be skipped if 'skip_msk' is set. For RX module, only receive the audio data in active slots, the data in inactive slots will be ignored. the bit map of active slot can not exceed (0x1«total_slot_num). e.g: slot_mask = (I2S_TDM_SLOT0 | I2S_TDM_SLOT3), here the active slot number is 2 and total_slot is not supposed to be smaller than 4.

Values:

enumerator **I2S_TDM_SLOT0**

I2S slot 0 enabled

enumerator **I2S_TDM_SLOT1**

I2S slot 1 enabled

enumerator **I2S_TDM_SLOT2**
I2S slot 2 enabled

enumerator **I2S_TDM_SLOT3**
I2S slot 3 enabled

enumerator **I2S_TDM_SLOT4**
I2S slot 4 enabled

enumerator **I2S_TDM_SLOT5**
I2S slot 5 enabled

enumerator **I2S_TDM_SLOT6**
I2S slot 6 enabled

enumerator **I2S_TDM_SLOT7**
I2S slot 7 enabled

enumerator **I2S_TDM_SLOT8**
I2S slot 8 enabled

enumerator **I2S_TDM_SLOT9**
I2S slot 9 enabled

enumerator **I2S_TDM_SLOT10**
I2S slot 10 enabled

enumerator **I2S_TDM_SLOT11**
I2S slot 11 enabled

enumerator **I2S_TDM_SLOT12**
I2S slot 12 enabled

enumerator **I2S_TDM_SLOT13**
I2S slot 13 enabled

enumerator **I2S_TDM_SLOT14**
I2S slot 14 enabled

enumerator **I2S_TDM_SLOT15**
I2S slot 15 enabled

2.5.14 Image Signal Processor

Introduction

ESP32-P4 includes an Image Signal Processor (ISP), which is a feature pipeline that consists of many image processing algorithms. ISP receives image data from the DVP camera or MIPI-CSI camera, or system memory, and writes the processed image data to the system memory through DMA. ISP shall work with other modules to read and write data, it can not work alone.

Terminology

- MIPI-CSI: Camera serial interface, a high-speed serial interface for cameras compliant with MIPI specifications
- DVP: Digital video parallel interface, generally composed of vsync, hsync, de, and data signals
- RAW: Unprocessed data directly output from an image sensor, typically divided into R, Gr, Gb, and B four channels classified into RAW8, RAW10, RAW12, etc., based on bit width
- RGB: Colored image format composed of red, green, and blue colors classified into RGB888, RGB565, etc., based on the bit width of each color
- YUV: Colored image format composed of luminance and chrominance classified into YUV444, YUV422, YUV420, etc., based on the data arrangement
- BF: Bayer Domain Filter
- AF: Auto-focus
- AWB: Auto-white balance
- CCM: Color correction matrix

ISP Pipeline

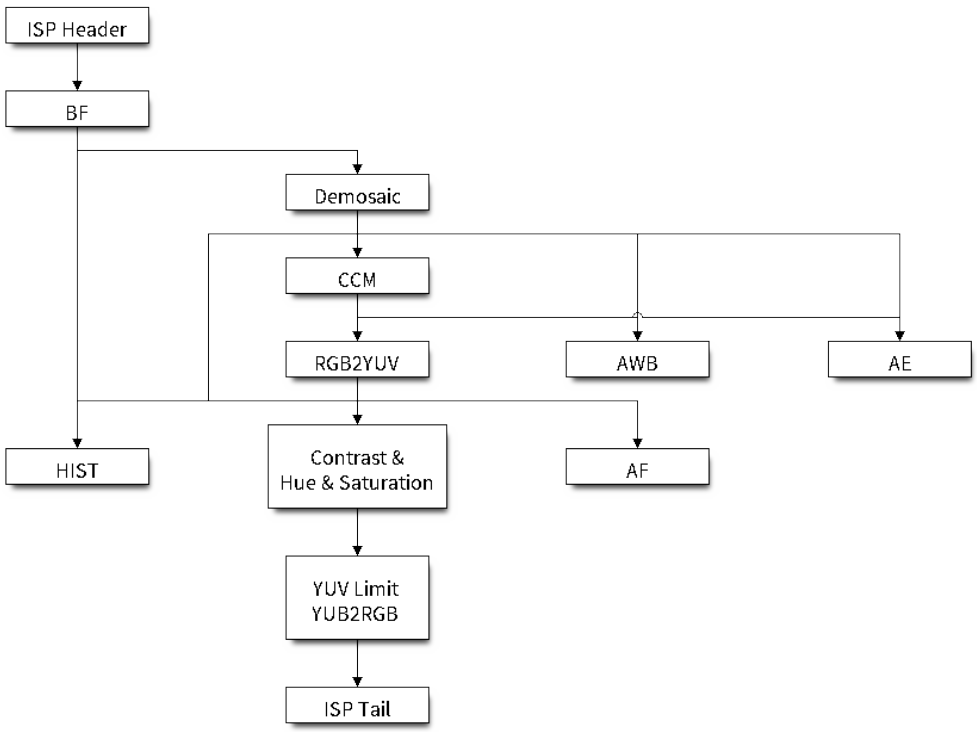


图 15: ISP Pipeline

Functional Overview

The ISP driver offers following services:

- *Resource Allocation* - covers how to allocate ISP resources with properly set of configurations. It also covers how to recycle the resources when they finished working.
- *Enable and disable ISP processor* - covers how to enable and disable an ISP processor.
- *Get AF statistics in one shot or continuous way* - covers how to get AF statistics one-shot or continuously.

- *Get AWB statistics in one shot or continuous way* - covers how to get AWB white patches statistics one-shot or continuously.
- *Configure CCM* - covers how to config the Color Correction Matrix.
- *Register callback* - covers how to hook user specific code to ISP driver event callback function.
- *Thread Safety* - lists which APIs are guaranteed to be thread safe by the driver.
- *Kconfig Options* - lists the supported Kconfig options that can bring different effects to the driver.
- *IRAM SAFE* - describes tips on how to make the ISP interrupt and control functions work better along with a disabled cache.

Resource Allocation

Install ISP Driver ISP driver requires the configuration that specified by `esp_isp_processor_cfg_t`.

If the configurations in `esp_isp_processor_cfg_t` is specified, users can call `esp_isp_new_processor()` to allocate and initialize an ISP processor. This function will return an ISP processor handle if it runs correctly. You can take following code as reference.

```
esp_isp_processor_cfg_t isp_config = {
    .clk_src = ISP_CLK_SRC_DEFAULT,
    .clk_hz = 80 * 1000 * 1000,
    .input_data_source = ISP_INPUT_DATA_SOURCE_CSI,
    .input_data_color_type = ISP_COLOR_RAW8,
    .output_data_color_type = ISP_COLOR_RGB565,
};

isp_proc_handle_t isp_proc = NULL;
ESP_ERROR_CHECK(esp_isp_new_processor(&isp_config, &isp_proc));
```

You can use the created handle to do driver enable / disable the ISP driver and do other ISP module installation.

Install ISP Auto-Focus (AF) Driver ISP auto-focus (AF) driver requires the configuration that specified by *esp_isp_af_config_t*.

If the configurations in *esp_isp_af_config_t* is specified, users can call *esp_isp_new_af_controller()* to allocate and initialize an ISP AF processor. This function will return an ISP AF processor handle if it runs correctly. You can take following code as reference.

```
esp_isp_af_config_t af_config = {
    .edge_thresh = 128,
};
isp_af_ctlr_t af_ctrlr = NULL;
ESP_ERROR_CHECK(esp_isp_new_af_controller(isp_proc, &af_config, &af_ctrlr));
```

You can use the created handle to do driver enable / disable the ISP AF driver and ISP AF Env module installation.

Install ISP Auto-White-Balance (AWB) Driver ISP auto-white-balance (AWB) driver requires the configuration specified by `esp_isp_awb_config_t`.

If an `esp_isp_awb_config_t` configuration is specified, you can call `esp_isp_new_awb_controller()` to allocate and initialize an ISP AWB processor. This function will return an ISP AWB processor handle on success. You can take following code as reference.

```
isp_awb_ctlr_t awb_ctlr = NULL;
uint32_t image_width = 800;
uint32_t image_height = 600;
/* The AWB configuration, please refer to the API comment for how to tune these␣
↪parameters */
esp_isp_awb_config_t awb_config = {
```

(下页继续)

(续上页)

```
    .sample_point = ISP_AWB_SAMPLE_POINT_AFTER_CCM,
    .window = {
        .top_left = {.x = image_width * 0.2, .y = image_height * 0.2},
        .btm_right = {.x = image_width * 0.8, .y = image_height * 0.8},
    },
    .white_patch = {
        .luminance = {.min = 0, .max = 220 * 3},
        .red_green_ratio = {.min = 0.0f, .max = 3.999f},
        .blue_green_ratio = {.min = 0.0f, .max = 3.999f},
    },
};
ESP_ERROR_CHECK(esp_isp_new_awb_controller(isp_proc, &awb_config, &awb_ctlr));
```

The AWB handle created in this step is required by other AWB APIs and AWB scheme.

Uninstall ISP Driver If a previously installed ISP processor is no longer needed, it's recommended to recycle the resource by calling `esp_isp_del_processor()`, so that to release the underlying hardware.

UnInstall ISP AF Driver If a previously installed ISP AF processor is no longer needed, it's recommended to recycle the resource by calling *`esp_isp_del_af_controller()`*, so that to release the underlying hardware.

UnInstall ISP AWB Driver If a previously installed ISP AWB processor is no longer needed, it's recommended to free the resource by calling `esp_isp_del_awb_controller()`, it will also release the underlying hardware.

Enable and Disable ISP

ISP

Before doing ISP pipeline, you need to enable the ISP processor first, by calling `esp_isp_enable()`. This function:

- Switches the driver state from **init** to **enable**.

Calling `esp_isp_disable()` does the opposite, that is, put the driver back to the **init** state.

ISP AF Processor

Before doing ISP AF, you need to enable the ISP AF processor first, by calling *`esp_isp_af_controller_enable()`*. This function:

- Switches the driver state from **init** to **enable**.

Calling *`esp_isp_af_controller_disable()`* does the opposite, that is, put the driver back to the **init** state.

AF One-shot and Continuous Statistics Calling *`esp_isp_af_controller_get_oneshot_statistics()`* to get oneshot AF statistics result. You can take following code as reference.

Aside from the above oneshot API, the ISP AF driver also provides a way to start AF statistics continuously. Calling *`esp_isp_af_controller_start_continuous_statistics()`* to start the continuous statistics and *`esp_isp_af_controller_stop_continuous_statistics()`* to stop it.

Note that if you want to use the continuous statistics, you need to register the *`esp_isp_af_env_detector_evt_cbs_t::on_env_statistics_done`* or *`esp_isp_af_env_detector_evt_cbs_t::on_env_change`* callback to get the statistics result. See how to register in *Register Event Callbacks*

```
esp_isp_af_config_t af_config = {
    .edge_thresh = 128,
};
isp_af_ctlr_t af_ctrlr = NULL;
ESP_ERROR_CHECK(esp_isp_new_af_controller(isp_proc, &af_config, &af_ctrlr));
ESP_ERROR_CHECK(esp_isp_af_controller_enable(af_ctrlr));
isp_af_result_t result = {};
/* Trigger the AF statistics and get its result for one time with timeout value␣
↪2000ms. */
ESP_ERROR_CHECK(esp_isp_af_controller_get_oneshot_statistics(af_ctrlr, 2000, &
↪result));

/* Start continuous AF statistics */
ESP_ERROR_CHECK(esp_isp_af_controller_start_continuous_statistics(af_ctrlr));
// You can do other stuffs here, the statistics result can be obtained in the␣
↪callback
// ......
// vTaskDelay(pdMS_TO_TICKS(1000));
/* Stop continuous AF statistics */
ESP_ERROR_CHECK(esp_isp_af_controller_stop_continuous_statistics(af_ctrlr));

/* Disable the af controller */
ESP_ERROR_CHECK(esp_isp_af_controller_disable(af_ctrlr));
/* Delete the af controller and free the resources */
ESP_ERROR_CHECK(esp_isp_del_af_controller(af_ctrlr));
```

Set AF Environment Detector Calling *esp_isp_af_controller_set_env_detector()* to set an ISP AF environment detector. You can take following code as reference.

```
esp_isp_af_env_config_t env_config = {
    .interval = 10,
};
isp_af_ctlr_t af_ctrlr = NULL;
ESP_ERROR_CHECK(esp_isp_new_af_controller(isp_proc, &af_config, &af_ctrlr));
ESP_ERROR_CHECK(esp_isp_af_controller_set_env_detector(af_ctrlr, &env_config));
```

Set AF Environment Detector Threshold Calling `esp_isp_af_env_detector_set_threshold()` to set the threshold of an ISP AF environment detector.

```
int definition_thresh = 0;
int luminance_thresh = 0;
ESP_ERROR_CHECK(esp_isp_af_env_detector_set_threshold(env_detector, definition_
↪thresh, luminance_thresh));
```

ISP AWB Processor

Before doing ISP AWB, you need to enable the ISP AWB processor first, by calling `esp_isp_awb_controller_enable()`. This function:

- Switches the driver state from **init** to **enable**.

Calling `esp_isp_awb_controller_disable()` does the opposite, that is, put the driver back to the **init** state.

AWB One-shot and Continuous Statistics Calling `esp_isp_awb_controller_get_oneshot_result()` to get oneshot AWB statistics result of white patches. You can take following code as reference.

Aside from the above oneshot API, the ISP AWB driver also provides a way to start AWB statistics continuously. Calling `esp_isp_awb_controller_start_continuous_statistics()` starts the continuous statistics and `esp_isp_awb_controller_stop_continuous_statistics()` stops it.

Note that if you want to use the continuous statistics, you need to register the `esp_isp_awb_cbs_t::on_statistics_done` callback to get the statistics result. See how to register it in *Register Event Callbacks*

```
bool example_isp_awb_on_statistics_done_cb(isp_awb_ctlr_t awb_ctlr, const esp_isp_
↪awb_evt_data_t *edata, void *user_data);
// ...
isp_awb_ctlr_t awb_ctlr = NULL;
uint32_t image_width = 800;
uint32_t image_height = 600;
/* The AWB configuration, please refer to the API comment for how to tune these␣
↪parameters */
esp_isp_awb_config_t awb_config = {
    .sample_point = ISP_AWB_SAMPLE_POINT_AFTER_CCM,
    .window = {
        .top_left = {.x = image_width * 0.2, .y = image_height * 0.2},
        .btm_right = {.x = image_width * 0.8, .y = image_height * 0.8},
    },
    .white_patch = {
        .luminance = {.min = 0, .max = 220 * 3},
        .red_green_ratio = {.min = 0.0f, .max = 3.999f},
        .blue_green_ratio = {.min = 0.0f, .max = 3.999f},
    },
};
isp_awb_stat_result_t stat_res = {};
/* Create the awb controller */
ESP_ERROR_CHECK(esp_isp_new_awb_controller(isp_proc, &awb_config, &awb_ctlr));
/* Register AWB callback */
esp_isp_awb_cbs_t awb_cb = {
    .on_statistics_done = example_isp_awb_on_statistics_done_cb,
};
ESP_ERROR_CHECK(esp_isp_awb_register_event_callbacks(awb_ctlr, &awb_cb, NULL));
/* Enabled the awb controller */
ESP_ERROR_CHECK(esp_isp_awb_controller_enable(awb_ctlr));

/* Get oneshot AWB statistics result */
ESP_ERROR_CHECK(esp_isp_awb_controller_get_oneshot_statistics(awb_ctlr, -1, &stat_
↪res));

/* Start continuous AWB statistics, note that continuous statistics requires `on_
↪statistics_done` callback */
ESP_ERROR_CHECK(esp_isp_awb_controller_start_continuous_statistics(awb_ctlr));
// You can do other stuffs here, the statistics result can be obtained in the␣
↪callback
// ......
// vTaskDelay(pdMS_TO_TICKS(1000));
/* Stop continuous AWB statistics */
ESP_ERROR_CHECK(esp_isp_awb_controller_stop_continuous_statistics(awb_ctlr));

/* Disable the awb controller */
ESP_ERROR_CHECK(esp_isp_awb_controller_disable(awb_ctlr));
/* Delete the awb controller and free the resources */
ESP_ERROR_CHECK(esp_isp_del_awb_controller(awb_ctlr));
```

Configure CCM Color Correction Matrix can scale the color ratio of RGB888 pixels. It can be used for adjusting the image color via some algorithms, for example, used for white balance by inputting the AWB computed result, or used as a Filter with some filter algorithms.

To adjust the color correction matrix, you can refer to the following code:

```
// ...
// Configure CCM
esp_isp_ccm_config_t ccm_cfg = {
    .matrix = {
        1.0, 0.0, 0.0,
        0.0, 1.0, 0.0,
        0.0, 0.0, 1.0
    },
    .saturation = false,
};
ESP_ERROR_CHECK(esp_isp_ccm_configure(isp_proc, &ccm_cfg));
// The configured CCM will be applied to the image once the CCM module is enabled
ESP_ERROR_CHECK(esp_isp_ccm_enable(isp_proc));
// CCM can also be configured after it is enabled
ccm_cfg.matrix[0][0] = 2.0;
ESP_ERROR_CHECK(esp_isp_ccm_configure(isp_proc, &ccm_cfg));
// Disable CCM if no longer needed
ESP_ERROR_CHECK(esp_isp_ccm_disable(isp_proc));
```

Register Event Callbacks

Register ISP AF Environment Detector Event Callbacks After the ISP AF environment detector starts up, it can generate a specific event dynamically. If you have some functions that should be called when the event happens, please hook your function to the interrupt service routine by calling *esp_isp_af_env_detector_register_event_callbacks()*. All supported event callbacks are listed in *esp_isp_af_env_detector_evt_cbs_t*:

- *esp_isp_af_env_detector_evt_cbs_t::on_env_statistics_done* sets a callback function for environment statistics done. As this function is called within the ISR context, you must ensure that the function does not attempt to block (e.g., by making sure that only FreeRTOS APIs with `ISR` suffix are called from within the function). The function prototype is declared in *esp_isp_af_env_detector_callback_t*.
- *esp_isp_af_env_detector_evt_cbs_t::on_env_change* sets a callback function for environment change. As this function is called within the ISR context, you must ensure that the function does not attempt to block (e.g., by making sure that only FreeRTOS APIs with `ISR` suffix are called from within the function). The function prototype is declared in *esp_isp_af_env_detector_callback_t*.

You can save your own context to *esp_isp_af_env_detector_register_event_callbacks()* as well, via the parameter `user_data`. The user data will be directly passed to the callback function.

Register ISP AWB Statistics Done Event Callbacks After the ISP AWB controller finished statistics of white patches, it can generate a specific event dynamically. If you want to be informed when the statistics done event takes place, please hook your function to the interrupt service routine by calling `esp_isp_awb_register_event_callbacks()`. All supported event callbacks are listed in `esp_isp_awb_cbs_t`:

- `esp_isp_awb_cbs_t::on_statistics_done` sets a callback function when finished statistics of the white patches. As this function is called within the ISR context, you must ensure that the function does not attempt to block (e.g., by making sure that only FreeRTOS APIs with `ISR` suffix are called from within the function). The function prototype is declared in `esp_isp_awb_callback_t`.

You can save your own context via the parameter `user_data` of `esp_isp_awb_register_event_callbacks()`. The user data will be directly passed to the callback function.

Thread Safety The factory function `esp_isp_new_processor()`, `esp_isp_del_processor()`, *esp_isp_new_af_controller()*, *esp_isp_del_af_controller()*,

`esp_isp_new_af_env_detector()`, and `esp_isp_del_af_env_detector()` are guaranteed to be thread safe by the driver, which means, user can call them from different RTOS tasks without protection by extra locks.

Kconfig Options

- *CONFIG_ISP_ISR_IRAM_SAFE* controls whether the default ISR handler should be masked when the cache is disabled

IRAM Safe By default, the ISP interrupt will be deferred when the cache is disabled because of writing or erasing the flash.

There is a Kconfig option *CONFIG_ISP_ISR_IRAM_SAFE* that:

- Enables the interrupt being serviced even when the cache is disabled
- Places all functions that used by the ISR into IRAM
- Places driver object into DRAM (in case it is mapped to PSRAM by accident)

This allows the interrupt to run while the cache is disabled, but comes at the cost of increased IRAM consumption.

API Reference

Header File

- components/esp_driver_isp/include/driver/isp.h
- This header file can be included with:

```
#include "driver/isp.h"
```

- This header file is a part of the API provided by the `esp_driver_isp` component. To declare that your component depends on `esp_driver_isp`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_isp
```

or

```
PRIV_REQUIRES esp_driver_isp
```

Header File

- components/esp_driver_isp/include/driver/isp_types.h
- This header file can be included with:

```
#include "driver/isp_types.h"
```

- This header file is a part of the API provided by the `esp_driver_isp` component. To declare that your component depends on `esp_driver_isp`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_isp
```

or

```
PRIV_REQUIRES esp_driver_isp
```

Structures

struct **isp_u32_range_t**

ISP unsigned integer range type.

备注: Whether the edge value are included depends on the variable itself

Public Members

uint32_t **min**

Minimum unsigned int value.

uint32_t **max**

Maximum unsigned int value.

struct **isp_float_range_t**

ISP float range type.

备注: Whether the edge value are included depends on the variable itself

Public Members

float **min**

Minimum float value.

float **max**

Maximum float value.

struct **isp_af_result_t**

ISP AF result.

Public Members

int **definition**[ISP_AF_WINDOW_NUM]

Definition, it refers how clear and sharp an image is.

int **luminance**[ISP_AF_WINDOW_NUM]

Luminance, it refers how luminant an image is.

struct **isp_awb_stat_result_t**

ISP AWB result.

Public Members

uint32_t **white_patch_num**

white patch number that counted by AWB in the window

uint32_t **sum_r**

The sum of R channel of these white patches.

uint32_t **sum_g**

The sum of G channel of these white patches.

uint32_t **sum_b**

The sum of B channel of these white patches.

Type Definitions

typedef struct isp_processor_t ***isp_proc_handle_t**

Type of ISP processor handle.

typedef struct isp_af_controller_t ***isp_af_ctlr_t**

Type of ISP AF controller handle.

typedef struct isp_awb_controller_t ***isp_awb_ctlr_t**

Type of ISP AWB controller handle.

Header File

- components/esp_driver_isp/include/driver/isp_af.h
- This header file can be included with:

```
#include "driver/isp_af.h"
```

- This header file is a part of the API provided by the `esp_driver_isp` component. To declare that your component depends on `esp_driver_isp`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_isp
```

or

```
PRIV_REQUIRES esp_driver_isp
```

Functions

esp_err_t **esp_isp_new_af_controller**(*isp_proc_handle_t* isp_proc, const *esp_isp_af_config_t* *af_config, *isp_af_ctlr_t* *ret_hdl)

New an ISP AF controller.

参数

- **isp_proc** -- **[in]** ISP Processor handle
- **af_config** -- **[in]** Pointer to AF config. Refer to *esp_isp_af_config_t*.
- **ret_hdl** -- **[out]** AF controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG If the combination of arguments is invalid
- ESP_ERR_INVALID_STATE Invalid state
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags
- ESP_ERR_NO_MEM If out of memory

esp_err_t **esp_isp_del_af_controller**(*isp_af_ctlr_t* af_ctrlr)

Delete an ISP AF controller.

参数 **af_ctrlr** -- **[in]** AF controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG If the combination of arguments is invalid.
- ESP_ERR_INVALID_STATE Driver state is invalid.

esp_err_t **esp_isp_af_controller_enable**(*isp_af_ctlr_t* af_ctrlr)

Enable an ISP AF controller.

参数 **af_ctrlr** -- **[in]** AF controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG If the combination of arguments is invalid.
- ESP_ERR_INVALID_STATE Driver state is invalid.

esp_err_t **esp_isp_af_controller_disable**(*isp_af_ctlr_t* af_ctrlr)

Disable an ISP AF controller.

参数 **af_ctrlr** -- **[in]** AF controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG If the combination of arguments is invalid.
- ESP_ERR_INVALID_STATE Driver state is invalid.

esp_err_t **esp_isp_af_controller_get_oneshot_statistics**(*isp_af_ctlr_t* af_ctrlr, int timeout_ms, *isp_af_result_t* *out_res)

Trigger AF luminance and definition statistics for one time and get the result.

备注: This function is a synchronous and block function, it only returns when AF luminance and definition statistics is done or timeout. It's a simple method to get the result directly for one time.

参数

- **af_ctrlr** -- **[in]** AF controller handle
- **timeout_ms** -- **[in]** Timeout in millisecond
  - timeout_ms < 0: Won't return until finished
  - timeout_ms = 0: No timeout, trigger one time statistics and return immediately, in this case, the result won't be assigned in this function, but you can get the result in the callback *esp_isp_af_env_detector_evt_cbs_t::on_env_statistics_done*
  - timeout_ms > 0: Wait for specified milliseconds, if not finished, then return timeout error
- **out_res** -- **[out]** AF luminance and definition statistics result, can be NULL if `timeout_ms = 0`

返回

- ESP_OK On success
- ESP_ERR_TIMEOUT If the waiting time exceeds the specified timeout.
- ESP_ERR_INVALID_ARG If the combination of arguments is invalid.
- ESP_ERR_INVALID_STATE Driver state is invalid.

esp_err_t **esp_isp_af_controller_start_continuous_statistics**(*isp_af_ctlr_t* af_ctrlr)

Start AF continuous statistics of the luminance and definition in the windows.

备注: This function is an asynchronous and non-block function, it will start the continuous statistics and return immediately. You have to register the AF callback and get the result from the callback event data.

参数 **af_ctrlr** -- **[in]** AF controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Null pointer
- ESP_ERR_INVALID_STATE Driver state is invalid.

esp_err_t **esp_isp_af_controller_stop_continuous_statistics**(*isp_af_ctlr_t* af_ctrlr)

Stop AF continuous statistics of the luminance and definition in the windows.

参数 **af_ctrlr** -- **[in]** AF controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Null pointer
- ESP_ERR_INVALID_STATE Driver state is invalid.

esp_err_t **esp_isp_af_controller_set_env_detector**(*isp_af_ctlr_t* af_ctrlr, const *esp_isp_af_env_config_t* *env_config)

Set ISP AF environment detector.

参数

- **af_ctrlr** -- **[in]** AF controller handle
- **env_config** -- **[in]** AF Env detector configuration

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG If the combination of arguments is invalid.
- ESP_ERR_INVALID_STATE Driver state is invalid.

esp_err_t **esp_isp_af_controller_set_env_detector_threshold**(*isp_af_ctlr_t* af_ctrlr, int definition_thresh, int luminance_thresh)

Set ISP AF environment detector detecting threshold.

参数

- **af_ctrlr** -- **[in]** AF controller handle
- **definition_thresh** -- **[in]** Threshold for definition
- **luminance_thresh** -- **[in]** Threshold for luminance

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG If the combination of arguments is invalid.
- ESP_ERR_INVALID_STATE Driver state is invalid.

esp_err_t **esp_isp_af_env_detector_register_event_callbacks**(*isp_af_ctlr_t* af_ctrlr, const *esp_isp_af_env_detector_evt_cbs_t* *cbs, void *user_data)

Register AF environment detector event callbacks.

备注: User can deregister a previously registered callback by calling this function and setting the to-be-deregistered callback member in the `cbs` structure to NULL.

备注: When CONFIG_ISP_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. Involved variables (including `user_data`) should be in internal RAM as well.

参数

- **af_ctrlr** -- **[in]** AF controller handle
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be delivered to the callback functions directly

返回

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_INVALID_STATE: Driver state is invalid, you shouldn't call this API at this moment

Structures

struct **esp_isp_af_config_t**

AF controller config.

Public Members

isp_window_t **window**[ISP_AF_WINDOW_NUM]

The sampling windows of AF.

int **edge_thresh**

Edge threshold, definition higher than this value will be counted as a valid pixel for calculating AF result.

int **intr_priority**

The interrupt priority, range 0~7, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3) otherwise the larger the higher, 7 is NMI.

struct **esp_isp_af_env_config_t**

AF environment detector config.

Public Members

int **interval**

Interval between environment detection, in frames. i.e., AF controller will trigger the statistic periodically to detect the environment change.

struct **esp_isp_af_env_detector_evt_data_t**

Event data structure.

Public Members

isp_af_result_t **af_result**

The AF statistics result

struct **esp_isp_af_env_detector_evt_cbs_t**

Group of ISP AF Env detector callbacks.

备注： These callbacks are all running in an ISR environment.

备注： When CONFIG_ISP_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. Involved variables should be in internal RAM as well.

Public Members

esp_isp_af_env_detector_callback_t **on_env_statistics_done**

Event callback, invoked when environment sample done.

esp_isp_af_env_detector_callback_t **on_env_change**

Event callback, invoked when environment change happens.

Type Definitions

typedef bool (***esp_isp_af_env_detector_callback_t***)(*isp_af_ctlr_t* af_ctrlr, const *esp_isp_af_env_detector_evt_data_t* *edata, void *user_data)

Prototype of ISP AF Env detector event callback.

Param af_ctrlr [in] ISP AF controller handle
Param edata [in] ISP AF Env detector event data
Param user_data [in] User registered context, registered when in `esp_isp_af_env_detector_register_event_callbacks()`
Return Whether a high priority task is woken up by this function

2.5.15 LCD

Introduction

ESP chips can generate various kinds of timings needed by common LCDs on the market, like SPI LCD, I2C LCD, Parallel LCD (Intel 8080), RGB/SRGB LCD, MIPI DSI LCD and etc. The `esp_lcd` component offers an abstracted driver framework to support them in a unified way.

An LCD typically consists of two main planes:

- **Control Plane**: This plane allows us to read and write to the internal registers of the LCD device controller. Host typically uses this plane for tasks such as initializing the LCD power supply and performing gamma calibration.
- **Data Plane**: The data plane is responsible for transmitting pixel data to the LCD device.

Functional Overview

In the context of `esp_lcd`, both the data plane and the control plane are represented by the *`esp_lcd_panel_handle_t`* type.

On some LCDs, these two planes may be combined into a single plane. In this configuration, pixel data is transmitted through the control plane, achieving functionality similar to that of the data plane. This merging is common in SPI LCDs and I2C LCDs. Additionally, there are LCDs that do not require a separate control plane. For instance, certain RGB LCDs automatically execute necessary initialization procedures after power-up. Host devices only need to continuously refresh pixel data through the data plane. However, it's essential to note that not all RGB LCDs eliminate the control plane entirely. Some LCD devices can simultaneously support multiple interfaces, requiring the host to send specific commands via the control plane (such as those based on the SPI interface) to enable the RGB mode.

This document will discuss how to create the control plane and data plane, as mentioned earlier, based on different types of LCDs.

SPI Interfaced LCD

1. Create an SPI bus. Please refer to *SPI Master API doc* for more details.
   Currently the driver supports SPI, Quad SPI and Octal SPI (simulate Intel 8080 timing) modes.

```
spi_bus_config_t buscfg = {
    .sclk_io_num = EXAMPLE_PIN_NUM_SCLK,
    .mosi_io_num = EXAMPLE_PIN_NUM_MOSI,
    .miso_io_num = EXAMPLE_PIN_NUM_MISO,
    .quadwp_io_num = -1,
```

(下页继续)

(续上页)

```
    .quadhd_io_num = -1,
    .max_transfer_sz = EXAMPLE_LCD_H_RES * 80 * sizeof(uint16_t), //␣
↪transfer 80 lines of pixels (assume pixel is RGB565) at most in one␣
↪SPI transaction
};
ESP_ERROR_CHECK(spi_bus_initialize(LCD_HOST, &buscfg, SPI_DMA_CH_
↪AUTO)); // Enable the DMA feature
```

2. Allocate an LCD IO device handle from the SPI bus. In this step, you need to provide the following information:
   - *esp_lcd_panel_io_spi_config_t::dc_gpio_num*: Sets the gpio number for the DC signal line (some LCD calls this `RS` line). The LCD driver uses this GPIO to switch between sending command and sending data.
   - *esp_lcd_panel_io_spi_config_t::cs_gpio_num*: Sets the gpio number for the CS signal line. The LCD driver uses this GPIO to select the LCD chip. If the SPI bus only has one device attached (i.e., this LCD), you can set the gpio number to `-1` to occupy the bus exclusively.
   - *esp_lcd_panel_io_spi_config_t::pclk_hz* sets the frequency of the pixel clock, in Hz. The value should not exceed the range recommended in the LCD spec.
   - *esp_lcd_panel_io_spi_config_t::spi_mode* sets the SPI mode. The LCD driver uses this mode to communicate with the LCD. For the meaning of the SPI mode, please refer to the *SPI Master API doc*.
   - *esp_lcd_panel_io_spi_config_t::lcd_cmd_bits* and *esp_lcd_panel_io_spi_config_t::lcd_param_bits* set the bit width of the command and parameter that recognized by the LCD controller chip. This is chip specific, you should refer to your LCD spec in advance.
   - *esp_lcd_panel_io_spi_config_t::trans_queue_depth* sets the depth of the SPI transaction queue. A bigger value means more transactions can be queued up, but it also consumes more memory.

```
esp_lcd_panel_io_handle_t io_handle = NULL;
esp_lcd_panel_io_spi_config_t io_config = {
    .dc_gpio_num = EXAMPLE_PIN_NUM_LCD_DC,
    .cs_gpio_num = EXAMPLE_PIN_NUM_LCD_CS,
    .pclk_hz = EXAMPLE_LCD_PIXEL_CLOCK_HZ,
    .lcd_cmd_bits = EXAMPLE_LCD_CMD_BITS,
    .lcd_param_bits = EXAMPLE_LCD_PARAM_BITS,
    .spi_mode = 0,
    .trans_queue_depth = 10,
};
// Attach the LCD to the SPI bus
ESP_ERROR_CHECK(esp_lcd_new_panel_io_spi((esp_lcd_spi_bus_handle_t)LCD_
↪HOST, &io_config, &io_handle));
```

3. Install the LCD controller driver. The LCD controller driver is responsible for sending the commands and parameters to the LCD controller chip. In this step, you need to specify the SPI IO device handle that allocated in the last step, and some panel specific configurations:
   - `esp_lcd_panel_dev_config_t::reset_gpio_num` sets the LCD's hardware reset GPIO number. If the LCD does not have a hardware reset pin, set this to `-1`.
   - `esp_lcd_panel_dev_config_t::rgb_ele_order` sets the R-G-B element order of each color data.
   - `esp_lcd_panel_dev_config_t::bits_per_pixel` sets the bit width of the pixel color data. The LCD driver uses this value to calculate the number of bytes to send to the LCD controller chip.
   - `esp_lcd_panel_dev_config_t::data_endian` specifies the data endian to be transmitted to the screen. No need to specify for color data within 1 byte, like RGB232. For drivers that do not support specifying data endian, this field would be ignored.

```
esp_lcd_panel_handle_t panel_handle = NULL;
esp_lcd_panel_dev_config_t panel_config = {
```

(下页继续)

(续上页)

```
    .reset_gpio_num = EXAMPLE_PIN_NUM_RST,
    .rgb_ele_order = LCD_RGB_ELEMENT_ORDER_BGR,
    .bits_per_pixel = 16,
};
// Create LCD panel handle for ST7789, with the SPI IO device handle
ESP_ERROR_CHECK(esp_lcd_new_panel_st7789(io_handle, &panel_config, &
↪panel_handle));
```

API Reference

Header File

- components/esp_lcd/include/esp_lcd_io_spi.h
- This header file can be included with:

```
#include "esp_lcd_io_spi.h"
```

- This header file is a part of the API provided by the `esp_lcd` component. To declare that your component depends on `esp_lcd`, add the following to your CMakeLists.txt:

```
REQUIRES esp_lcd
```

or

```
PRIV_REQUIRES esp_lcd
```

Functions

esp_err_t **esp_lcd_new_panel_io_spi**(*esp_lcd_spi_bus_handle_t* bus, const *esp_lcd_panel_io_spi_config_t* *io_config, *esp_lcd_panel_io_handle_t* *ret_io)

Create LCD panel IO handle, for SPI interface.

参数

- **bus** -- **[in]** SPI bus handle
- **io_config** -- **[in]** IO configuration, for SPI interface
- **ret_io** -- **[out]** Returned IO handle

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

Structures

struct **esp_lcd_panel_io_spi_config_t**

Panel IO configuration structure, for SPI interface.

Public Members

int **cs_gpio_num**

GPIO used for CS line

int **dc_gpio_num**

GPIO used to select the D/C line, set this to -1 if the D/C line is not used

int **spi_mode**

Traditional SPI mode (0~3)

unsigned int **pclk_hz**

Frequency of pixel clock

size_t **trans_queue_depth**

Size of internal transaction queue

esp_lcd_panel_io_color_trans_done_cb_t **on_color_trans_done**

Callback invoked when color data transfer has finished

void ***user_ctx**

User private data, passed directly to on_color_trans_done's user_ctx

int **lcd_cmd_bits**

Bit-width of LCD command

int **lcd_param_bits**

Bit-width of LCD parameter

unsigned int **dc_high_on_cmd**

If enabled, DC level = 1 indicates command transfer

unsigned int **dc_low_on_data**

If enabled, DC level = 0 indicates color data transfer

unsigned int **dc_low_on_param**

If enabled, DC level = 0 indicates parameter transfer

unsigned int **octal_mode**

transmit with octal mode (8 data lines), this mode is used to simulate Intel 8080 timing

unsigned int **quad_mode**

transmit with quad mode (4 data lines), this mode is useful when transmitting LCD parameters (Only use one line for command)

unsigned int **sio_mode**

Read and write through a single data line (MOSI)

unsigned int **lsb_first**

transmit LSB bit first

unsigned int **cs_high_active**

CS line is high active

struct *esp_lcd_panel_io_spi_config_t*::[anonymous] **flags**

Extra flags to fine-tune the SPI device

Type Definitions

typedef int **esp_lcd_spi_bus_handle_t**

Type of LCD SPI bus handle

I2C Interfaced LCD

1. Create I2C bus. Please refer to *I2C API doc* for more details.

```
i2c_master_bus_handle_t i2c_bus = NULL;
i2c_master_bus_config_t bus_config = {
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .glitch_ignore_cnt = 7,
    .i2c_port = I2C_BUS_PORT,
    .sda_io_num = EXAMPLE_PIN_NUM_SDA,
    .scl_io_num = EXAMPLE_PIN_NUM_SCL,
    .flags.enable_internal_pullup = true,
};
ESP_ERROR_CHECK(i2c_new_master_bus(&bus_config, &i2c_bus));
```

2. Allocate an LCD IO device handle from the I2C bus. In this step, you need to provide the following information:
   - *esp_lcd_panel_io_i2c_config_t::dev_addr* sets the I2C device address of the LCD controller chip. The LCD driver uses this address to communicate with the LCD controller chip.
   - *esp_lcd_panel_io_i2c_config_t::scl_speed_hz* sets the I2C clock frequency in Hz. The value should not exceed the range recommended in the LCD spec.
   - *esp_lcd_panel_io_i2c_config_t::lcd_cmd_bits* and *esp_lcd_panel_io_i2c_config_t::lcd_param_bits* set the bit width of the command and parameter that recognized by the LCD controller chip. This is chip specific, you should refer to your LCD spec in advance.

```
esp_lcd_panel_io_handle_t io_handle = NULL;
esp_lcd_panel_io_i2c_config_t io_config = {
    .dev_addr = EXAMPLE_I2C_HW_ADDR,
    .scl_speed_hz = EXAMPLE_LCD_PIXEL_CLOCK_HZ,
    .control_phase_bytes = 1, // refer to LCD spec
    .dc_bit_offset = 6,       // refer to LCD spec
    .lcd_cmd_bits = EXAMPLE_LCD_CMD_BITS,
    .lcd_param_bits = EXAMPLE_LCD_CMD_BITS,
};
ESP_ERROR_CHECK(esp_lcd_new_panel_io_i2c(i2c_bus, &io_config, &io_
↪handle));
```

3. Install the LCD controller driver. The LCD controller driver is responsible for sending the commands and parameters to the LCD controller chip. In this step, you need to specify the I2C IO device handle that allocated in the last step, and some panel specific configurations:
   - `esp_lcd_panel_dev_config_t::reset_gpio_num` sets the LCD's hardware reset GPIO number. If the LCD does not have a hardware reset pin, set this to `-1`.
   - `esp_lcd_panel_dev_config_t::bits_per_pixel` sets the bit width of the pixel color data. The LCD driver uses this value to calculate the number of bytes to send to the LCD controller chip.

```
esp_lcd_panel_handle_t panel_handle = NULL;
esp_lcd_panel_dev_config_t panel_config = {
    .bits_per_pixel = 1,
    .reset_gpio_num = EXAMPLE_PIN_NUM_RST,
};
ESP_ERROR_CHECK(esp_lcd_new_panel_ssd1306(io_handle, &panel_config, &
↪panel_handle));
```

API Reference

Header File

- components/esp_lcd/include/esp_lcd_io_i2c.h
- This header file can be included with:

```
#include "esp_lcd_io_i2c.h"
```

- This header file is a part of the API provided by the `esp_lcd` component. To declare that your component depends on `esp_lcd`, add the following to your CMakeLists.txt:

```
REQUIRES esp_lcd
```

or

```
PRIV_REQUIRES esp_lcd
```

Functions

esp_err_t **esp_lcd_new_panel_io_i2c_v1**(uint32_t bus, const *esp_lcd_panel_io_i2c_config_t* *io_config, *esp_lcd_panel_io_handle_t* *ret_io)

Create LCD panel IO handle, for I2C interface in legacy implementation.

备注: Please don't call this function in your project directly. Please call `esp_lcd_new_panel_to_i2c` instead.

参数

- **bus** -- **[in]** I2C bus handle, (in uint32_t)
- **io_config** -- **[in]** IO configuration, for I2C interface
- **ret_io** -- **[out]** Returned IO handle

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

esp_err_t **esp_lcd_new_panel_io_i2c_v2**(*i2c_master_bus_handle_t* bus, const *esp_lcd_panel_io_i2c_config_t* *io_config, *esp_lcd_panel_io_handle_t* *ret_io)

Create LCD panel IO handle, for I2C interface in new implementation.

备注: Please don't call this function in your project directly. Please call `esp_lcd_new_panel_to_i2c` instead.

参数

- **bus** -- **[in]** I2C bus handle, (in i2c_master_dev_handle_t)
- **io_config** -- **[in]** IO configuration, for I2C interface
- **ret_io** -- **[out]** Returned IO handle

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

Structures

struct **esp_lcd_panel_io_i2c_config_t**

Panel IO configuration structure, for I2C interface.

Public Members

uint32_t **dev_addr**

I2C device address

esp_lcd_panel_io_color_trans_done_cb_t **on_color_trans_done**

Callback invoked when color data transfer has finished

void ***user_ctx**

User private data, passed directly to on_color_trans_done's user_ctx

size_t **control_phase_bytes**

I2C LCD panel will encode control information (e.g. D/C selection) into control phase, in several bytes

unsigned int **dc_bit_offset**

Offset of the D/C selection bit in control phase

int **lcd_cmd_bits**

Bit-width of LCD command

int **lcd_param_bits**

Bit-width of LCD parameter

unsigned int **dc_low_on_data**

If this flag is enabled, DC line = 0 means transfer data, DC line = 1 means transfer command; vice versa

unsigned int **disable_control_phase**

If this flag is enabled, the control phase isn't used

struct *esp_lcd_panel_io_i2c_config_t*::[anonymous] **flags**

Extra flags to fine-tune the I2C device

uint32_t **scl_speed_hz**

I2C LCD SCL frequency (hz)

Macros

esp_lcd_new_panel_io_i2c(bus, io_config, ret_io)

Create LCD panel IO handle.

参数

- **bus** -- **[in]** I2C bus handle
- **io_config** -- **[in]** IO configuration, for I2C interface
- **ret_io** -- **[out]** Returned IO handle

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

Type Definitions

typedef uint32_t **esp_lcd_i2c_bus_handle_t**

Type of LCD I2C bus handle

I80 Interfaced LCD

1. Create I80 bus by *esp_lcd_new_i80_bus()*. You need to set up the following parameters for an Intel 8080 parallel bus:
   - *esp_lcd_i80_bus_config_t::clk_src* sets the clock source of the I80 bus. Note, the default clock source may be different between ESP targets.
   - *esp_lcd_i80_bus_config_t::wr_gpio_num* sets the GPIO number of the pixel clock (also referred as `WR` in some LCD spec)
   - *esp_lcd_i80_bus_config_t::dc_gpio_num* sets the GPIO number of the data/command select pin (also referred as `RS` in some LCD spec)
   - *esp_lcd_i80_bus_config_t::bus_width* sets the bit width of the data bus (only support `8` or `16`)
   - *esp_lcd_i80_bus_config_t::data_gpio_nums* is the array of the GPIO number of the data bus. The number of GPIOs should be equal to the *esp_lcd_i80_bus_config_t::bus_width* value.
   - *esp_lcd_i80_bus_config_t::max_transfer_bytes* sets the maximum number of bytes that can be transferred in one transaction.

```
esp_lcd_i80_bus_handle_t i80_bus = NULL;
esp_lcd_i80_bus_config_t bus_config = {
    .clk_src = LCD_CLK_SRC_DEFAULT,
    .dc_gpio_num = EXAMPLE_PIN_NUM_DC,
    .wr_gpio_num = EXAMPLE_PIN_NUM_PCLK,
    .data_gpio_nums = {
        EXAMPLE_PIN_NUM_DATA0,
        EXAMPLE_PIN_NUM_DATA1,
        EXAMPLE_PIN_NUM_DATA2,
        EXAMPLE_PIN_NUM_DATA3,
        EXAMPLE_PIN_NUM_DATA4,
        EXAMPLE_PIN_NUM_DATA5,
        EXAMPLE_PIN_NUM_DATA6,
        EXAMPLE_PIN_NUM_DATA7,
    },
    .bus_width = 8,
    .max_transfer_bytes = EXAMPLE_LCD_H_RES * 100 * sizeof(uint16_t), /
↪/ transfer 100 lines of pixels (assume pixel is RGB565) at most in␣
↪one transaction
    .dma_burst_size = EXAMPLE_DMA_BURST_SIZE,
};
ESP_ERROR_CHECK(esp_lcd_new_i80_bus(&bus_config, &i80_bus));
```

2. Allocate an LCD IO device handle from the I80 bus. In this step, you need to provide the following information:
   - *esp_lcd_panel_io_i80_config_t::cs_gpio_num* sets the GPIO number of the chip select pin.
   - *esp_lcd_panel_io_i80_config_t::pclk_hz* sets the pixel clock frequency in Hz. Higher pixel clock frequency results in higher refresh rate, but may cause flickering if the DMA bandwidth is not sufficient or the LCD controller chip does not support high pixel clock frequency.
   - *esp_lcd_panel_io_i80_config_t::lcd_cmd_bits* and *esp_lcd_panel_io_i80_config_t::lcd_param_bits* set the bit width of the command and parameter that recognized by the LCD controller chip. This is chip specific, you should refer to your LCD spec in advance.
   - *esp_lcd_panel_io_i80_config_t::trans_queue_depth* sets the maximum number of transactions that can be queued in the LCD IO device. A bigger value means more transactions can be queued up, but it also consumes more memory.

```
esp_lcd_panel_io_handle_t io_handle = NULL;
esp_lcd_panel_io_i80_config_t io_config = {
    .cs_gpio_num = EXAMPLE_PIN_NUM_CS,
    .pclk_hz = EXAMPLE_LCD_PIXEL_CLOCK_HZ,
    .trans_queue_depth = 10,
```

(下页继续)

(续上页)

```
    .dc_levels = {
        .dc_idle_level = 0,
        .dc_cmd_level = 0,
        .dc_dummy_level = 0,
        .dc_data_level = 1,
    },
    .lcd_cmd_bits = EXAMPLE_LCD_CMD_BITS,
    .lcd_param_bits = EXAMPLE_LCD_PARAM_BITS,
};
ESP_ERROR_CHECK(esp_lcd_new_panel_io_i80(i80_bus, &io_config, &io_
↪handle));
```

3. Install the LCD controller driver. The LCD controller driver is responsible for sending the commands and parameters to the LCD controller chip. In this step, you need to specify the I80 IO device handle that allocated in the last step, and some panel specific configurations:
    - `esp_lcd_panel_dev_config_t::bits_per_pixel` sets the bit width of the pixel color data. The LCD driver uses this value to calculate the number of bytes to send to the LCD controller chip.
    - `esp_lcd_panel_dev_config_t::reset_gpio_num` sets the GPIO number of the reset pin. If the LCD controller chip does not have a reset pin, you can set this value to `-1`.
    - `esp_lcd_panel_dev_config_t::rgb_ele_order` sets the color order the pixel color data.

```
esp_lcd_panel_dev_config_t panel_config = {
    .reset_gpio_num = EXAMPLE_PIN_NUM_RST,
    .rgb_ele_order = LCD_RGB_ELEMENT_ORDER_RGB,
    .bits_per_pixel = 16,
};
ESP_ERROR_CHECK(esp_lcd_new_panel_st7789(io_handle, &panel_config, &
↪panel_handle));
```

API Reference

Header File

- components/esp_lcd/include/esp_lcd_io_i80.h
- This header file can be included with:

```
#include "esp_lcd_io_i80.h"
```

- This header file is a part of the API provided by the `esp_lcd` component. To declare that your component depends on `esp_lcd`, add the following to your CMakeLists.txt:

```
REQUIRES esp_lcd
```

or

```
PRIV_REQUIRES esp_lcd
```

Functions

esp_err_t **esp_lcd_new_i80_bus**(const *esp_lcd_i80_bus_config_t* *bus_config, *esp_lcd_i80_bus_handle_t* *ret_bus)

Create Intel 8080 bus handle.

参数

- **bus_config** -- **[in]** Bus configuration
- **ret_bus** -- **[out]** Returned bus handle

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_NOT_FOUND if no free bus is available
- ESP_OK on success

esp_err_t **esp_lcd_del_i80_bus**(*esp_lcd_i80_bus_handle_t* bus)

Destroy Intel 8080 bus handle.

参数 **bus** -- **[in]** Intel 8080 bus handle, created by `esp_lcd_new_i80_bus()`

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_INVALID_STATE if there still be some device attached to the bus
- ESP_OK on success

esp_err_t **esp_lcd_new_panel_io_i80**(*esp_lcd_i80_bus_handle_t* bus, const *esp_lcd_panel_io_i80_config_t* *io_config, *esp_lcd_panel_io_handle_t* *ret_io)

Create LCD panel IO, for Intel 8080 interface.

参数

- **bus** -- **[in]** Intel 8080 bus handle, created by `esp_lcd_new_i80_bus()`
- **io_config** -- **[in]** IO configuration, for i80 interface
- **ret_io** -- **[out]** Returned panel IO handle

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_SUPPORTED if some configuration can't be satisfied, e.g. pixel clock out of the range
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

void ***esp_lcd_i80_alloc_draw_buffer**(*esp_lcd_panel_io_handle_t* io, size_t size, uint32_t caps)

Allocate a draw buffer that can be used by I80 interfaced LCD panel.

备注: This function differs from the normal 'heap_caps_*' functions in that it can also automatically handle the alignment required by DMA burst, cache line size, etc.

参数

- **io** -- **[in]** Panel IO handle, created by `esp_lcd_new_panel_io_i80()`
- **size** -- **[in]** Size of memory to be allocated
- **caps** -- **[in]** Bitwise OR of MALLOC_CAP_* flags indicating the type of memory desired for the allocation

返回 Pointer to a new buffer of size 'size' with capabilities 'caps', or NULL if allocation failed

Structures

struct **esp_lcd_i80_bus_config_t**

LCD Intel 8080 bus configuration structure.

Public Members

int **dc_gpio_num**

GPIO used for D/C line

int **wr_gpio_num**

GPIO used for WR line

lcd_clock_source_t **clk_src**

Clock source for the I80 LCD peripheral

int **data_gpio_nums**[ESP_LCD_I80_BUS_WIDTH_MAX]

GPIOs used for data lines

size_t **bus_width**

Number of data lines, 8 or 16

size_t **max_transfer_bytes**

Maximum transfer size, this determines the length of internal DMA link

size_t **psram_trans_align**

DMA transfer alignment for data allocated from PSRAM

size_t **dma_burst_size**

DMA burst size, in bytes

size_t **sram_trans_align**

DMA transfer alignment for data allocated from SRAM

struct **esp_lcd_panel_io_i80_config_t**

Panel IO configuration structure, for intel 8080 interface.

Public Members

int **cs_gpio_num**

GPIO used for CS line, set to -1 will declaim exclusively use of I80 bus

uint32_t **pclk_hz**

Frequency of pixel clock

size_t **trans_queue_depth**

Transaction queue size, larger queue, higher throughput

esp_lcd_panel_io_color_trans_done_cb_t **on_color_trans_done**

Callback invoked when color data was transferred done

void ***user_ctx**

User private data, passed directly to on_color_trans_done's user_ctx

int **lcd_cmd_bits**

Bit-width of LCD command

int **lcd_param_bits**

Bit-width of LCD parameter

unsigned int **dc_idle_level**

Level of DC line in IDLE phase

unsigned int **dc_cmd_level**

Level of DC line in CMD phase

unsigned int **dc_dummy_level**

Level of DC line in DUMMY phase

unsigned int **dc_data_level**

Level of DC line in DATA phase

struct *esp_lcd_panel_io_i80_config_t*::[anonymous] **dc_levels**

Each i80 device might have its own D/C control logic

unsigned int **cs_active_high**

If set, a high level of CS line will select the device, otherwise, CS line is low level active

unsigned int **reverse_color_bits**

Reverse the data bits, D[N:0] -> D[0:N]

unsigned int **swap_color_bytes**

Swap adjacent two color bytes

unsigned int **pclk_active_neg**

The display will write data lines when there's a falling edge on WR signal (a.k.a the PCLK)

unsigned int **pclk_idle_low**

The WR signal (a.k.a the PCLK) stays at low level in IDLE phase

struct *esp_lcd_panel_io_i80_config_t*::[anonymous] **flags**

Panel IO config flags

Macros

ESP_LCD_I80_BUS_WIDTH_MAX

Maximum width of I80 bus

Type Definitions

typedef struct esp_lcd_i80_bus_t ***esp_lcd_i80_bus_handle_t**

Type of LCD intel 8080 bus handle

MIPI DSI Interfaced LCD

1. Create a DSI bus, it will initialize the DPHY as well.

```
esp_lcd_dsi_bus_handle_t mipi_dsi_bus = NULL;
esp_lcd_dsi_bus_config_t bus_config = {
    .bus_id = 0, // index from 0, specify the DSI host to use
    .num_data_lanes = 2, // Number of data lanes to use, can't set a␣
↪value that exceeds the chip's capability
    .phy_clk_src = MIPI_DSI_PHY_CLK_SRC_DEFAULT, // Clock source for␣
↪the DPHY
    .lane_bit_rate_mbps = EXAMPLE_MIPI_DSI_LANE_BITRATE_MBPS, // Bit␣
↪rate of the data lanes, in Mbps
};
ESP_ERROR_CHECK(esp_lcd_new_dsi_bus(&bus_config, &mipi_dsi_bus));
```

2. Derive the DBI interface from the DSI bus. The DBI interface mostly is used as the control IO layer in the esp_lcd component. This interface provides the functions to read/write the configuration registers inside the LCD device. In this step, you need to provide the following information:
    - *esp_lcd_dbi_io_config_t::virtual_channel*: The virtual channel number to use. The virtual channel is a logical channel that is used to multiplex the data from different sources. If you only have one LCD connected, you can set this to `0`.
    - *esp_lcd_dbi_io_config_t::lcd_cmd_bits*: The bit width of the command that recognized by the LCD controller chip. This is chip specific, you should refer to your LCD spec in advance.
    - *esp_lcd_dbi_io_config_t::lcd_param_bits*: The bit width of the parameter that recognized by the LCD controller chip. This is chip specific, you should refer to your LCD spec in advance.

```
esp_lcd_panel_io_handle_t mipi_dbi_io = NULL;
esp_lcd_dbi_io_config_t dbi_config = {
    .virtual_channel = 0,
    .lcd_cmd_bits = 8,   // according to the LCD spec
    .lcd_param_bits = 8, // according to the LCD spec
};
ESP_ERROR_CHECK(esp_lcd_new_panel_io_dbi(mipi_dsi_bus, &dbi_config, &
↪mipi_dbi_io));
```

3. Install the LCD controller driver. The LCD controller driver is responsible for sending the commands and parameters to the LCD controller chip. In this step, you need to specify the MIPI DBI IO handle that allocated in the last step, and some panel specific configurations:
    - `esp_lcd_panel_dev_config_t::bits_per_pixel` sets the bit width of each pixel. The LCD driver uses this value to calculate the number of bytes to send to the LCD controller chip.
    - `esp_lcd_panel_dev_config_t::reset_gpio_num` sets the GPIO number of the reset pin. If the LCD controller chip does not have a reset pin, you can set this value to `-1`.
    - `esp_lcd_panel_dev_config_t::rgb_ele_order` set the RGB element order of the pixel data, it can be **RGB** or **BGR**.

```
esp_lcd_panel_handle_t ili9881c_ctrl_panel = NULL;
esp_lcd_panel_dev_config_t lcd_dev_config = {
    .bits_per_pixel = 24, // MIPI LCD usually uses 24bit (i.e.,␣
↪RGB888) per pixel
    .rgb_ele_order = LCD_RGB_ELEMENT_ORDER_RGB,
    .reset_gpio_num = EXAMPLE_PIN_NUM_LCD_RST,
};
ESP_ERROR_CHECK(esp_lcd_new_panel_ili9881c(mipi_dbi_io, &lcd_dev_
↪config, &ili9881c_ctrl_panel));
```

4. With the LCD control panel that returned in the last step, you can reset the LCD device followed by a basic initialization. After that, you can turn on the display.

```
ESP_ERROR_CHECK(esp_lcd_panel_reset(ili9881c_ctrl_panel));
ESP_ERROR_CHECK(esp_lcd_panel_init(ili9881c_ctrl_panel));
ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(ili9881c_ctrl_panel, true));
```

5. However, you still can't send pixel data to the MIPI LCD with the control panel, because MIPI LCD has a

high resolution and there's no GRAM in the LCD controller. We need to maintain the LCD frame buffer and flush it to the LCD via the MIPI DSI DPI interface. To allocate a DPI data panel, you need to provide many essential parameters, including the DPI clock frequency, the pixel format, the video timing, and so on.

- *esp_lcd_dpi_panel_config_t::virtual_channel* Like the DBI interface, we also need to set the virtual channel for the DPI interface. If you only have one LCD connected, you can set this to `0`.
- *esp_lcd_dpi_panel_config_t::dpi_clk_src* sets the clock source for the DPI interface. The available clock sources are listed in `mipi_dsi_dpi_clock_source_t`.
- *esp_lcd_dpi_panel_config_t::dpi_clock_freq_mhz* sets the DPI clock frequency in MHz. Higher pixel clock frequency results in higher refresh rate, but may cause flickering if the DMA bandwidth is not sufficient or the LCD controller chip does not support high pixel clock frequency.
- *esp_lcd_dpi_panel_config_t::in_color_format* sets the pixel format of the input pixel data. The available pixel formats are listed in *lcd_color_format_t*. We usually use **RGB888** for MIPI LCD to get the best color depth.
- *esp_lcd_dpi_panel_config_t::video_timing* sets the LCD panel specific timing parameters. All required parameters are listed in the *esp_lcd_video_timing_t*, including the LCD resolution and blanking porches. Please fill them according to the datasheet of your LCD.
- *esp_lcd_dpi_panel_config_t::extra_dpi_panel_flags::use_dma2d* sets whether to use the 2D DMA peripheral to copy the user data to the frame buffer, asynchronously.

```
esp_lcd_panel_handle_t mipi_dpi_panel = NULL;
esp_lcd_dpi_panel_config_t dpi_config = {
    .virtual_channel = 0,
    .dpi_clk_src = MIPI_DSI_DPI_CLK_SRC_DEFAULT,
    .dpi_clock_freq_mhz = 1 * 1000,
    .in_color_format = LCD_COLOR_FMT_RGB888,
    .video_timing = {
        .h_size = EXAMPLE_MIPI_DSI_LCD_H_RES,
        .v_size = EXAMPLE_MIPI_DSI_LCD_V_RES,
        .hsync_back_porch = EXAMPLE_MIPI_DSI_LCD_HBP,
        .hsync_pulse_width = EXAMPLE_MIPI_DSI_LCD_HSYNC,
        .hsync_front_porch = EXAMPLE_MIPI_DSI_LCD_HFP,
        .vsync_back_porch = EXAMPLE_MIPI_DSI_LCD_VBP,
        .vsync_pulse_width = EXAMPLE_MIPI_DSI_LCD_VSYNC,
        .vsync_front_porch = EXAMPLE_MIPI_DSI_LCD_VFP,
    },
    .flags.use_dma2d = true,
};
ESP_ERROR_CHECK(esp_lcd_new_panel_dpi(mipi_dsi_bus, &dpi_config, &mipi_
↪dpi_panel));
ESP_ERROR_CHECK(esp_lcd_panel_init(mipi_dpi_panel));
```

API Reference

Header File

- components/esp_lcd/dsi/include/esp_lcd_mipi_dsi.h
- This header file can be included with:

```
#include "esp_lcd_mipi_dsi.h"
```

- This header file is a part of the API provided by the `esp_lcd` component. To declare that your component depends on `esp_lcd`, add the following to your CMakeLists.txt:

```
REQUIRES esp_lcd
```

or

```
PRIV_REQUIRES esp_lcd
```

Functions

esp_err_t **esp_lcd_new_dsi_bus**(const *esp_lcd_dsi_bus_config_t* *bus_config, *esp_lcd_dsi_bus_handle_t* *ret_bus)

Create MIPI DSI bus handle.

参数

- **bus_config** -- **[in]** Bus configuration
- **ret_bus** -- **[out]** Returned bus handle

返回

- ESP_OK: Create MIPI DSI bus successfully
- ESP_ERR_INVALID_ARG: Create MIPI DSI bus failed because of invalid argument
- ESP_ERR_NO_MEM: Create MIPI DSI bus failed because of out of memory
- ESP_ERR_NOT_FOUND: Create MIPI DSI bus failed because no more free DSI hardware instance
- ESP_FAIL: Create MIPI DSI bus failed because of other error

esp_err_t **esp_lcd_del_dsi_bus**(*esp_lcd_dsi_bus_handle_t* bus)

Destroy MIPI DSI bus handle.

参数 **bus** -- **[in]** MIPI DSI bus handle, returned from `esp_lcd_new_dsi_bus`

返回

- ESP_OK: Destroy MIPI DSI bus successfully
- ESP_ERR_INVALID_ARG: Destroy MIPI DSI bus failed because of invalid argument
- ESP_FAIL: Destroy MIPI DSI bus failed because of other error

esp_err_t **esp_lcd_new_panel_io_dbi**(*esp_lcd_dsi_bus_handle_t* bus, const *esp_lcd_dbi_io_config_t* *io_config, *esp_lcd_panel_io_handle_t* *ret_io)

Create LCD panel IO, for MIPI DSI DBI interface.

备注: Although we call it "DBI", internally the driver is using a co-called "generic" interface for transmitting/receiving LCD commands and parameters.

参数

- **bus** -- **[in]** MIPI DSI bus handle, returned from `esp_lcd_new_dsi_bus`
- **io_config** -- **[in]** IO configuration
- **ret_io** -- **[out]** Returned panel IO handle

返回

- ESP_OK: Create MIPI DSI command IO successfully
- ESP_ERR_INVALID_ARG: Create MIPI DSI command IO failed because of invalid argument
- ESP_ERR_NO_MEM: Create MIPI DSI command IO failed because of out of memory
- ESP_FAIL: Create MIPI DSI command IO failed because of other error

esp_err_t **esp_lcd_new_panel_dpi**(*esp_lcd_dsi_bus_handle_t* bus, const *esp_lcd_dpi_panel_config_t* *panel_config, *esp_lcd_panel_handle_t* *ret_panel)

Create LCD panel for MIPI DSI DPI interface.

参数

- **bus** -- **[in]** MIPI DSI bus handle, returned from `esp_lcd_new_dsi_bus`
- **panel_config** -- **[in]** DSI data panel configuration
- **ret_panel** -- **[out]** Returned LCD panel handle

返回

- ESP_OK: Create MIPI DSI data panel successfully
- ESP_ERR_INVALID_ARG: Create MIPI DSI data panel failed because of invalid argument

- ESP_ERR_NO_MEM: Create MIPI DSI data panel failed because of out of memory
- ESP_ERR_NOT_SUPPORTED: Create MIPI DSI data panel failed because of unsupported feature
- ESP_FAIL: Create MIPI DSI data panel failed because of other error

esp_err_t **esp_lcd_dpi_panel_get_frame_buffer**(*esp_lcd_panel_handle_t* dpi_panel, uint32_t fb_num, void **fb0, ...)

Get the address of the frame buffer(s) that allocated by the driver.

参数

- **dpi_panel** -- **[in]** MIPI DPI panel handle, returned from esp_lcd_new_panel_dpi()
- **fb_num** -- **[in]** Number of frame buffer(s) to get. This value must be the same as the number of the followed parameters.
- **fb0** -- **[out]** Address of the frame buffer 0 (first frame buffer)
- **...** -- **[out]** List of other frame buffers if any

返回

- ESP_ERR_INVALID_ARG: Get frame buffer address failed because of invalid argument
- ESP_OK: Get frame buffer address successfully

esp_err_t **esp_lcd_dpi_panel_set_pattern**(*esp_lcd_panel_handle_t* dpi_panel, mipi_dsi_pattern_type_t pattern)

Set pre-defined pattern to the screen for testing or debugging purpose.

参数

- **dpi_panel** -- **[in]** MIPI DPI panel handle, returned from esp_lcd_new_panel_dpi()
- **pattern** -- **[in]** Pattern type

返回

- ESP_OK: Set pattern successfully
- ESP_ERR_INVALID_ARG: Set pattern failed because of invalid argument
- ESP_FAIL: Set pattern failed because of other error

esp_err_t **esp_lcd_dpi_panel_set_color_conversion**(*esp_lcd_panel_handle_t* dpi_panel, const *esp_lcd_color_conv_config_t* *config)

Set color conversion configuration for DPI panel.

参数

- **dpi_panel** -- **[in]** MIPI DPI panel handle, returned from esp_lcd_new_panel_dpi()
- **config** -- **[in]** Color conversion configuration

返回

- ESP_OK: Set color conversion configuration successfully
- ESP_ERR_INVALID_ARG: Set color conversion configuration failed because of invalid argument
- ESP_FAIL: Set color conversion configuration failed because of other error

esp_err_t **esp_lcd_dpi_panel_register_event_callbacks**(*esp_lcd_panel_handle_t* dpi_panel, const *esp_lcd_dpi_panel_event_callbacks_t* *cbs, void *user_ctx)

Register LCD DPI panel callbacks.

参数

- **dpi_panel** -- **[in]** LCD DPI panel handle, which is returned from esp_lcd_new_panel_dpi()
- **cbs** -- **[in]** structure with all LCD panel callbacks
- **user_ctx** -- **[in]** User private data, passed directly to callback's user_ctx

返回

- ESP_ERR_INVALID_ARG: Register callbacks failed because of invalid argument
- ESP_OK: Register callbacks successfully

Structures

struct **esp_lcd_dsi_bus_config_t**

MIPI DSI bus configuration structure.

Public Members

int **bus_id**

Select which DSI controller, index from 0

uint8_t **num_data_lanes**

Number of data lanes, if set to 0, the driver will fallback to use maximum number of lanes

mipi_dsi_phy_clock_source_t **phy_clk_src**

MIPI DSI PHY clock source

uint32_t **lane_bit_rate_mbps**

Lane bit rate in Mbps

struct **esp_lcd_dbi_io_config_t**

Panel IO configuration structure, for MIPI DSI command interface.

Public Members

uint8_t **virtual_channel**

Virtual channel ID, index from 0

int **lcd_cmd_bits**

Bit-width of LCD command

int **lcd_param_bits**

Bit-width of LCD parameter

struct **esp_lcd_dpi_panel_config_t**

MIPI DSI DPI panel configuration structure.

Public Members

uint8_t **virtual_channel**

Virtual channel ID, index from 0

mipi_dsi_dpi_clock_source_t **dpi_clk_src**

MIPI DSI DPI clock source

uint32_t **dpi_clock_freq_mhz**

DPI clock frequency in MHz

lcd_color_rgb_pixel_format_t **pixel_format**

Pixel format that used by the MIPI LCD device

lcd_color_format_t **in_color_format**

Format of the input data (color space and pixel format), which is the format stored in the frame buffer

lcd_color_format_t **out_color_format**

Format of the output data (color space and pixel format), which is the format that the panel can accept

uint8_t **num_fbs**

Number of screen-sized frame buffers that allocated by the driver By default (set to either 0 or 1) only one frame buffer will be created

esp_lcd_video_timing_t **video_timing**

Video timing

struct *esp_lcd_dpi_panel_config_t*::*extra_dpi_panel_flags* **flags**

Extra configuration flags

struct **extra_dpi_panel_flags**

Extra configuration flags for MIPI DSI DPI panel.

Public Members

uint32_t **use_dma2d**

Use DMA2D to copy user buffer to the frame buffer when necessary

uint32_t **disable_lp**

Disable low-power for DPI

struct **esp_lcd_dpi_panel_event_data_t**

Type of LCD DPI panel event data.

struct **esp_lcd_dpi_panel_event_callbacks_t**

Type of LCD DPI panel callbacks.

Public Members

esp_lcd_dpi_panel_color_trans_done_cb_t **on_color_trans_done**

Invoked when user's color buffer copied to the internal frame buffer. This is an indicator that the draw buffer can be recycled safely. But doesn't mean the draw buffer finishes the refreshing to the screen.

esp_lcd_dpi_panel_refresh_done_cb_t **on_refresh_done**

Invoked when the internal frame buffer finishes refreshing to the screen

Type Definitions

typedef struct esp_lcd_dsi_bus_t ***esp_lcd_dsi_bus_handle_t**

Type of MIPI DSI bus handle

typedef bool (*`esp_lcd_dpi_panel_general_cb_t`)(*esp_lcd_panel_handle_t* panel, *esp_lcd_dpi_panel_event_data_t* *edata, void *user_ctx)

A general function callback prototype for DPI panel driver.

Param panel [in] LCD panel handle, which is created by factory API like esp_lcd_new_panel_dpi()
Param edata [in] DPI panel event data, fed by driver
Param user_ctx [in] User data
Return Whether a high priority task has been waken up by this function

typedef *esp_lcd_dpi_panel_general_cb_t* `esp_lcd_dpi_panel_color_trans_done_cb_t`

Declare the prototype of the function that will be invoked when driver finishes coping user's color buffer to frame buffer.

typedef *esp_lcd_dpi_panel_general_cb_t* `esp_lcd_dpi_panel_refresh_done_cb_t`

Declare the prototype of the function that will be invoked when driver finishes refreshing the frame buffer to the screen.

备注: ESP-IDF provides only a limited number of LCD device controller drivers out of the box (e.g., ST7789), more drivers are available in the Espressif Component Registry.

LCD Control Panel Operations

- `esp_lcd_panel_reset()` can reset the LCD control panel.
- `esp_lcd_panel_init()` performs a basic initialization of the control panel. To perform more manufacture specific initialization, please refer to *Steps to Add Manufacture Specific Initialization*.
- By combining using `esp_lcd_panel_swap_xy()` and `esp_lcd_panel_mirror()`, you can achieve the functionality of rotating or mirroring the LCD screen.
- `esp_lcd_panel_disp_on_off()` can turn on or off the LCD screen by cutting down the output path from the frame buffer to the LCD screen. Please note, this is not controlling the LCD backlight. Backlight control is not covered by the `esp_lcd` driver.
- `esp_lcd_panel_disp_sleep()` can reduce the power consumption of the LCD screen by entering the sleep mode. The internal frame buffer is still retained.

LCD Data Panel Operations

- `esp_lcd_panel_reset()` can reset the LCD data panel.
- `esp_lcd_panel_init()` performs a basic initialization of the data panel.
- `esp_lcd_panel_draw_bitmap()` is the function which does the magic to flush the user draw buffer to the LCD screen, where the target draw window is configurable. Please note, this function expects the draw buffer is a 1-D array and there's no stride in between each lines.

Steps to Add Manufacture Specific Initialization

The LCD controller drivers (e.g., st7789) in ESP-IDF only provide basic initialization in the `esp_lcd_panel_init()`, leaving the vast majority of settings to the default values. Some LCD modules needs to set a bunch of manufacture specific configurations before it can display normally. These configurations usually include gamma, power voltage and so on. If you want to add manufacture specific initialization, please follow the steps below:

```
esp_lcd_panel_reset(panel_handle);
esp_lcd_panel_init(panel_handle);
```

(下页继续)

(续上页)

```
// set extra configurations e.g., gamma control
// with the underlying IO handle
// please consult your manufacture for special commands and corresponding values
esp_lcd_panel_io_tx_param(io_handle, GAMMA_CMD, (uint8_t[]) {
        GAMMA_ARRAY
    }, N);
// turn on the display
esp_lcd_panel_disp_on_off(panel_handle, true);
```

Application Example

- Software JPEG decoding and display - peripherals/lcd/tjpgd
- Universal SPI LCD example with SPI touch - peripherals/lcd/spi_lcd_touch
- i80 controller based LCD and LVGL animation UI - peripherals/lcd/i80_controller
- I2C interfaced OLED display scrolling text - peripherals/lcd/i2c_oled
- MIPI DSI display example - peripherals/lcd/mipi_dsi

API Reference

Header File

- components/hal/include/hal/lcd_types.h
- This header file can be included with:

```
#include "hal/lcd_types.h"
```

Type Definitions

typedef *soc_periph_lcd_clk_src_t* **lcd_clock_source_t**

LCD clock source.

Enumerations

enum **lcd_rgb_data_endian_t**

RGB data endian.

Values:

enumerator **LCD_RGB_DATA_ENDIAN_BIG**

RGB data endian: MSB first

enumerator **LCD_RGB_DATA_ENDIAN_LITTLE**

RGB data endian: LSB first

enum **lcd_color_space_t**

LCD color space.

Values:

enumerator **LCD_COLOR_SPACE_RGB**

Color space: RGB

enumerator **LCD_COLOR_SPACE_YUV**
Color space: YUV

enum **lcd_color_rgb_pixel_format_t**
LCD color pixel format in RGB color space.
Values:

enumerator **LCD_COLOR_PIXEL_FORMAT_RGB565**
16 bits, 5 bits per R/B value, 6 bits for G value

enumerator **LCD_COLOR_PIXEL_FORMAT_RGB666**
18 bits, 6 bits per R/G/B value

enumerator **LCD_COLOR_PIXEL_FORMAT_RGB888**
24 bits, 8 bits per R/G/B value

enum **lcd_color_format_t**
LCD color format.
Values:

enumerator **LCD_COLOR_FMT_RGB565**
RGB565.

enumerator **LCD_COLOR_FMT_RGB666**
RGB666.

enumerator **LCD_COLOR_FMT_RGB888**
RGB888.

enumerator **LCD_COLOR_FMT_YUV422**
YUV422.

enum **lcd_color_range_t**
LCD color range.
Values:

enumerator **LCD_COLOR_RANGE_LIMIT**
Limited color range

enumerator **LCD_COLOR_RANGE_FULL**
Full color range

enum **lcd_yuv_sample_t**
YUV sampling method.
Values:

enumerator **LCD_YUV_SAMPLE_422**
YUV 4:2:2 sampling

enumerator **LCD_YUV_SAMPLE_420**

YUV 4:2:0 sampling

enumerator **LCD_YUV_SAMPLE_411**

YUV 4:1:1 sampling

enum **lcd_yuv_conv_std_t**

The standard used for conversion between RGB and YUV.

Values:

enumerator **LCD_YUV_CONV_STD_BT601**

YUV<->RGB conversion standard: BT.601

enumerator **LCD_YUV_CONV_STD_BT709**

YUV<->RGB conversion standard: BT.709

enum **lcd_yuv422_pack_order_t**

YUV422 packing order.

Values:

enumerator **LCD_YUV422_PACK_ORDER_YUYV**

YUYV

enumerator **LCD_YUV422_PACK_ORDER_YVYU**

YVYU

enumerator **LCD_YUV422_PACK_ORDER_UYVY**

UYVY

enumerator **LCD_YUV422_PACK_ORDER_VYUY**

VYUY

Header File

- components/esp_lcd/include/esp_lcd_types.h
- This header file can be included with:

```
#include "esp_lcd_types.h"
```

- This header file is a part of the API provided by the `esp_lcd` component. To declare that your component depends on `esp_lcd`, add the following to your CMakeLists.txt:

```
REQUIRES esp_lcd
```

or

```
PRIV_REQUIRES esp_lcd
```

Structures

struct **esp_lcd_video_timing_t**

Timing parameters for the video data transmission.

Public Members

uint32_t **h_size**

Horizontal resolution, i.e. the number of pixels in a line

uint32_t **v_size**

Vertical resolution, i.e. the number of lines in the frame

uint32_t **hsync_pulse_width**

Horizontal sync width, in pixel clock

uint32_t **hsync_back_porch**

Horizontal back porch, number of pixel clock between hsync and start of line active data

uint32_t **hsync_front_porch**

Horizontal front porch, number of pixel clock between the end of active data and the next hsync

uint32_t **vsync_pulse_width**

Vertical sync width, in number of lines

uint32_t **vsync_back_porch**

Vertical back porch, number of invalid lines between vsync and start of frame

uint32_t **vsync_front_porch**

Vertical front porch, number of invalid lines between the end of frame and the next vsync

struct **esp_lcd_panel_io_event_data_t**

Type of LCD panel IO event data.

struct **esp_lcd_panel_io_callbacks_t**

Type of LCD panel IO callbacks.

Public Members

esp_lcd_panel_io_color_trans_done_cb_t **on_color_trans_done**

Callback invoked when color data transfer has finished

struct **esp_lcd_color_conv_config_t**

Configuration of LCD color conversion.

Public Members

lcd_color_range_t **in_color_range**

Color range of the input color

lcd_color_range_t **out_color_range**

Color range of the output color

lcd_yuv_conv_std_t **conv_std**

YUV conversion standard: BT601, BT709

lcd_yuv422_pack_order_t **in_pack_order**

YUV422 packing order of the input color

struct *esp_lcd_color_conv_config_t*::[anonymous]::[anonymous]::[anonymous] **yuv422**

YUV422 specific

struct *esp_lcd_color_conv_config_t*::[anonymous]::[anonymous] **yuv**

YUV specific

union *esp_lcd_color_conv_config_t*::[anonymous] **spec**

Extra configuration for specific color conversion

Type Definitions

typedef struct esp_lcd_panel_io_t ***esp_lcd_panel_io_handle_t**

Type of LCD panel IO handle

typedef struct esp_lcd_panel_t ***esp_lcd_panel_handle_t**

Type of LCD panel handle

typedef bool (***esp_lcd_panel_io_color_trans_done_cb_t**)(*esp_lcd_panel_io_handle_t* panel_io, *esp_lcd_panel_io_event_data_t* *edata, void *user_ctx)

Declare the prototype of the function that will be invoked when panel IO finishes transferring color data.

Param panel_io [in] LCD panel IO handle, which is created by factory API like `esp_lcd_new_panel_io_spi()`
Param edata [in] Panel IO event data, fed by driver
Param user_ctx [in] User data, passed from `esp_lcd_panel_io_xxx_config_t`
Return Whether a high priority task has been waken up by this function

Enumerations

enum **lcd_rgb_element_order_t**

RGB element order.

Values:

enumerator **LCD_RGB_ELEMENT_ORDER_RGB**

RGB element order: RGB

enumerator **LCD_RGB_ELEMENT_ORDER_BGR**

RGB element order: BGR

Header File

- components/esp_lcd/include/esp_lcd_panel_io.h
- This header file can be included with:

```
#include "esp_lcd_panel_io.h"
```

- This header file is a part of the API provided by the `esp_lcd` component. To declare that your component depends on `esp_lcd`, add the following to your CMakeLists.txt:

```
REQUIRES esp_lcd
```

or

```
PRIV_REQUIRES esp_lcd
```

Functions

esp_err_t **esp_lcd_panel_io_rx_param**(*esp_lcd_panel_io_handle_t* io, int lcd_cmd, void *param, size_t param_size)

Transmit LCD command and receive corresponding parameters.

备注: Commands sent by this function are short, so they are sent using polling transactions. The function does not return before the command transfer is completed. If any queued transactions sent by `esp_lcd_panel_io_tx_color()` are still pending when this function is called, this function will wait until they are finished and the queue is empty before sending the command(s).

参数

- **io** -- **[in]** LCD panel IO handle, which is created by other factory API like `esp_lcd_new_panel_io_spi()`
- **lcd_cmd** -- **[in]** The specific LCD command, set to -1 if no command needed
- **param** -- **[out]** Buffer for the command data
- **param_size** -- **[in]** Size of `param` buffer

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_SUPPORTED if read is not supported by transport
- ESP_OK on success

esp_err_t **esp_lcd_panel_io_tx_param**(*esp_lcd_panel_io_handle_t* io, int lcd_cmd, const void *param, size_t param_size)

Transmit LCD command and corresponding parameters.

备注: Commands sent by this function are short, so they are sent using polling transactions. The function does not return before the command transfer is completed. If any queued transactions sent by `esp_lcd_panel_io_tx_color()` are still pending when this function is called, this function will wait until they are finished and the queue is empty before sending the command(s).

参数

- **io** -- **[in]** LCD panel IO handle, which is created by other factory API like `esp_lcd_new_panel_io_spi()`
- **lcd_cmd** -- **[in]** The specific LCD command, set to -1 if no command needed
- **param** -- **[in]** Buffer that holds the command specific parameters, set to NULL if no parameter is needed for the command
- **param_size** -- **[in]** Size of `param` in memory, in bytes, set to zero if no parameter is needed for the command

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

esp_err_t **esp_lcd_panel_io_tx_color**(*esp_lcd_panel_io_handle_t* io, int lcd_cmd, const void *color, size_t color_size)

Transmit LCD RGB data.

备注： This function will package the command and RGB data into a transaction, and push into a queue. The real transmission is performed in the background (DMA+interrupt). The caller should take care of the lifecycle of the `color` buffer. Recycling of color buffer should be done in the callback `on_color_trans_done()`.

参数

- **`io`** -- **[in]** LCD panel IO handle, which is created by factory API like `esp_lcd_new_panel_io_spi()`
- **`lcd_cmd`** -- **[in]** The specific LCD command, set to -1 if no command needed
- **`color`** -- **[in]** Buffer that holds the RGB color data
- **`color_size`** -- **[in]** Size of `color` in memory, in bytes

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

esp_err_t **`esp_lcd_panel_io_del`**(*esp_lcd_panel_io_handle_t* io)

Destroy LCD panel IO handle (deinitialize panel and free all corresponding resource)

参数 **`io`** -- **[in]** LCD panel IO handle, which is created by factory API like `esp_lcd_new_panel_io_spi()`

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

esp_err_t **`esp_lcd_panel_io_register_event_callbacks`**(*esp_lcd_panel_io_handle_t* io, const *esp_lcd_panel_io_callbacks_t* *cbs, void *user_ctx)

Register LCD panel IO callbacks.

参数

- **`io`** -- **[in]** LCD panel IO handle, which is created by factory API like `esp_lcd_new_panel_io_spi()`
- **`cbs`** -- **[in]** structure with all LCD panel IO callbacks
- **`user_ctx`** -- **[in]** User private data, passed directly to callback's user_ctx

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

Header File

- components/esp_lcd/include/esp_lcd_panel_ops.h
- This header file can be included with:

```
#include "esp_lcd_panel_ops.h"
```

- This header file is a part of the API provided by the `esp_lcd` component. To declare that your component depends on `esp_lcd`, add the following to your CMakeLists.txt:

```
REQUIRES esp_lcd
```

or

```
PRIV_REQUIRES esp_lcd
```

Functions

esp_err_t **`esp_lcd_panel_reset`**(*esp_lcd_panel_handle_t* panel)

Reset LCD panel.

备注: Panel reset must be called before attempting to initialize the panel using `esp_lcd_panel_init()`.

参数 **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`

返回

- ESP_OK on success

esp_err_t **esp_lcd_panel_init**(*esp_lcd_panel_handle_t* panel)

Initialize LCD panel.

备注: Before calling this function, make sure the LCD panel has finished the `reset` stage by `esp_lcd_panel_reset()`.

参数 **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`

返回

- ESP_OK on success

esp_err_t **esp_lcd_panel_del**(*esp_lcd_panel_handle_t* panel)

Deinitialize the LCD panel.

参数 **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`

返回

- ESP_OK on success

esp_err_t **esp_lcd_panel_draw_bitmap**(*esp_lcd_panel_handle_t* panel, int x_start, int y_start, int x_end, int y_end, const void *color_data)

Draw bitmap on LCD panel.

参数

- **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **x_start** -- **[in]** Start pixel index in the target frame buffer, on x-axis (x_start is included)
- **y_start** -- **[in]** Start pixel index in the target frame buffer, on y-axis (y_start is included)
- **x_end** -- **[in]** End pixel index in the target frame buffer, on x-axis (x_end is not included)
- **y_end** -- **[in]** End pixel index in the target frame buffer, on y-axis (y_end is not included)
- **color_data** -- **[in]** RGB color data that will be dumped to the specific window range

返回

- ESP_OK on success

esp_err_t **esp_lcd_panel_mirror**(*esp_lcd_panel_handle_t* panel, bool mirror_x, bool mirror_y)

Mirror the LCD panel on specific axis.

备注: Combined with `esp_lcd_panel_swap_xy()`, one can realize screen rotation

参数

- **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **mirror_x** -- **[in]** Whether the panel will be mirrored about the x axis
- **mirror_y** -- **[in]** Whether the panel will be mirrored about the y axis

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

esp_err_t **esp_lcd_panel_swap_xy**(*esp_lcd_panel_handle_t* panel, bool swap_axes)

Swap/Exchange x and y axis.

备注: Combined with `esp_lcd_panel_mirror()`, one can realize screen rotation

参数

- **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **swap_axes** -- **[in]** Whether to swap the x and y axis

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

esp_err_t **esp_lcd_panel_set_gap**(*esp_lcd_panel_handle_t* panel, int x_gap, int y_gap)

Set extra gap in x and y axis.

The gap is the space (in pixels) between the left/top sides of the LCD panel and the first row/column respectively of the actual contents displayed.

备注: Setting a gap is useful when positioning or centering a frame that is smaller than the LCD.

参数

- **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **x_gap** -- **[in]** Extra gap on x axis, in pixels
- **y_gap** -- **[in]** Extra gap on y axis, in pixels

返回

- ESP_OK on success

esp_err_t **esp_lcd_panel_invert_color**(*esp_lcd_panel_handle_t* panel, bool invert_color_data)

Invert the color (bit-wise invert the color data line)

参数

- **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **invert_color_data** -- **[in]** Whether to invert the color data

返回

- ESP_OK on success

esp_err_t **esp_lcd_panel_disp_on_off**(*esp_lcd_panel_handle_t* panel, bool on_off)

Turn on or off the display.

参数

- **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **on_off** -- **[in]** True to turns on display, False to turns off display

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

esp_err_t **esp_lcd_panel_disp_off**(*esp_lcd_panel_handle_t* panel, bool off)

Turn off the display.

参数

- **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **off** -- **[in]** Whether to turn off the screen

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

esp_err_t **esp_lcd_panel_disp_sleep**(*esp_lcd_panel_handle_t* panel, bool sleep)

Enter or exit sleep mode.

参数

- **panel** -- **[in]** LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **sleep** -- **[in]** True to enter sleep mode, False to wake up

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

Header File

- components/esp_lcd/include/esp_lcd_panel_vendor.h
- This header file can be included with:

```
#include "esp_lcd_panel_vendor.h"
```

- This header file is a part of the API provided by the `esp_lcd` component. To declare that your component depends on `esp_lcd`, add the following to your CMakeLists.txt:

```
REQUIRES esp_lcd
```

or

```
PRIV_REQUIRES esp_lcd
```

2.5.16 低压差线性稳压器 (LDO)

简介

ESP32-P4 芯片内部集成了 4 路低压差线性稳压器 (LDO)，每路的电压都是可编程调节的。在硬件的参考设计中，我们通常会将其中一些的 LDO 输出作为内部 Flash 和 PSRAM 的电源，剩余的一些 LDO 可以用于给外部设备供电。

备注： 使用前请阅读手册，确保你需要的电流不会超过芯片的规格。

功能概述

下文将分节介绍 LDO 驱动的功能：

- *LDO 通道申请* - 介绍 LDO 通道的种类以及如何申请 LDO 通道资源
- *LDO 通道电压调节* - 介绍如何调节 LDO 通道的输出电压

LDO 通道申请 LDO 通道可以分为两种，一种是电压固定的，另一种是电压可调的。对于同一个输出通道，如果电压固定，那么可以允许有多个用户同时使用（软件允许一个变量拥有多个不可变引用）。如果电压可调，那么只能允许一个用户使用（软件上不允许一个变量有多个可变引用或者可变和不可变引用同时存在）。

LDO 通道在驱动软件中由*esp_ldo_channel_handle_t* 句柄表示。申请 LDO 通道资源的函数是*esp_ldo_acquire_channel()*。申请成功后，会返回一个 LDO 通道的句柄，这个句柄可以用于后续的电压调节操作。在申请通道的时候，我们需要通过*esp_ldo_channel_config_t* 结构体来指定 LDO 通道的基本信息，包括通道 ID，期望的输出电压，以及电压是否可以动态调节。

- *esp_ldo_channel_config_t::chan_id* - LDO 通道的唯一标记，用于区分不同的 LDO 通道。注意，这需要你根据电路原理图和芯片数据手册来确定。比如，一个标记着"LDO_VO3" 的 LDO 通道，对应的 ID 是 3。
- *esp_ldo_channel_config_t::voltage_mv* - 期望的输出电压，单位是毫伏。
- *esp_ldo_channel_config_t::ldo_extra_flags::adjustable* - 是否允许调节输出电压。只有设置为 *true*，才允许使用*esp_ldo_channel_adjust_voltage()* 函数来动态地调节输出电压。

由于允许多个用户同时使用固定电压的 LDO 通道，所以驱动内部会维持一个引用计数器。当最后一个用户释放 LDO 通道资源时，LDO 通道会被自动关闭。释放 LDO 通道资源的函数是*esp_ldo_release_channel()*。另外还需要注意，申请和释放 LDO 通道在使用的时候需要成对出现。

LDO 通道电压调节 *esp_ldo_channel_adjust_voltage()* 函数用来在运行时调整 LDO 通道的输出电压。但是，这个函数只能用于可调节电压的 LDO 通道，否则会返回错误。

注意，由于硬件限制，LDO 通道电压的精度可能会 50~100mV 左右的偏差，请勿依赖于 LDO 通道的输出电压来进行精确的模拟量控制。

应用示例

- Use the internal LDO channel to power up the MIPI DPHY: peripherals/lcd/mipi_dsi

API 参考

Header File

- components/esp_hw_support/ldo/include/esp_ldo_regulator.h
- This header file can be included with:

```
#include "esp_ldo_regulator.h"
```

Functions

esp_err_t **esp_ldo_acquire_channel**(const *esp_ldo_channel_config_t* *config, *esp_ldo_channel_handle_t* *out_handle)

Acquire an LDO channel with the specified configuration.

备注： This function can't automatically search a LDO channel for you, you must specify a LDO channel ID manually, based on your schematic.

备注： The same channel can be acquired multiple times in different places of the application code, however, if the LDO channel is adjustable, you can't acquire it multiple times, in case user A changes the voltage and breaks the voltage setting of user B.

备注： You should release the channel by `esp_ldo_release_channel` when it's no longer needed.

参数

- **config** -- **[in]** The configuration of the LDO channel
- **out_handle** -- **[out]** The returned LDO channel handle

返回

- ESP_OK: Acquire the LDO channel successfully
- ESP_ERR_INVALID_ARG: Acquire the LDO channel failed due to invalid arguments
- ESP_FAIL: Acquire the LDO channel failed due to other reasons

esp_err_t **esp_ldo_release_channel**(*esp_ldo_channel_handle_t* chan)

Release the LDO channel.

参数 **chan** -- **[in]** The LDO channel handle returned from `esp_ldo_acquire_channel`

返回

- ESP_OK: Release the LDO channel successfully
- ESP_ERR_INVALID_ARG: Release the LDO channel failed due to invalid arguments
- ESP_ERR_INVALID_STATE: Release the LDO channel failed due to invalid state, e.g., the channel handle is double released
- ESP_FAIL: Release the LDO channel failed due to other reasons

esp_err_t **esp_ldo_channel_adjust_voltage**(*esp_ldo_channel_handle_t* chan, int voltage_mv)

Adjust the voltage of the LDO channel.

参数

- **chan** -- **[in]** The LDO channel handle returned from `esp_ldo_acquire_channel`
- **voltage_mv** -- **[in]** The voltage value to be set to the LDO channel, in millivolts

返回

- ESP_OK: Adjust the voltage of the LDO channel successfully
- ESP_ERR_INVALID_ARG: Adjust the voltage of the LDO channel failed due to invalid arguments
- ESP_ERR_NOT_SUPPORTED: Adjust the voltage of the LDO channel failed due to the channel is not adjustable
- ESP_FAIL: Adjust the voltage of the LDO channel failed due to other reasons

esp_err_t **esp_ldo_dump**(FILE *stream)

Dump LDO channel status to the specified stream.

参数 **stream** -- **[in]** IO stream. Can be stdout, stderr, or a file/string stream.

返回

- ESP_OK: Dump the LDO channel status successfully
- ESP_FAIL: Dump the LDO channel status failed

Structures

struct **esp_ldo_channel_config_t**

LDO channel configurations.

Public Members

int **chan_id**

You must set the LDO channel ID according to the datasheet, e.g., set it to 1 for LDO_VO1

int **voltage_mv**

The voltage value to be set to the LDO channel

struct *esp_ldo_channel_config_t*::*ldo_extra_flags* **flags**

Flags for the LDO channel

struct **ldo_extra_flags**

Extra flags of a LDO channel.

Public Members

uint32_t **adjustable**

Whether the LDO channel is adjustable, and the voltage can be updated by `esp_ldo_channel_adjust_voltage`

uint32_t **owned_by_hw**

If the LDO channel is owned by hardware, then software configurations will be overridden by hardware

Type Definitions

typedef struct ldo_regulator_channel_t ***esp_ldo_channel_handle_t**

Type of LDO regulator channel handle.

2.5.17 LED PWM 控制器

概述

LED 控制器 (LEDC) 主要用于控制 LED，也可产生 PWM 信号用于其他设备的控制。该控制器有 8 路通道，可以产生独立的波形，驱动 RGB LED 等设备。

LED PWM 控制器可在无需 CPU 干预的情况下自动改变占空比，实现亮度和颜色渐变。

功能概览

设置 LEDC 通道分三步完成。注意，与 ESP32 不同，ESP32-P4 仅支持设置通道为低速模式。

1. 定时器配置 指定 PWM 信号的频率和占空比分辨率。
2. 通道配置 绑定定时器和输出 PWM 信号的 GPIO。
3. 改变 *PWM* 信号 输出 PWM 信号来驱动 LED。可通过软件控制或使用硬件渐变功能来改变 LED 的亮度。

另一个可选步骤是可以在渐变终端设置一个中断。

备注： 首次 LEDC 配置时，建议先配置定时器（调用函数*ledc_timer_config()*），再配置通道（调用函数*ledc_channel_config()*）。这样可以确保 IO 脚上的 PWM 信号自有输出开始其频率就是正确的。

定时器配置 要设置定时器，可调用函数*ledc_timer_config()*，并将包括如下配置参数的数据结构*ledc_timer_config_t* 传递给该函数：

- 速度模式（值必须为 `LEDC_LOW_SPEED_MODE`）
- 定时器索引*ledc_timer_t*
- PWM 信号频率（Hz）

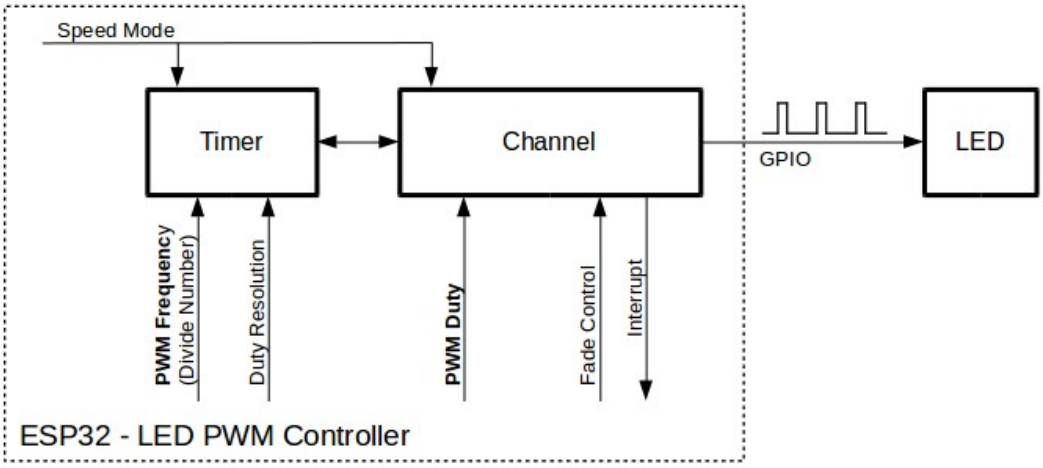


图 16: LED PWM 控制器 API 的关键配置

- PWM 占空比分辨率
- 时钟源 *ledc_clk_cfg_t*

频率和占空比分辨率相互关联。PWM 频率越高，占空比分辨率越低，反之亦然。如果 API 不是用来改变 LED 亮度，而是用于其它目的，这种相互关系可能会很重要。更多信息详见*频率和占空比分辨率支持范围* 一节。

时钟源同样可以限制 PWM 频率。选择的时钟源频率越高，可以配置的 PWM 频率上限就越高。

表 4: ESP32-P4 LEDC 时钟源特性

时钟名称	时钟频率	时钟功能
PLL_80M_CLK	80 MHz	/
RC_FAST_CLK	~ 20 MHz	支持动态调频（DFS）功能，支持 Light-sleep 模式
XTAL_CLK	40 MHz	支持动态调频 (DFS) 功能

备注:

1. 如果 ESP32-P4 的定时器选用了 `RC_FAST_CLK` 作为其时钟源，驱动会通过内部校准来得知这个时钟源的实际频率。这样确保了输出 PWM 信号频率的精准性。
2. ESP32-P4 的所有定时器共用一个时钟源。因此 ESP32-P4 不支持给不同的定时器配置不同的时钟源。

LEDC 驱动提供了一个辅助函数 *ledc_find_suitable_duty_resolution()*。传入时钟源频率及期望的 PWM 信号频率，这个函数可以直接找到最大可配的占空比分辨率值。

当一个定时器不再被任何通道所需要时，可以通过调用相同的函数 *ledc_timer_config()* 来重置这个定时器。此时，函数入参的配置结构体需要指定：

- *ledc_timer_config_t::speed_mode* 重置定时器的所属速度模式（*ledc_mode_t*）
- *ledc_timer_config_t::timer_num* 重置定时器的索引（*ledc_timer_t*）
- *ledc_timer_config_t::deconfigure* 将指定定时器重置必须配置此项为 true

通道配置 定时器设置好后，请配置所需的通道（*ledc_channel_t* 之一）。配置通道需调用函数 *ledc_channel_config()*。

通道的配置与定时器设置类似，需向通道配置函数传递包括通道配置参数的结构体*ledc_channel_config_t*。

此时，通道会按照*ledc_channel_config_t*的配置开始运作，并在选定的 GPIO 上生成由定时器设置指定的频率和占空比的 PWM 信号。在通道运作过程中，可以随时通过调用函数*ledc_stop()*将其暂停。

改变 PWM 信号 通道开始运行、生成具有恒定占空比和频率的 PWM 信号之后，有几种方式可以改变该信号。驱动 LED 时，主要通过改变占空比来变化光线亮度。

以下两节介绍了如何使用软件和硬件改变占空比。如有需要，PWM 信号的频率也可更改，详见*改变 PWM 频率* 一节。

备注： 在 ESP32-P4 的 LED PWM 控制器中，所有的定时器和通道都只支持低速模式。对 PWM 设置的任何改变，都需要由软件显式地触发（见下文）。

使用软件改变 PWM 占空比 调用函数*ledc_set_duty()*可以设置新的占空比。之后，调用函数*ledc_update_duty()*使新配置生效。要查看当前设置的占空比，可使用 _get_ 函数*ledc_get_duty()*。

另外一种设置占空比和其他通道参数的方式是调用*通道配置* 一节提到的函数*ledc_channel_config()*。

传递给函数的占空比数值范围取决于选定的 `duty_resolution`，应为 `0` 至 `(2 ** duty_resolution)`。例如，如选定的占空比分辨率为 10，则占空比的数值范围为 0 至 1024。此时分辨率为 ~ 0.1%。

警告： 在 ESP32-P4 上，当通道绑定的定时器配置了其最大 PWM 占空比分辨率（`MAX_DUTY_RES`），通道的占空比不能被设置到 `(2 ** MAX_DUTY_RES)`。否则，硬件内部占空比计数器会溢出，并导致占空比计算错误。

使用硬件改变 PWM 占空比 LED PWM 控制器硬件可逐渐改变占空比的数值。要使用此功能，需用函数*ledc_fade_func_install()*使能渐变，之后用下列可用渐变函数之一配置：

- *ledc_set_fade_with_time()*
- *ledc_set_fade_with_step()*
- *ledc_set_fade()*

ESP32-P4 的硬件额外支持多达 16 次，无需 CPU 介入的连续渐变。此功能可以更加有效便捷得实现一个带伽马校正的渐变。

众所周知，人眼所感知的亮度与 PWM 占空比并非成线性关系。为了能使人感观上认为一盏灯明暗的变化是线性的，我们对其 PWM 信号的占空比控制必须为非线性的，俗称伽马校正。LED PWM 控制器可以通过多段线型拟合来模仿伽马曲线渐变。你需要自己在应用程序中分配一段用以保存渐变参数的内存块，并提供开始和结束的占空比，伽马校正公式，以及期望的线性渐变段数信息，*ledc_fill_multi_fade_param_list()*就能快速生成所有分段线性渐变的参数。或者你也可以自己直接构造一个*ledc_fade_param_config_t*的数组。在获得所有渐变参数后，通过将*ledc_fade_param_config_t*数组的指针和渐变区间数传入*ledc_set_multi_fade()*，一次连续渐变的配置就完成了。

最后需要调用*ledc_fade_start()*开启渐变。渐变可以在阻塞或非阻塞模式下运行，具体区别请查看*ledc_fade_mode_t*。需要特别注意的是，不管在哪种模式下，下一次渐变或是单次占空比配置的指令生效都必须等到前一次渐变完成或被中止。中止一个正在运行中的渐变需要调用函数*ledc_fade_stop()*。

此外，在使能渐变后，每个通道都可以额外通过调用*ledc_cb_register()*注册一个回调函数用以获得渐变完成的事件通知。回调函数的原型被定义在*ledc_cb_t*。每个回调函数都应当返回一个布尔值给

驱动的中断处理函数，用以表示是否有高优先级任务被其唤醒。此外，值得注意的是，由于驱动的中断处理函数被放在了 IRAM 中，回调函数和其调用的函数也需要被放在 IRAM 中。*ledc_cb_register()* 会检查回调函数及函数上下文的指针地址是否在正确的存储区域。

如不需要渐变和渐变中断，可用函数*ledc_fade_func_uninstall()* 关闭。

改变 PWM 频率 LED PWM 控制器 API 有多种方式即时改变 PWM 频率：

- 通过调用函数*ledc_set_freq()* 设置频率。可用函数*ledc_get_freq()* 查看当前频率。
- 通过调用函数*ledc_bind_channel_timer()* 将其他定时器绑定到该通道来改变频率和占空比分辨率。
- 通过调用函数*ledc_channel_config()* 改变通道的定时器。

控制 PWM 的更多方式 有一些较底层的定时器特定函数可用于更改 PWM 设置：

- *ledc_timer_set()*
- *ledc_timer_rst()*
- *ledc_timer_pause()*
- *ledc_timer_resume()*

前两个功能可通过函数*ledc_channel_config()* 在后台运行，在定时器配置后启动。

使用中断 配置 LED PWM 控制器通道时，可在*ledc_channel_config_t* 中选取参数*ledc_intr_type_t*，在渐变完成时触发中断。

要注册处理程序来处理中断，可调用函数*ledc_isr_register()*。

频率和占空比分辨率支持范围

LED PWM 控制器主要用于驱动 LED。该控制器 PWM 占空比设置的分辨率范围较广。比如，PWM 频率为 5 kHz 时，占空比分辨率最大可为 13 位。这意味着占空比可为 0 至 100% 之间的任意值，分辨率为 ~0.012%（2 ** 13 = 8192 LED 亮度的离散电平）。然而，这些参数取决于为 LED PWM 控制器定时器计时的时钟信号，LED PWM 控制器为通道提供时钟（具体可参考定时器配置 和 **ESP32-P4 技术参考手册 > LED PWM 计时器 (LEDC)** [PDF]）。

LED PWM 控制器可用于生成频率较高的信号，足以为数码相机模组等其他设备提供时钟。此时，最大频率可为 40 MHz，占空比分辨率为 1 位。也就是说，占空比固定为 50%，无法调整。

LED PWM 控制器 API 会在设定的频率和占空比分辨率超过 LED PWM 控制器硬件范围时报错。例如，试图将频率设置为 20 MHz、占空比分辨率设置为 3 位时，串行端口监视器上会报告如下错误：

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try␣
↪reducing freq_hz or duty_resolution. div_param=128
```

此时，占空比分辨率或频率必须降低。比如，将占空比分辨率设置为 2 会解决这一问题，让占空比设置为 25% 的倍数，即 25%、50% 或 75%。

如设置的频率和占空比分辨率低于所支持的最低值，LED PWM 驱动器也会反映并报告，如：

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try␣
↪increasing freq_hz or duty_resolution. div_param=128000000
```

占空比分辨率通常用*ledc_timer_bit_t* 设置，范围是 10 至 15 位。如需较低的占空比分辨率（上至 10，下至 1），可直接输入相应数值。

应用实例

使用 LEDC 基本实例请参照 peripherals/ledc/ledc_basic。

使用 LEDC 改变占空比和渐变控制的实例请参照 peripherals/ledc/ledc_fade。

使用 LEDC 对 RGB LED 实现带伽马校正的颜色控制实例请参照 peripherals/ledc/ledc_gamma_curve_fade。

API 参考

Header File

- components/esp_driver_ledc/include/driver/ledc.h
- This header file can be included with:

```
#include "driver/ledc.h"
```

- This header file is a part of the API provided by the `esp_driver_ledc` component. To declare that your component depends on `esp_driver_ledc`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_ledc
```

or

```
PRIV_REQUIRES esp_driver_ledc
```

Functions

esp_err_t **ledc_channel_config**(const *ledc_channel_config_t* *ledc_conf)

LEDC channel configuration Configure LEDC channel with the given channel/output gpio_num/interrupt/source timer/frequency(Hz)/LEDC duty.

参数 **ledc_conf** -- Pointer of LEDC channel configure struct

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

uint32_t **ledc_find_suitable_duty_resolution**(uint32_t src_clk_freq, uint32_t timer_freq)

Helper function to find the maximum possible duty resolution in bits for ledc_timer_config()

参数

- **src_clk_freq** -- LEDC timer source clock frequency (Hz) (See doxygen comments of `ledc_clk_cfg_t` or get from `esp_clk_tree_src_get_freq_hz`)
- **timer_freq** -- Desired LEDC timer frequency (Hz)

返回

- 0 The timer frequency cannot be achieved
- Others The largest duty resolution value to be set

esp_err_t **ledc_timer_config**(const *ledc_timer_config_t* *timer_conf)

LEDC timer configuration Configure LEDC timer with the given source timer/frequency(Hz)/duty_resolution.

参数 **timer_conf** -- Pointer of LEDC timer configure struct

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Can not find a proper pre-divider number base on the given frequency and the current duty_resolution.
- ESP_ERR_INVALID_STATE Timer cannot be de-configured because timer is not configured or is not paused

esp_err_t **ledc_update_duty**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel)

LEDC update channel parameters.

备注： Call this function to activate the LEDC updated parameters. After ledc_set_duty, we need to call this function to update the settings. And the new LEDC parameters don't take effect until the next PWM cycle.

备注： ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update

备注： If `CONFIG_LEDC_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Cache is disabled.

备注： This function is allowed to run within ISR context.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **ledc_set_pin**(int gpio_num, *ledc_mode_t* speed_mode, *ledc_channel_t* ledc_channel)

Set LEDC output gpio.

备注： This function only routes the LEDC signal to GPIO through matrix, other LEDC resources initialization are not involved. Please use `ledc_channel_config()` instead to fully configure a LEDC channel.

参数

- **gpio_num** -- The LEDC output gpio
- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **ledc_channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **ledc_stop**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t idle_level)

LEDC stop. Disable LEDC output, and set idle level.

备注： If `CONFIG_LEDC_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Cache is disabled.

备注： This function is allowed to run within ISR context.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **idle_level** -- Set output idle level after LEDC stops.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **ledc_set_freq**(*ledc_mode_t* speed_mode, *ledc_timer_t* timer_num, uint32_t freq_hz)

LEDC set channel frequency (Hz)

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **timer_num** -- LEDC timer index (0-3), select from ledc_timer_t
- **freq_hz** -- Set the LEDC frequency

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Can not find a proper pre-divider number base on the given frequency and the current duty_resolution.

uint32_t **ledc_get_freq**(*ledc_mode_t* speed_mode, *ledc_timer_t* timer_num)

LEDC get channel frequency (Hz)

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **timer_num** -- LEDC timer index (0-3), select from ledc_timer_t

返回

- 0 error
- Others Current LEDC frequency

esp_err_t **ledc_set_duty_with_hpoint**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t duty, uint32_t hpoint)

LEDC set duty and hpoint value Only after calling ledc_update_duty will the duty update.

备注: ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update

---

备注: For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **duty** -- Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution)]
- **hpoint** -- Set the LEDC hpoint value, the range is [0, (2**duty_resolution)-1]

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

int **ledc_get_hpoint**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel)

LEDC get hpoint value, the counter value when the output is set high level.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.

- **channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

返回

- LEDC_ERR_VAL if parameter error
- Others Current hpoint value of LEDC channel

esp_err_t **ledc_set_duty**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t duty)

LEDC set duty This function do not change the hpoint value of this channel. if needed, please call ledc_set_duty_with_hpoint. only after calling ledc_update_duty will the duty update.

备注: ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update.

备注: For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **duty** -- Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution)]

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

uint32_t **ledc_get_duty**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel)

LEDC get duty This function returns the duty at the present PWM cycle. You shouldn't expect the function to return the new duty in the same cycle of calling ledc_update_duty, because duty update doesn't take effect until the next cycle.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

返回

- LEDC_ERR_DUTY if parameter error
- Others Current LEDC duty

esp_err_t **ledc_set_fade**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t duty, *ledc_duty_direction_t* fade_direction, uint32_t step_num, uint32_t duty_cycle_num, uint32_t duty_scale)

LEDC set gradient Set LEDC gradient, After the function calls the ledc_update_duty function, the function can take effect.

备注: For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.

- **channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **duty** -- Set the start of the gradient duty, the range of duty setting is [0, (2**duty_resolution)]
- **fade_direction** -- Set the direction of the gradient
- **step_num** -- Set the number of the gradient
- **duty_cycle_num** -- Set how many LEDC tick each time the gradient lasts
- **duty_scale** -- Set gradient change amplitude

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **ledc_isr_register**(void (*fn)(void*), void *arg, int intr_alloc_flags, *ledc_isr_handle_t* *handle)

Register LEDC interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

参数

- **fn** -- Interrupt handler function.
- **arg** -- User-supplied argument passed to the handler function.
- **intr_alloc_flags** -- Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.
- **handle** -- Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NOT_FOUND Failed to find available interrupt source

esp_err_t **ledc_timer_set**(*ledc_mode_t* speed_mode, *ledc_timer_t* timer_sel, uint32_t clock_divider, uint32_t duty_resolution, *ledc_clk_src_t* clk_src)

Configure LEDC settings.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **timer_sel** -- Timer index (0-3), there are 4 timers in LEDC module
- **clock_divider** -- Timer clock divide value, the timer clock is divided from the selected clock source
- **duty_resolution** -- Resolution of duty setting in number of bits. The range is [1, SOC_LEDC_TIMER_BIT_WIDTH]
- **clk_src** -- Select LEDC source clock.

返回

- (-1) Parameter error
- Other Current LEDC duty

esp_err_t **ledc_timer_rst**(*ledc_mode_t* speed_mode, *ledc_timer_t* timer_sel)

Reset LEDC timer.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **timer_sel** -- LEDC timer index (0-3), select from ledc_timer_t

返回

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

esp_err_t **ledc_timer_pause**(*ledc_mode_t* speed_mode, *ledc_timer_t* timer_sel)

Pause LEDC timer counter.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **timer_sel** -- LEDC timer index (0-3), select from ledc_timer_t

返回

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

esp_err_t **ledc_timer_resume**(*ledc_mode_t* speed_mode, *ledc_timer_t* timer_sel)

Resume LEDC timer.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **timer_sel** -- LEDC timer index (0-3), select from ledc_timer_t

返回

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

esp_err_t **ledc_bind_channel_timer**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, *ledc_timer_t* timer_sel)

Bind LEDC channel with the selected timer.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **timer_sel** -- LEDC timer index (0-3), select from ledc_timer_t

返回

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

esp_err_t **ledc_set_fade_with_step**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t target_duty, uint32_t scale, uint32_t cycle_num)

Set LEDC fade function.

备注: Call ledc_fade_func_install() once before calling this function. Call ledc_fade_start() after this to start fading.

备注: ledc_set_fade_with_step, ledc_set_fade_with_time and ledc_fade_start are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_fade_step_and_start

---

备注: For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **target_duty** -- Target duty of fading [0, (2**duty_resolution)]
- **scale** -- Controls the increase or decrease step scale.
- **cycle_num** -- increase or decrease the duty every cycle_num cycles

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_FAIL Fade function init error

esp_err_t **ledc_set_fade_with_time** (*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t target_duty, int max_fade_time_ms)

Set LEDC fade function, with a limited time.

备注: Call ledc_fade_func_install() once before calling this function. Call ledc_fade_start() after this to start fading.

备注: ledc_set_fade_with_step, ledc_set_fade_with_time and ledc_fade_start are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_fade_step_and_start

---

备注: For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **target_duty** -- Target duty of fading [0, (2**duty_resolution)]
- **max_fade_time_ms** -- The maximum time of the fading (ms).

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_FAIL Fade function init error

esp_err_t **ledc_fade_func_install** (int intr_alloc_flags)

Install LEDC fade function. This function will occupy interrupt of LEDC module.

参数 **intr_alloc_flags** -- Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Intr flag error
- ESP_ERR_NOT_FOUND Failed to find available interrupt source
- ESP_ERR_INVALID_STATE Fade function already installed

void **ledc_fade_func_uninstall** (void)

Uninstall LEDC fade function.

esp_err_t **ledc_fade_start** (*ledc_mode_t* speed_mode, *ledc_channel_t* channel, *ledc_fade_mode_t* fade_mode)

Start LEDC fading.

备注: Call ledc_fade_func_install() once before calling this function. Call this API right after ledc_set_fade_with_time or ledc_set_fade_with_step before to start fading.

备注： Starting fade operation with this API is not thread-safe, use with care.

备注： For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel number
- **fade_mode** -- Whether to block until fading done. See ledc_types.h ledc_fade_mode_t for more info. Note that this function will not return until fading to the target duty if LEDC_FADE_WAIT_DONE mode is selected.

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE Channel not initialized or fade function not installed.
- ESP_ERR_INVALID_ARG Parameter error.

esp_err_t **ledc_fade_stop**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel)

Stop LEDC fading. The duty of the channel is guaranteed to be fixed at most one PWM cycle after the function returns.

备注： This API can be called if a new fixed duty or a new fade want to be set while the last fade operation is still running in progress.

备注： Call this API will abort the fading operation only if it was started by calling ledc_fade_start with LEDC_FADE_NO_WAIT mode.

备注： If a fade was started with LEDC_FADE_WAIT_DONE mode, calling this API afterwards has no use in stopping the fade. Fade will continue until it reaches the target duty.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel number

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Fade function init error

esp_err_t **ledc_set_duty_and_update**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t duty, uint32_t hpoint)

A thread-safe API to set duty for LEDC channel and return when duty updated.

备注： For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **duty** -- Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution)]
- **hpoint** -- Set the LEDC hpoint value, the range is [0, (2**duty_resolution)-1]

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Fade function init error

esp_err_t **ledc_set_fade_time_and_start** (*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t target_duty, uint32_t max_fade_time_ms, *ledc_fade_mode_t* fade_mode)

A thread-safe API to set and start LEDC fade function, with a limited time.

备注: Call ledc_fade_func_install() once, before calling this function.

备注: For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **target_duty** -- Target duty of fading [0, (2**duty_resolution)]
- **max_fade_time_ms** -- The maximum time of the fading (ms).
- **fade_mode** -- choose blocking or non-blocking mode

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_FAIL Fade function init error

esp_err_t **ledc_set_fade_step_and_start** (*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t target_duty, uint32_t scale, uint32_t cycle_num, *ledc_fade_mode_t* fade_mode)

A thread-safe API to set and start LEDC fade function.

备注: Call ledc_fade_func_install() once before calling this function.

备注: For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

- **target_duty** -- Target duty of fading [0, (2**duty_resolution)]
- **scale** -- Controls the increase or decrease step scale.
- **cycle_num** -- increase or decrease the duty every cycle_num cycles
- **fade_mode** -- choose blocking or non-blocking mode

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_FAIL Fade function init error

esp_err_t **ledc_cb_register**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, *ledc_cbs_t* *cbs, void *user_arg)

LEDC callback registration function.

备注: The callback is called from an ISR, it must never attempt to block, and any FreeRTOS API called must be ISR capable.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **cbs** -- Group of LEDC callback functions
- **user_arg** -- user registered data for the callback function

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_FAIL Fade function init error

esp_err_t **ledc_set_multi_fade**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t start_duty, const *ledc_fade_param_config_t* *fade_params_list, uint32_t list_len)

Set a LEDC multi-fade.

备 注: Call `ledc_fade_func_install()` once before calling this function. Call `ledc_fade_start()` after this to start fading.

备注: This function is not thread-safe, do not call it to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_multi_fade_and_start

---

备注: This function does not prohibit from duty overflow. User should take care of this by themselves. If duty overflow happens, the PWM signal will suddenly change from 100% duty cycle to 0%, or the other way around.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **start_duty** -- Set the start of the gradient duty, the range of duty setting is [0, (2**duty_resolution)]

- **fade_params_list** -- Pointer to the array of fade parameters for a multi-fade
- **list_len** -- Length of the fade_params_list, i.e. number of fade ranges for a multi-fade (1 - SOC_LEDC_GAMMA_CURVE_FADE_RANGE_MAX)

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_FAIL Fade function init error

esp_err_t **ledc_set_multi_fade_and_start**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t start_duty, const *ledc_fade_param_config_t* *fade_params_list, uint32_t list_len, *ledc_fade_mode_t* fade_mode)

A thread-safe API to set and start LEDC multi-fade function.

备注: Call `ledc_fade_func_install()` once before calling this function.

备注: Fade will always begin from the current duty cycle. Make sure it is stable and synchronized to the desired initial value before calling this function. Otherwise, you may see unexpected duty change.

备注: This function does not prohibit from duty overflow. User should take care of this by themselves. If duty overflow happens, the PWM signal will suddenly change from 100% duty cycle to 0%, or the other way around.

参数

- **speed_mode** -- Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** -- LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **start_duty** -- Set the start of the gradient duty, the range of duty setting is [0, (2**duty_resolution)]
- **fade_params_list** -- Pointer to the array of fade parameters for a multi-fade
- **list_len** -- Length of the fade_params_list, i.e. number of fade ranges for a multi-fade (1 - SOC_LEDC_GAMMA_CURVE_FADE_RANGE_MAX)
- **fade_mode** -- Choose blocking or non-blocking mode

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_FAIL Fade function init error

esp_err_t **ledc_fill_multi_fade_param_list**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t start_duty, uint32_t end_duty, uint32_t linear_phase_num, uint32_t max_fade_time_ms, uint32_t (*gamma_correction_operator)(uint32_t), uint32_t fade_params_list_size, *ledc_fade_param_config_t* *fade_params_list, uint32_t *hw_fade_range_num)

Helper function to fill the fade params for a multi-fade. Useful if desires a gamma curve fading.

备注: The fade params are calculated based on the given start_duty and end_duty. If the duty is not at the start duty (gamma-corrected) when the fade begins, you may see undesired brightness change. Therefore, please always remember thet when passing the fade_params to either `ledc_set_multi_fade`

or `ledc_set_multi_fade_and start`, the start_duty argument has to be the gamma-corrected start_duty.

参数

- **`speed_mode`** -- **[in]** Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **`channel`** -- **[in]** LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **`start_duty`** -- **[in]** Duty cycle [0, (2**duty_resolution)] where the multi-fade begins with. This value should be a non-gamma-corrected duty cycle.
- **`end_duty`** -- **[in]** Duty cycle [0, (2**duty_resolution)] where the multi-fade ends with. This value should be a non-gamma-corrected duty cycle.
- **`linear_phase_num`** -- **[in]** Number of linear fades to simulate a gamma curved fade (1 - SOC_LEDC_GAMMA_CURVE_FADE_RANGE_MAX)
- **`max_fade_time_ms`** -- **[in]** The maximum time of the fading (ms).
- **`gamma_correction_operator`** -- **[in]** User provided gamma correction function. The function argument should be able to take any value within [0, (2**duty_resolution)]. And returns the gamma-corrected duty cycle.
- **`fade_params_list_size`** -- **[in]** The size of the fade_params_list user allocated (1 - SOC_LEDC_GAMMA_CURVE_FADE_RANGE_MAX)
- **`fade_params_list`** -- **[out]** Pointer to the array of *ledc_fade_param_config_t* structure
- **`hw_fade_range_num`** -- **[out]** Number of fade ranges for this multi-fade

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized
- ESP_FAIL Required number of hardware ranges exceeds the size of the *ledc_fade_param_config_t* array user allocated

esp_err_t **`ledc_read_fade_param`**(*ledc_mode_t* speed_mode, *ledc_channel_t* channel, uint32_t range, uint32_t *dir, uint32_t *cycle, uint32_t *scale, uint32_t *step)

Get the fade parameters that are stored in gamma ram for a certain fade range.

Gamma ram is where saves the fade parameters for each fade range. The fade parameters are written in during fade configuration. When fade begins, the duty will change according to the parameters in gamma ram.

参数

- **`speed_mode`** -- **[in]** Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **`channel`** -- **[in]** LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **`range`** -- **[in]** Range index (0 - (SOC_LEDC_GAMMA_CURVE_FADE_RANGE_MAX-1)), it specifies to which range in gamma ram to read
- **`dir`** -- **[out]** Pointer to accept fade direction value
- **`cycle`** -- **[out]** Pointer to accept fade cycle value
- **`scale`** -- **[out]** Pointer to accept fade scale value
- **`step`** -- **[out]** Pointer to accept fade step value

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Channel not initialized

Structures

struct **`ledc_channel_config_t`**

Configuration parameters of LEDC channel for ledc_channel_config function.

Public Members

int **gpio_num**

the LEDC output gpio_num, if you want to use gpio16, gpio_num = 16

ledc_mode_t **speed_mode**

LEDC speed speed_mode, high-speed mode (only exists on esp32) or low-speed mode

ledc_channel_t **channel**

LEDC channel (0 - LEDC_CHANNEL_MAX-1)

ledc_intr_type_t **intr_type**

configure interrupt, Fade interrupt enable or Fade interrupt disable

ledc_timer_t **timer_sel**

Select the timer source of channel (0 - LEDC_TIMER_MAX-1)

uint32_t **duty**

LEDC channel duty, the range of duty setting is [0, (2**duty_resolution)]

int **hpoint**

LEDC channel hpoint value, the range is [0, (2**duty_resolution)-1]

unsigned int **output_invert**

Enable (1) or disable (0) gpio output invert

struct *ledc_channel_config_t*::[anonymous] **flags**

LEDC flags

struct **ledc_timer_config_t**

Configuration parameters of LEDC timer for ledc_timer_config function.

Public Members

ledc_mode_t **speed_mode**

LEDC speed speed_mode, high-speed mode (only exists on esp32) or low-speed mode

ledc_timer_bit_t **duty_resolution**

LEDC channel duty resolution

ledc_timer_t **timer_num**

The timer source of channel (0 - LEDC_TIMER_MAX-1)

uint32_t **freq_hz**

LEDC timer frequency (Hz)

ledc_clk_cfg_t **clk_cfg**

Configure LEDC source clock from ledc_clk_cfg_t. Note that LEDC_USE_RC_FAST_CLK and LEDC_USE_XTAL_CLK are non-timer-specific clock sources. You can not have one LEDC timer uses RC_FAST_CLK as the clock source and have another LEDC timer uses XTAL_CLK as its clock source. All chips except esp32 and esp32s2 do not have timer-specific clock sources, which means clock source for all timers must be the same one.

bool **deconfigure**

Set this field to de-configure a LEDC timer which has been configured before Note that it will not check whether the timer wants to be de-configured is binded to any channel. Also, the timer has to be paused first before it can be de-configured. When this field is set, duty_resolution, freq_hz, clk_cfg fields are ignored.

struct **ledc_cb_param_t**

LEDC callback parameter.

Public Members

ledc_cb_event_t **event**

Event name

uint32_t **speed_mode**

Speed mode of the LEDC channel group

uint32_t **channel**

LEDC channel (0 - LEDC_CHANNEL_MAX-1)

uint32_t **duty**

LEDC current duty of the channel, the range of duty is [0, (2**duty_resolution)]

struct **ledc_cbs_t**

Group of supported LEDC callbacks.

备注： The callbacks are all running under ISR environment

Public Members

ledc_cb_t **fade_cb**

LEDC fade_end callback function

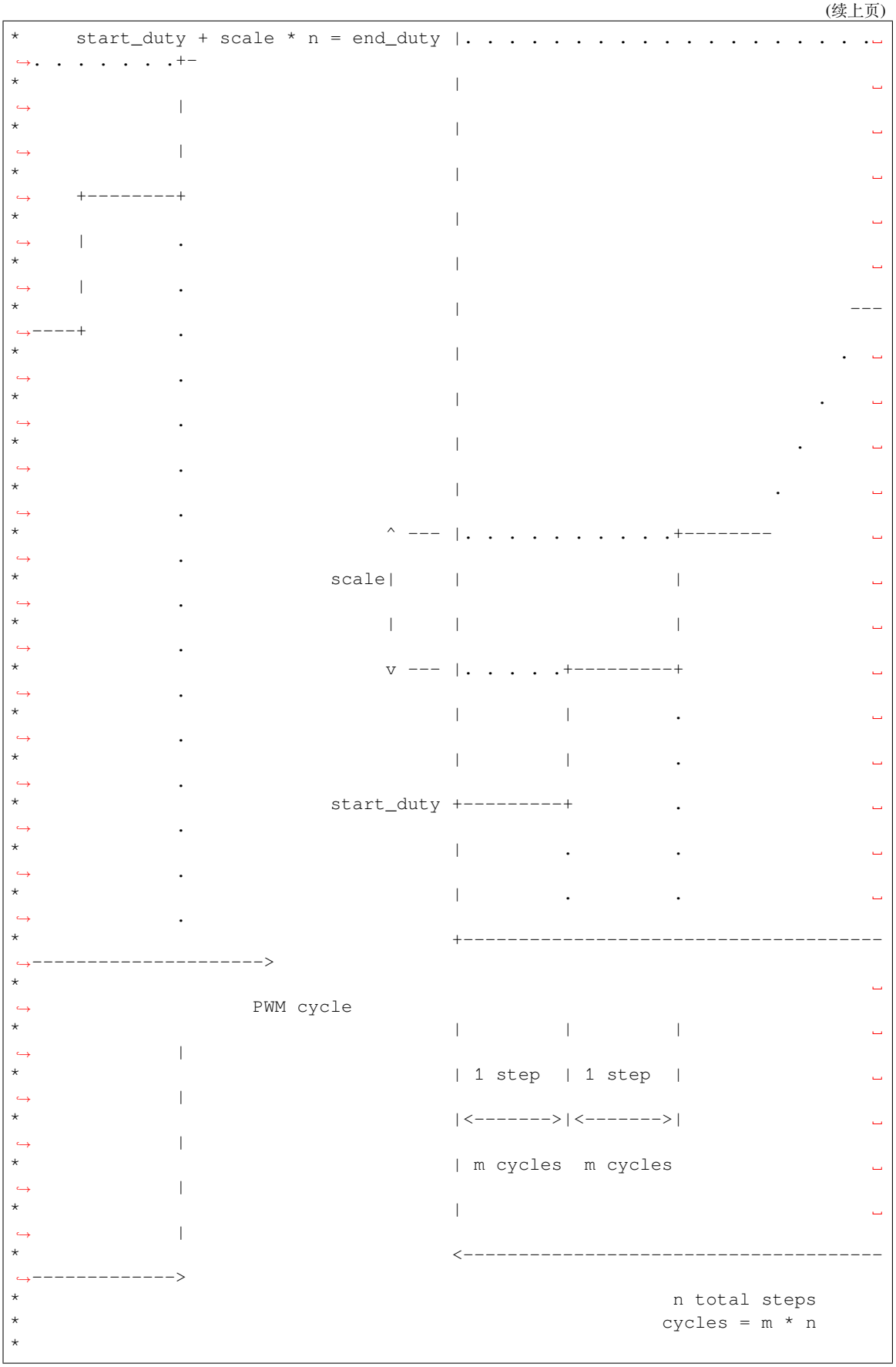
struct **ledc_fade_param_config_t**

Structure for the fade parameters for one hardware fade to be written to gamma wr register.

```
*                                    duty ^                  ONE HW LINEAR FADE
*                                         |
*                                         |
*                                         |
*                                         |
```

(下页继续)

(续上页)


start_duty + scale * n = end_duty
scale
start_duty
PWM cycle
1 step
1 step
m cycles
m cycles
n total steps
cycles = m * n

备注： Be aware of the maximum value available on each element

Public Members

uint32_t **dir**

Duty change direction. Set 1 as increase, 0 as decrease

uint32_t **cycle_num**

Number of PWM cycles of each step [0, 2**SOC_LEDC_FADE_PARAMS_BIT_WIDTH-1]

uint32_t **scale**

Duty change of each step [0, 2**SOC_LEDC_FADE_PARAMS_BIT_WIDTH-1]

uint32_t **step_num**

Total number of steps in one hardware fade [0, 2**SOC_LEDC_FADE_PARAMS_BIT_WIDTH-1]

Macros

LEDC_ERR_DUTY

LEDC_ERR_VAL

Type Definitions

typedef *intr_handle_t* **ledc_isr_handle_t**

typedef bool (***ledc_cb_t**)(const *ledc_cb_param_t* *param, void *user_arg)

Type of LEDC event callback.

Param param LEDC callback parameter
Param user_arg User registered data
Return Whether a high priority task has been waken up by this function

Enumerations

enum **ledc_cb_event_t**

LEDC callback event type.

Values:

enumerator **LEDC_FADE_END_EVT**

LEDC fade end event

Header File

- components/hal/include/hal/ledc_types.h
- This header file can be included with:

```
#include "hal/ledc_types.h"
```

Type Definitions

typedef *soc_periph_ledc_clk_src_legacy_t* **ledc_clk_cfg_t**

LEDC clock source configuration struct.

In theory, the following enumeration shall be placed in LEDC driver's header. However, as the next enumeration, `ledc_clk_src_t`, makes the use of some of these values and to avoid mutual inclusion of the headers, we must define it here.

Enumerations

enum **ledc_mode_t**

Values:

enumerator **LEDC_LOW_SPEED_MODE**

LEDC low speed speed_mode

enumerator **LEDC_SPEED_MODE_MAX**

LEDC speed limit

enum **ledc_intr_type_t**

Values:

enumerator **LEDC_INTR_DISABLE**

Disable LEDC interrupt

enumerator **LEDC_INTR_FADE_END**

Enable LEDC interrupt

enumerator **LEDC_INTR_MAX**

enum **ledc_duty_direction_t**

Values:

enumerator **LEDC_DUTY_DIR_DECREASE**

LEDC duty decrease direction

enumerator **LEDC_DUTY_DIR_INCREASE**

LEDC duty increase direction

enumerator **LEDC_DUTY_DIR_MAX**

enum **ledc_slow_clk_sel_t**

LEDC global clock sources.

Values:

enumerator **LEDC_SLOW_CLK_RC_FAST**

LEDC low speed timer clock source is RC_FAST clock

enumerator **LEDC_SLOW_CLK_PLL_DIV**

LEDC low speed timer clock source is a PLL_DIV clock

enumerator **LEDC_SLOW_CLK_XTAL**

LEDC low speed timer clock source XTAL clock

enumerator **LEDC_SLOW_CLK_RTC8M**

Alias of 'LEDC_SLOW_CLK_RC_FAST'

enum **ledc_clk_src_t**

LEDC timer-specific clock sources.

Note: Setting numeric values to match ledc_clk_cfg_t values are a hack to avoid collision with LEDC_AUTO_CLK in the driver, as these enums have very similar names and user may pass one of these by mistake.

Values:

enumerator **LEDC_SCLK**

Selecting this value for LEDC_TICK_SEL_TIMER let the hardware take its source clock from LEDC_CLK_SEL

enum **ledc_timer_t**

Values:

enumerator **LEDC_TIMER_0**

LEDC timer 0

enumerator **LEDC_TIMER_1**

LEDC timer 1

enumerator **LEDC_TIMER_2**

LEDC timer 2

enumerator **LEDC_TIMER_3**

LEDC timer 3

enumerator **LEDC_TIMER_MAX**

enum **ledc_channel_t**

Values:

enumerator **LEDC_CHANNEL_0**

LEDC channel 0

enumerator **LEDC_CHANNEL_1**

LEDC channel 1

enumerator **LEDC_CHANNEL_2**

LEDC channel 2

enumerator **LEDC_CHANNEL_3**

LEDC channel 3

enumerator **LEDC_CHANNEL_4**
LEDC channel 4

enumerator **LEDC_CHANNEL_5**
LEDC channel 5

enumerator **LEDC_CHANNEL_6**
LEDC channel 6

enumerator **LEDC_CHANNEL_7**
LEDC channel 7

enumerator **LEDC_CHANNEL_MAX**

enum **ledc_timer_bit_t**
Values:

enumerator **LEDC_TIMER_1_BIT**
LEDC PWM duty resolution of 1 bits

enumerator **LEDC_TIMER_2_BIT**
LEDC PWM duty resolution of 2 bits

enumerator **LEDC_TIMER_3_BIT**
LEDC PWM duty resolution of 3 bits

enumerator **LEDC_TIMER_4_BIT**
LEDC PWM duty resolution of 4 bits

enumerator **LEDC_TIMER_5_BIT**
LEDC PWM duty resolution of 5 bits

enumerator **LEDC_TIMER_6_BIT**
LEDC PWM duty resolution of 6 bits

enumerator **LEDC_TIMER_7_BIT**
LEDC PWM duty resolution of 7 bits

enumerator **LEDC_TIMER_8_BIT**
LEDC PWM duty resolution of 8 bits

enumerator **LEDC_TIMER_9_BIT**
LEDC PWM duty resolution of 9 bits

enumerator **LEDC_TIMER_10_BIT**
LEDC PWM duty resolution of 10 bits

enumerator **LEDC_TIMER_11_BIT**
LEDC PWM duty resolution of 11 bits

enumerator **LEDC_TIMER_12_BIT**
LEDC PWM duty resolution of 12 bits

enumerator **LEDC_TIMER_13_BIT**
LEDC PWM duty resolution of 13 bits

enumerator **LEDC_TIMER_14_BIT**
LEDC PWM duty resolution of 14 bits

enumerator **LEDC_TIMER_15_BIT**
LEDC PWM duty resolution of 15 bits

enumerator **LEDC_TIMER_16_BIT**
LEDC PWM duty resolution of 16 bits

enumerator **LEDC_TIMER_17_BIT**
LEDC PWM duty resolution of 17 bits

enumerator **LEDC_TIMER_18_BIT**
LEDC PWM duty resolution of 18 bits

enumerator **LEDC_TIMER_19_BIT**
LEDC PWM duty resolution of 19 bits

enumerator **LEDC_TIMER_20_BIT**
LEDC PWM duty resolution of 20 bits

enumerator **LEDC_TIMER_BIT_MAX**

enum **ledc_fade_mode_t**
Values:

enumerator **LEDC_FADE_NO_WAIT**
LEDC fade function will return immediately

enumerator **LEDC_FADE_WAIT_DONE**
LEDC fade function will block until fading to the target duty

enumerator **LEDC_FADE_MAX**

2.5.18 Camera Controller Driver

Introduction

ESP32-P4 has the following hardware that is intended for communication with external camera sensor:

- MIPI Camera Serial Interface (CSI)

The Camera Controller Driver is designed for this hardware peripheral.

Functional Overview

- *Resource Allocation* - covers how to allocate camera controller instances with properly set of configurations. It also covers how to recycle the resources when they are no longer needed.
- *Enable and disable a camera controller* - covers how to enable and disable a camera controller.
- *Start and stop a camera controller* - covers how to start and stop a camera controller.
- *Receive from a camera sensor or something else* - covers how to receive camera signal from a sensor or something else.
- *Register callback* - covers how to hook user specific code to camera controller driver event callback function.
- *Thread Safety* - lists which APIs are guaranteed to be thread safe by the driver.
- *Kconfig Options* - lists the supported Kconfig options that can bring different effects to the driver.
- *IRAM SAFE* - describes tips on how to make the CSI interrupt and control functions work better along with a disabled cache.

Resource Allocation

Install Camera Controller Driver A Camera Controller Driver can be implemented by the CSI peripheral, which requires the configuration that specified by `esp_cam_ctlr_csi_config_t`.

If the configurations in `esp_cam_ctlr_csi_config_t` is specified, users can call `esp_cam_new_csi_ctlr()` to allocate and initialize a CSI camera controller handle. This function will return an CSI camera controller handle if it runs correctly. You can take following code as reference.

```
esp_cam_ctlr_csi_config_t csi_config = {
    .ctlr_id = 0,
    .h_res = MIPI_CSI_DISP_HSIZE,
    .v_res = MIPI_CSI_DISP_VSIZE_640P,
    .lane_bit_rate_mbps = MIPI_CSI_LANE_BITRATE_MBPS,
    .input_data_color_type = CAM_CTLR_COLOR_RAW8,
    .output_data_color_type = CAM_CTLR_COLOR_RGB565,
    .data_lane_num = 2,
    .byte_swap_en = false,
    .queue_items = 1,
};
esp_cam_ctlr_handle_t handle = NULL;
ESP_ERROR_CHECK(esp_cam_new_csi_ctlr(&csi_config, &handle));
```

Uninstall Camera Controller Driver If a previously installed Camera Controller Driver is no longer needed, it's recommended to recycle the resource by calling `esp_cam_ctlr_del()`, so that to release the underlying hardware.

Enable and Disable Camera Controller Driver Before starting camera controller operation, you need to enable the camera controller controller first, by calling `esp_cam_ctlr_enable()`. This function:

- Switches the driver state from **init** to **enable**.

```
ESP_ERROR_CHECK(esp_cam_ctlr_enable(handle));
```

Calling `esp_cam_ctlr_disable()` does the opposite, that is, put the driver back to the **init** state.

```
ESP_ERROR_CHECK(esp_cam_ctlr_disable(handle));
```

Start and Stop Camera Controller Driver Before receiving camera signal from camera sensor, you need to start the Camera Controller Driver first, by calling `esp_cam_ctlr_start()`. This function:

- Switches the driver stat from **enable** to **start**

```
ESP_ERROR_CHECK(esp_cam_ctlr_start(handle));
```

Calling *esp_cam_ctlr_stop()* does the opposite, that is, put the driver back to the **enable** state.

```
ESP_ERROR_CHECK(esp_cam_ctlr_stop(handle));
```

Receive from A Camera Sensor Now you can call *esp_cam_ctlr_receive()* to receive from a camera sensor or something else.

```
ESP_ERROR_CHECK(esp_cam_ctlr_receive(handle, &my_trans, ESP_CAM_CTLR_MAX_DELAY));
```

Register Event Callbacks After the Camera Controller Driver starts receiving, it can generate a specific event dynamically. If you have some functions that should be called when the event happens, please hook your function to the interrupt service routine by calling *esp_cam_ctlr_register_event_callbacks()*. All supported event callbacks are listed in *esp_cam_ctlr_evt_cbs_t*:

- *esp_cam_ctlr_evt_cbs_t::on_get_new_trans* sets a callback function when the Camera Controller Driver gets a new transaction which is passed from *esp_cam_ctlr_receive()*. As this function is called within the ISR context, you must ensure that the function does not attempt to block (e.g., by making sure that only FreeRTOS APIs with `ISR` suffix are called from within the function).
- *esp_cam_ctlr_evt_cbs_t::on_trans_finished* sets a callback function when the Camera Controller Driver finishes a transaction. As this function is called within the ISR context, you must ensure that the function does not attempt to block (e.g., by making sure that only FreeRTOS APIs with `ISR` suffix are called from within the function).

Thread Safety The factory function *esp_cam_new_csi_ctlr()* and *esp_cam_ctlr_del()* are guaranteed to be thread safe by the driver, which means, user can call them from different RTOS tasks without protection by extra locks.

Kconfig Options

- *CONFIG_CAM_CTLR_MIPI_CSI_ISR_IRAM_SAFE* controls whether the default ISR handler should be masked when the cache is disabled

IRAM Safe By default, the CSI interrupt will be deferred when the cache is disabled because of writing or erasing the flash.

There is a Kconfig option *CONFIG_CAM_CTLR_MIPI_CSI_ISR_IRAM_SAFE* that:

- Enables the interrupt being serviced even when the cache is disabled
- Places all functions that used by the ISR into IRAM
- Places driver object into DRAM (in case it is mapped to PSRAM by accident)

This allows the interrupt to run while the cache is disabled, but comes at the cost of increased IRAM consumption. So user callbacks need to notice that the code and data inside (the callback) should be IRAM-safe or DRAM-safe, when cache is disabled.

API Reference

Header File

- components/esp_driver_cam/include/esp_cam_ctlr.h
- This header file can be included with:

```
#include "esp_cam_ctlr.h"
```

- This header file is a part of the API provided by the `esp_driver_cam` component. To declare that your component depends on `esp_driver_cam`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_cam
```

or

```
PRIV_REQUIRES esp_driver_cam
```

Functions

esp_err_t **esp_cam_ctlr_enable**(*esp_cam_ctlr_handle_t* handle)

Enable ESP CAM controller.

参数 **handle** -- **[in]** ESP CAM controller handle

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid state

esp_err_t **esp_cam_ctlr_start**(*esp_cam_ctlr_handle_t* handle)

Start ESP CAM controller.

参数 **handle** -- **[in]** ESP CAM controller handle

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid state

esp_err_t **esp_cam_ctlr_stop**(*esp_cam_ctlr_handle_t* handle)

Stop ESP CAM controller.

参数 **handle** -- **[in]** ESP CAM controller handle

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid state

esp_err_t **esp_cam_ctlr_disable**(*esp_cam_ctlr_handle_t* handle)

Disable ESP CAM controller.

参数 **handle** -- **[in]** ESP CAM controller handle

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid state

esp_err_t **esp_cam_ctlr_receive**(*esp_cam_ctlr_handle_t* handle, *esp_cam_ctlr_trans_t* *trans, uint32_t timeout_ms)

Receive data to the given transaction.

参数

- **handle** -- **[in]** ESP CAM controller handle
- **trans** -- **[in]** ESP CAM controller transaction type
- **timeout_ms** -- **[in]** Timeout in ms

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid state

esp_err_t **esp_cam_ctlr_del**(*esp_cam_ctlr_handle_t* handle)

Delete ESP CAM controller handle.

参数 **handle** -- **[in]** ESP CAM controller handle

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid state

esp_err_t **esp_cam_ctlr_register_event_callbacks**(*esp_cam_ctlr_handle_t* handle, const *esp_cam_ctlr_evt_cbs_t* *cbs, void *user_data)

Register ESP CAM controller event callbacks.

参数

- **handle** -- **[in]** ESP CAM controller handle
- **cbs** -- **[in]** ESP CAM controller event callbacks
- **user_data** -- **[in]** User data

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid state

esp_err_t **esp_cam_ctlr_get_frame_buffer**(*esp_cam_ctlr_handle_t* handle, uint32_t fb_num, const void **fb0, ...)

Get ESP CAM controller internal malloced backup buffer(s) addr.

备注: Generally, data in internal buffer is ready when `on_trans_finished` event

参数

- **handle** -- **[in]** ESP CAM controller handle
- **fb_num** -- **[in]** Number of frame buffer(s) to get. This value must be the same as the number of the followed fbN parameters
- **fb0** -- **[out]** Address of the frame buffer 0 (first frame buffer)
- **...** -- **[out]** List of other frame buffers if any

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Invalid driver state

esp_err_t **esp_cam_ctlr_get_frame_buffer_len**(*esp_cam_ctlr_handle_t* handle, size_t *ret_fb_len)

Get ESP CAM controller internal backup buffer length.

参数

- **handle** -- **[in]** ESP CAM controller handle
- **ret_fb_len** -- **[out]** Optional, The size of each frame buffer, in bytes.

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: NULL ptr
- ESP_ERR_INVALID_STATE: Invalid driver state

Header File

- components/esp_driver_cam/include/esp_cam_ctlr_types.h
- This header file can be included with:

```
#include "esp_cam_ctlr_types.h"
```

- This header file is a part of the API provided by the `esp_driver_cam` component. To declare that your component depends on `esp_driver_cam`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_cam
```

or

```
PRIV_REQUIRES esp_driver_cam
```

Structures

struct **esp_cam_ctlr_trans_t**

ESP CAM controller transaction type.

Public Members

void ***buffer**

Transaction buffer.

size_t **buflen**

Len of the transaction buffer.

size_t **received_size**

Received size, this received_size will be written by the driver, indicating the real received size.

struct **esp_cam_ctlr_evt_cbs_t**

ESP CAM controller event callbacks.

Public Members

bool (***on_get_new_trans**)(*esp_cam_ctlr_handle_t* handle, *esp_cam_ctlr_trans_t* *trans, void *user_data)

On get new transaction callback.

Param handle **[in]** ESP CAM controller handle
Param trans **[in]** New transaction
Param user_data **[in]** User registered data
Return Whether a high priority task is woken up by this function

bool (***on_trans_finished**)(*esp_cam_ctlr_handle_t* handle, *esp_cam_ctlr_trans_t* *trans, void *user_data)

On transaction finish callback.

Param handle **[in]** ESP CAM controller handle
Param trans **[out]** Finished transaction
Param user_data **[in]** User registered data
Return Whether a high priority task is woken up by this function

Macros

ESP_CAM_CTLR_MAX_DELAY

ESP CAM controller max timeout value.

Type Definitions

typedef struct esp_cam_ctlr_t ***esp_cam_ctlr_handle_t**

ESP CAM controller handle.

Header File

- components/esp_driver_cam/csi/include/esp_cam_ctlr_csi.h
- This header file can be included with:

```
#include "esp_cam_ctlr_csi.h"
```

- This header file is a part of the API provided by the `esp_driver_cam` component. To declare that your component depends on `esp_driver_cam`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_cam
```

or

```
PRIV_REQUIRES esp_driver_cam
```

Functions

esp_err_t **esp_cam_new_csi_ctlr**(const *esp_cam_ctlr_csi_config_t* *config, *esp_cam_ctlr_handle_t* *ret_handle)

New ESP CAM CSI controller.

参数

- **config** -- **[in]** CSI controller configurations
- **ret_handle** -- **[out]** Returned ESP CAM controller handle

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NO_MEM: Out of memory
- ESP_ERR_NOT_SUPPORTED: Currently not support modes or types
- ESP_ERR_NOT_FOUND: CSI is registered already

Structures

struct **esp_cam_ctlr_csi_config_t**

ESP CAM CSI controller configurations.

Public Members

int **ctlr_id**

CSI controller ID.

mipi_csi_phy_clock_source_t **clk_src**

CSI phy clock source.

uint32_t **h_res**

Input horizontal resolution, i.e. the number of pixels in a line.

uint32_t **v_res**

Input vertical resolution, i.e. the number of lines in a frame.

uint8_t **data_lane_num**

Data lane num.

int **lane_bit_rate_mbps**

Lane bit rate in Mbps.

cam_ctlr_color_t **input_data_color_type**

Input color type.

cam_ctlr_color_t **output_data_color_type**

Output color type.

int **queue_items**

Queue items.

uint32_t **byte_swap_en**

Enable byte swap.

uint32_t **bk_buffer_dis**

Disable backup buffer.

struct *esp_cam_ctlr_csi_config_t*::[anonymous] **[anonymous]**

Boolean Flags.

2.5.19 电机控制脉宽调制器 (MCPWM)

MCPWM 外设是一个多功能 PWM 生成器，集成多个子模块，在电力电子应用（如电机控制、数字电源等）中至关重要。MCPWM 外设通常适用于以下场景：

- 数字电机控制，如有刷/无刷直流电机、RC 伺服电机
- 基于开关模式的数字电源转换
- 功率数模转换器 (Power DAC)，其中占空比等于 DAC 的模拟值
- 计算外部脉宽，并将其转换为其他模拟值，如速度、距离
- 为磁场定向控制 (FOC) 生成空间矢量调制 (SVPWM) 信号

外设的主要子模块如下图所示：

- **MCPWM 定时器模块**：最终输出 PWM 信号的时间基准。它也决定了其他子模块的事件时序。
- **MCPWM 操作器模块**：生成 PWM 波形的关键模块。它由其他子模块组成，如比较器、PWM 生成器、死区生成器和载波调制器。
- **MCPWM 比较器模块**：输入时间基准值，并不断与配置的阈值进行比较。当定时器计数值等于任何一个阈值时，生成一个比较事件，MCPWM 生成器随即相应更新其电平。
- **MCPWM 生成器模块**：根据 MCPWM 定时器、MCPWM 比较器等子模块触发的各种事件，生成一对独立或互补的 PWM 波形。
- **MCPWM 故障检测模块**：通过 GPIO 交换矩阵检测外部的故障情况。检测到故障信号时，MCPWM 操作器将强制所有生成器进入预先定义的状态，从而保护系统。
- **MCPWM 同步模块**：同步 MCPWM 定时器，以确保由不同的 MCPWM 生成器最终生成的 PWM 信号具有固定的相位差。可以通过 GPIO 交换矩阵和 MCPWM 定时器事件生成同步信号。
- **死区生成器模块**：在此前生成的 PWM 边沿上插入额外的延迟。
- **载波模块**：可通过 PWM 波形生成器和死区生成器，将一个高频载波信号调制为 PWM 波形，这是控制功率开关器件的必需功能。
- **制动控制**：MCPWM 操作器支持配置检测到特定故障时生成器的制动控制方式。根据故障的严重程度，可以选择立即关闭或是逐周期调节 PWM 输出。

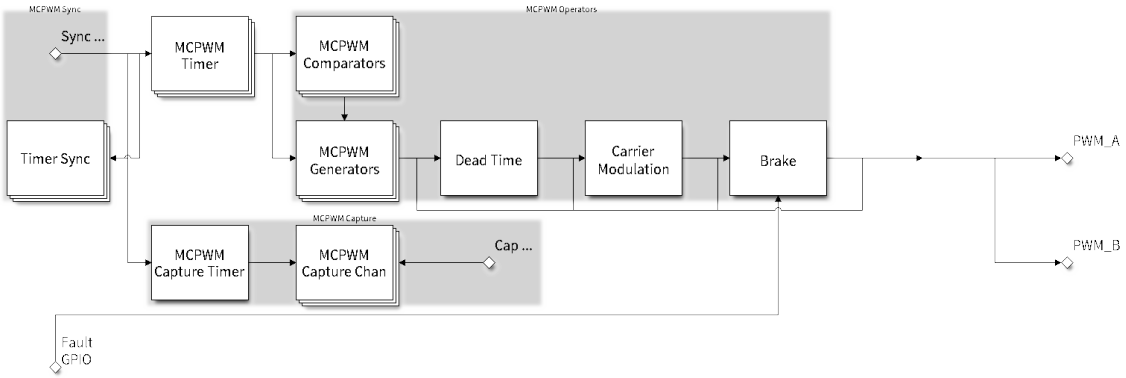


图 17: MCPWM 概述

- **MCPWM 捕获模块**：独立子模块，不依赖于上述 MCPWM 操作器工作。捕获模块包括一个专用的定时器和几个独立的通道，每个通道都与 GPIO 相连。GPIO 上的脉冲触发捕获定时器以存储时间基准值，随后通过中断进行通知。此模块有助于更加精准地测量脉宽。此外，捕获定时器也可以通过 MCPWM 同步子模块进行同步。

功能概述

下文将分节概述 MCPWM 的功能：

- *资源配置及初始化* - 介绍各类 MCPWM 模块的分配，如定时器、操作器、比较器、生成器等。随后介绍的 IO 设置和控制功能也将围绕这些模块进行。
- *定时器操作和事件* - 介绍 MCPWM 定时器支持的控制功能和事件回调。
- *比较器操作和事件* - 介绍 MCPWM 比较器支持的控制功能和事件回调。
- *生成器对事件执行的操作* - 介绍如何针对 MCPWM 定时器和比较器生成的特定事件，设置 MCPWM 生成器的相应执行操作。
- *经典 PWM 波形的生成器配置* - 介绍一些经典 PWM 波形的生成器配置。
- *死区* - 介绍如何设置 MCPWM 生成器的死区时间。
- *经典 PWM 波形的死区配置* - 介绍一些经典 PWM 波形的死区配置。
- *载波调制* - 介绍如何在最终输出的 PWM 波形上调制高频载波。
- *故障检测和制动控制* - 介绍如何为 MCPWM 操作器配置特定故障事件下的制动操作。
- *生成器强制操作* - 介绍如何强制异步控制生成器的输出水平。
- *同步模块* - 介绍如何同步 MCPWM 定时器，并确保生成的最终输出 PWM 信号具有固定的相位差。
- *捕获模块* - 介绍如何使用 MCPWM 捕获模块测量信号脉宽。
- *ETM 事件与任务* - MCPWM 提供了哪些事件和任务可以连接到 ETM 通道上。
- *电源管理* - 介绍不同的时钟源对功耗的影响。
- *IRAM 安全* - 介绍如何协调 RMT 中断与禁用缓存。
- *线程安全* - 列出了由驱动程序认证为线程安全的 API。
- *Kconfig 选项* - 列出了针对驱动的数个 Kconfig 支持选项。

资源配置及初始化　如上图所示，MCPWM 外设由数个子模块组成。本节将介绍各个子模块的资源配置方式。

MCPWM 定时器　调用`mcpwm_new_timer()`函数，以配置结构体`mcpwm_timer_config_t`为参数，分配一个 MCPWM 定时器为对象。结构体定义为：

- `mcpwm_timer_config_t::group_id` 指定 MCPWM 组 ID，范围为 [0, `SOC_MCPWM_GROUPS` - 1]。需注意，位于不同组的定时器彼此独立。

- *mcpwm_timer_config_t::intr_priority* 设置中断的优先级。如果设置为 0，则会分配一个默认优先级的中断，否则会使用指定的优先级。
- *mcpwm_timer_config_t::clk_src* 设置定时器的时钟源。
- *mcpwm_timer_config_t::resolution_hz* 设置定时器的预期分辨率。内部驱动将根据时钟源和分辨率设置合适的分频器。
- *mcpwm_timer_config_t::count_mode* 设置定时器的计数模式。
- *mcpwm_timer_config_t::period_ticks* 设置定时器的周期，以 Tick 为单位（通过*mcpwm_timer_config_t::resolution_hz* 设置 Tick 分辨率）。
- *mcpwm_timer_config_t::update_period_on_empty* 设置当定时器计数为零时是否更新周期值。
- *mcpwm_timer_config_t::update_period_on_sync* 设置当定时器接收同步信号时是否更新周期值。

分配成功后，*mcpwm_new_timer()* 将返回一个指向已分配定时器的指针。否则，函数将返回错误代码。具体来说，当 MCPWM 组中没有空闲定时器时，将返回*ESP_ERR_NOT_FOUND* 错误。[1]

反之，调用*mcpwm_del_timer()* 函数将释放已分配的定时器。

MCPWM 操作器 调用*mcpwm_new_operator()()* 函数，以配置结构体*mcpwm_operator_config_t* 为参数，分配一个 MCPWM 操作器为对象。结构体定义为：

- *mcpwm_operator_config_t::group_id* 指定 MCPWM 组 ID，范围为 [0, *SOC_MCPWM_GROUPS* - 1]。需注意，位于不同组的操作器彼此独立。
- *mcpwm_operator_config_t::intr_priority* 设置中断的优先级。如果设置为 0，则会分配一个默认优先级的中断，否则会使用指定的优先级。
- *mcpwm_operator_config_t::update_gen_action_on_tez* 设置是否在定时器计数为零时更新生成器操作。此处及下文提到的定时器指通过*mcpwm_operator_connect_timer()* 连接到操作器的定时器。
- *mcpwm_operator_config_t::update_gen_action_on_tep* 设置当定时器计数达到峰值时是否更新生成器操作。
- *mcpwm_operator_config_t::update_gen_action_on_sync* 设置当定时器接收同步信号时是否更新生成器操作。
- *mcpwm_operator_config_t::update_dead_time_on_tez* 设置当定时器计数为零时是否更新死区时间。
- *mcpwm_operator_config_t::update_dead_time_on_tep* 设置当定时器计数达到峰值时是否更新死区时间。
- *mcpwm_operator_config_t::update_dead_time_on_sync* 设置当定时器接收同步信号时是否更新死区时间。

分配成功后，*mcpwm_new_operator()()* 将返回一个指向已分配操作器的指针。否则，函数将返回错误代码。具体来说，当 MCPWM 组中没有空闲操作器时，将返回*ESP_ERR_NOT_FOUND* 错误。[1]

反之，调用*mcpwm_del_operator()()* 函数将释放已分配的操作器。

MCPWM 比较器 调用*mcpwm_new_comparator()* 函数，以一个 MCPWM 操作器句柄和配置结构体*mcpwm_comparator_config_t* 为参数，分配一个 MCPWM 比较器为对象。操作器句柄由*mcpwm_new_operator()()* 生成，结构体定义为：

- *mcpwm_comparator_config_t::intr_priority* 设置中断的优先级。如果设置为 0，则会分配一个默认优先级的中断，否则会使用指定的优先级。
- *mcpwm_comparator_config_t::update_cmp_on_tez* 设置当定时器计数为零时是否更新比较阈值。
- *mcpwm_comparator_config_t::update_cmp_on_tep* 设置当定时器计数达到峰值时是否更新比较阈值。
- *mcpwm_comparator_config_t::update_cmp_on_sync* 设置当定时器接收同步信号时是否更新比较阈值。

[1] 不同的 ESP 芯片上的 MCPWM 资源数量可能存在差异（如组、定时器、比较器、操作器、生成器、触发器等）。详情请参见 [TRM]。当分配了超出资源数量的 MCPWM 资源时，在检测到没有可用硬件资源后，驱动程序将返回错误。请在进行资源配置及初始化 时务必检查返回值。

分配成功后，*mcpwm_new_comparator()* 将返回一个指向已分配比较器的指针。否则，函数将返回错误代码。具体来说，当 MCPWM 操作器中没有空闲比较器时，将返回*ESP_ERR_NOT_FOUND* 错误。[1]

反之，调用*mcpwm_del_comparator()* 函数将释放已分配的比较器。

MCPWM 中还有另外一种比较器——“事件比较器”，它不能直接控制 PWM 的输出，只能用来产生 EMT 子系统中使用到的事件。事件比较器能够设置的阈值也是可配的。调用*mcpwm_new_event_comparator()* 函数可以申请一个事件比较器，该函数返回的句柄类型和*mcpwm_new_comparator()* 函数一样，但是需要的配置结构体是不同的。事件比较器的配置位于*mcpwm_event_comparator_config_t*。更多相关内容请参阅*ETM 事件与任务*。

MCPWM 生成器 调用*mcpwm_new_generator()* 函数，以一个 MCPWM 操作器句柄和配置结构体*mcpwm_generator_config_t* 为参数，分配一个 MCPWM 生成器为对象。操作器句柄由*mcpwm_new_operator()()* 生成，结构体定义为：

- *mcpwm_generator_config_t::gen_gpio_num* 设置生成器使用的 GPIO 编号。
- *mcpwm_generator_config_t::invert_pwm* 设置是否反相 PWM 信号。
- *mcpwm_generator_config_t::io_loop_back* 设置是否启用回环模式。该模式仅用于调试，使用 GPIO 交换矩阵外设同时启用 GPIO 输入和输出。
- *mcpwm_generator_config_t::io_od_mode* 设置是否启用漏极开路输出。
- *mcpwm_generator_config_t::pull_up* 和*mcpwm_generator_config_t::pull_down* 用来设置是否启用内部上下拉电阻。

分配成功后，*mcpwm_new_generator()* 将返回一个指向已分配生成器的指针。否则，函数将返回错误代码。具体来说，当 MCPWM 操作器中没有空闲生成器时，将返回*ESP_ERR_NOT_FOUND* 错误。Page 518, 1

反之，调用*mcpwm_del_generator()* 函数将释放已分配的生成器。

MCPWM 故障 MCPWM 故障分为两种类型：来自 GPIO 的故障信号和软件故障。

调用*mcpwm_new_gpio_fault()* 函数，以配置结构体*mcpwm_gpio_fault_config_t* 为参数，分配一个 GPIO 故障为对象。结构体定义为：

- *mcpwm_gpio_fault_config_t::group_id* 设置 MCPWM 组 ID，范围为 [0, *SOC_MCPWM_GROUPS* - 1]。需注意，位于不同组的 GPIO 故障彼此独立，也就是说，1 组的操作器无法检测到 0 组的 GPIO 故障。
- *mcpwm_gpio_fault_config_t::intr_priority* 设置中断的优先级。如果设置为 0，则会分配一个默认优先级的中断，否则会使用指定的优先级。
- *mcpwm_gpio_fault_config_t::gpio_num* 设置故障所使用的 GPIO 编号。
- *mcpwm_gpio_fault_config_t::active_level* 设置故障信号的有效电平。
- *mcpwm_gpio_fault_config_t::pull_up* 和*mcpwm_gpio_fault_config_t::pull_down* 设置是否在内部拉高和/或拉低 GPIO。
- *mcpwm_gpio_fault_config_t::io_loop_back* 设置是否启用回环模式。该模式仅用于调试，使用 GPIO 交换矩阵外设同时启用 GPIO 输入和输出。

分配成功后，*mcpwm_new_gpio_fault()* 将返回一个指向已分配故障的指针。否则，函数将返回错误代码。具体来说，当指定 MCPWM 组中没有空闲 GPIO 故障时，将返回*ESP_ERR_NOT_FOUND* 错误。Page 518, 1

调用函数*mcpwm_soft_fault_activate()* 使一个软件故障对象触发故障，无需等待来自 GPIO 的真实故障信号。调用*mcpwm_new_soft_fault()* 函数，以配置结构体*mcpwm_soft_fault_config_t* 为参数，分配一个软件故障为对象。该结构体暂时保留，供后续使用。

分配成功后，*mcpwm_new_soft_fault()* 将返回一个指向已分配故障的指针。否则，函数将返回错误代码。具体来说，当内存不足以支持该故障对象时，将返回*ESP_ERR_NO_MEM* 错误。虽然软件故障和 GPIO 故障是不同类型的故障，但返回的故障句柄为同一类型。

反之，调用*mcpwm_del_fault()* 函数将释放已分配的故障。此函数同时适用于软件故障和 GPIO 故障。

MCPWM 同步源 同步源用于同步 MCPWM 定时器和 MCPWM 捕获定时器，分为三种类型：来自 GPIO 的同步源、软件生成的同步源和 MCPWM 定时器事件生成的同步源。

调用*mcpwm_new_gpio_sync_src()* 函数，以配置结构体*mcpwm_gpio_sync_src_config_t* 为参数，分配一个 GPIO 同步源。结构体定义为：

- *mcpwm_gpio_sync_src_config_t::group_id* 指定 MCPWM 组 ID，范围为 [0, *SOC_MCPWM_GROUPS* - 1]。需注意，位于不同组的 GPIO 同步源彼此独立，也就是说，1 组的定时器无法检测到 0 组的 GPIO 同步源。
- *mcpwm_gpio_sync_src_config_t::gpio_num* 设置同步源使用的 GPIO 编号。
- *mcpwm_gpio_sync_src_config_t::active_neg* 设置同步信号在下降沿是否有效。
- *mcpwm_gpio_sync_src_config_t::pull_up* 和*mcpwm_gpio_sync_src_config_t::pull_down* 设置是否在内部拉高和/或拉低 GPIO。
- *mcpwm_gpio_sync_src_config_t::io_loop_back* 设置是否启用回环模式。该模式仅用于调试，使用 GPIO 交换矩阵外设同时启用 GPIO 输入和输出。

分配成功后，*mcpwm_new_gpio_sync_src()* 将返回一个指向已分配同步源的指针。否则，函数将返回错误代码。具体来说，当 MCPWM 组中没有空闲 GPIO 时钟源时，将返回*ESP_ERR_NOT_FOUND* 错误。Page 518, 1

调用*mcpwm_new_timer_sync_src()* 函数，以配置结构体*mcpwm_timer_sync_src_config_t* 为参数，分配一个定时器事件同步源。结构体定义为：

- *mcpwm_timer_sync_src_config_t::timer_event* 指定产生同步信号的定时器事件。
- *mcpwm_timer_sync_src_config_t::propagate_input_sync* 是否广播输入同步信号（即将输入同步信号传输到其同步输出）。

分配成功后，*mcpwm_new_timer_sync_src()* 将返回一个指向已分配同步源的指针。否则，函数将返回错误代码。具体来说，若是分配的同步源此前已分配给了同一个定时器，将返回*ESP_ERR_INVALID_STATE* 错误。

也可以调用*mcpwm_new_soft_sync_src()* 函数，以配置结构体*mcpwm_soft_sync_config_t* 为参数，分配一个软件同步源。该结构体暂时保留，供后续使用。

分配成功后，*mcpwm_new_soft_sync_src()* 将返回一个指向已分配同步源的指针。否则，函数将返回错误代码。具体来说，当内存不足以支持分配的同步源时，将返回*ESP_ERR_NO_MEM* 错误。需注意，为确保软件同步源能够正常工作，应预先调用*mcpwm_soft_sync_activate()*。

相反，调用*mcpwm_del_sync_src()* 函数将释放分配的同步源对象。此函数适用于所有类型的同步源。

MCPWM 捕获定时器和通道 MCPWM 组有一个专用定时器，用于捕获特定事件发生时的时间戳。捕获定时器连接了数个独立通道，每个通道都分配了各自的 GPIO。

调用*mcpwm_new_capture_timer()* 函数，以配置结构体*mcpwm_capture_timer_config_t* 为参数，分配一个捕获定时器。结构体定义为：

- *mcpwm_capture_timer_config_t::group_id* 设置 MCPWM 组 ID，范围为 [0, *SOC_MCPWM_GROUPS* - 1]。
- *mcpwm_capture_timer_config_t::clk_src* 设置捕获定时器的时钟源。
- *mcpwm_capture_timer_config_t::resolution_hz* 设置捕获定时器的预期分辨率。内部驱动将根据时钟源和分辨率设置合适的分频器。设置为 0 时，驱动会自己选取一个适当的分辨率，后续你可以通过*mcpwm_capture_timer_get_resolution()* 查看当前定时器的分辨率。

分配成功后，*mcpwm_new_capture_timer()* 将返回一个指向已分配捕获定时器的指针。否则，函数将返回错误代码。具体来说，当 MCPWM 组中没有空闲捕获定时器时，将返回*ESP_ERR_NOT_FOUND* 错误。Page 518, 1

接下来，可以调用*mcpwm_new_capture_channel()* 函数，以一个捕获定时器句柄和配置结构体*mcpwm_capture_channel_config_t* 为参数，分配一个捕获通道。结构体定义为：

- *mcpwm_capture_channel_config_t::intr_priority* 设置中断的优先级。如果设置为 0，则会分配一个默认优先级的中断，否则会使用指定的优先级。
- *mcpwm_capture_channel_config_t::gpio_num* 设置捕获通道使用的 GPIO 编号。
- *mcpwm_capture_channel_config_t::prescale* 设置输入信号的预分频器。
- *mcpwm_capture_channel_config_t::extra_capture_channel_flags::pos_edge* 和*mcpwm_capture_channel_config_t::extra_capture_channel_flags::neg_edge* 设置是否在输入信号的上升沿和/或下降沿捕获时间戳。

- *mcpwm_capture_channel_config_t::extra_capture_channel_flags::pull_up* 和*mcpwm_capture_channel_config_t::extra_capture_channel_flags::pull_down* 设置是否在内部拉高和/或拉低 GPIO。
- *mcpwm_capture_channel_config_t::extra_capture_channel_flags::invert_cap_signal* 设置是否取反捕获信号。
- *mcpwm_capture_channel_config_t::extra_capture_channel_flags::io_loop_back* 设置是否启用回环模式。该模式仅用于调试，使用 GPIO 交换矩阵外设同时启用 GPIO 输入和输出。
- *mcpwm_capture_channel_config_t::extra_capture_channel_flags::keep_io_conf_at_exit* 设置是否在删除通道时保留 GPIO 的相关配置。

分配成功后，*mcpwm_new_capture_channel()* 将返回一个指向已分配捕获通道的指针。否则，函数将返回错误代码。具体来说，当捕获定时器中没有空闲捕获通道时，将返回*ESP_ERR_NOT_FOUND* 错误。

反之，调用*mcpwm_del_capture_channel()* 和*mcpwm_del_capture_timer()* 将释放已分配的捕获通道和定时器。

MCPWM 中断优先级 MCPWM 允许为定时器、操作器、比较器、故障以及捕获事件分别配置中断，中断优先级由各自的 `config_t::intr_priority` 决定。且同一个 MCPWM 组中的事件共享同一个中断源。注册多个中断事件时，中断优先级需要保持一致。

备注： MCPWM 组注册多个中断事件时，驱动将以第一个事件的中断优先级作为 MCPWM 组的中断优先级。

定时器操作和事件

更新定时器周期 定时器周期在创建定时器时就已经通过*mcpwm_timer_config_t::period_ticks* 被初始化过了。你还可以在运行期间，调用*mcpwm_timer_set_period()* 函数来更新定时周期。新周期的生效时机由*mcpwm_timer_config_t::update_period_on_empty* 和*mcpwm_timer_config_t::update_period_on_sync* 共同决定。如果他们两个参数都是 `false`，那么新的定时周期会立即生效。

注册定时器事件回调 MCPWM 定时器运行时会生成不同的事件。若有函数需在特定事件发生时调用，则应预先调用*mcpwm_timer_register_event_callbacks()*，将所需函数挂载至中断服务程序 (ISR) 中。驱动中定时器回调函数原型声明为*mcpwm_timer_event_cb_t*，其所支持的事件回调类型则列在*mcpwm_timer_event_callbacks_t* 中：

- *mcpwm_timer_event_callbacks_t::on_full* 设置定时器计数达到峰值时的回调函数。
- *mcpwm_timer_event_callbacks_t::on_empty* 设置定时器计数为零时的回调函数。
- *mcpwm_timer_event_callbacks_t::on_stop* 设置定时器停止时的回调函数。

由于上述回调函数是在 ISR 中调用的，因此，这些函数 **不应**涉及 block 操作。可以检查调用 API 的后缀，确保在函数中只调用了后缀为 `ISR` 的 FreeRTOS API。

函数*mcpwm_timer_register_event_callbacks()* 中的 `user_data` 参数用于保存用户上下文，将直接传递至各个回调函数。

此函数会在不启用 MCPWM 定时器的情况下延迟安装其中断服务。因此，需在调用*mcpwm_timer_enable()* 函数前调用该函数，否则将返回*ESP_ERR_INVALID_STATE* 错误。更多信息请参见*启用和禁用定时器*。

启用和禁用定时器 在对定时器进行 IO 控制前，需要预先调用*mcpwm_timer_enable()* 函数启用定时器。这个函数将：

- 将定时器的状态从 **init** 切换到 **enable**。
- 若中断服务此前已通过*mcpwm_timer_register_event_callbacks()* 函数延迟安装，则启用中断服务。

- 若选择了特定时钟源（例如 PLL_160M 时钟），则获取相应的电源管理锁。更多信息请参见电源管理。

反之，调用*mcpwm_timer_disable()* 会将定时器切换回 **init** 状态、禁用中断服务并释放电源管理锁。

启动和停止定时器 通过基本的 IO 控制，即可启动和停止定时器。使用不同的*mcpwm_timer_start_stop_cmd_t* 命令调用*mcpwm_timer_start_stop()* 便可立即启动定时器，或在发生特定事件时停止定时器。此外，还可以通过配置，让定时器仅计数一轮。也就是说，在计数达到峰值或零后，定时器自行停止。

连接定时器和操作器 调用*mcpwm_operator_connect_timer()* 函数，连接分配的 MCPWM 定时器和 MCPWM 操作器。连接后，操作器即可将定时器作为时基，生成所需的 PWM 波形。需注意，MCPWM 定时器和操作器必须位于同一个组中。否则，将返回*ESP_ERR_INVALID_ARG* 错误。

比较器操作和事件

注册比较器事件回调 MCPWM 比较器可以在定时器计数器等于比较值时发送通知。若有函数需在比较事件发生时调用，则应预先调用*mcpwm_comparator_register_event_callbacks()*，将所需函数挂载至中断服务程序 (ISR) 中。驱动中比较器回调函数原型声明为*mcpwm_compare_event_cb_t*，其所支持的事件回调类型则列在*mcpwm_comparator_event_callbacks_t* 中：

- *mcpwm_comparator_event_callbacks_t::on_reach* 设置当定时器计数器等于比较值时的比较器回调函数。

回调函数会提供类型为*mcpwm_compare_event_data_t* 的事件特定数据。由于上述回调函数是在 ISR 中调用的，因此，这些函数 **不应**涉及 block 操作。可以检查调用 API 的后缀，确保在函数中只调用了后缀为 `ISR` 的 FreeRTOS API。

函数*mcpwm_comparator_register_event_callbacks()* 中的 `user_data` 参数用于保存用户上下文，将直接传递至各个回调函数。

此函数会延迟安装 MCPWM 比较器的中断服务。中断服务只能通过*mcpwm_del_comparator* 移除。

备注： 对于事件比较器，你无法通过该函数来注册回调函数，因为事件比较器触发产生任何中断事件。

设置比较值 运行 MCPWM 比较器时，可以调用*mcpwm_comparator_set_compare_value()* 设置比较值。需注意以下几点：

- 重新设置的比较值可能不会立即生效。比较值的更新时间通过*mcpwm_comparator_config_t::update_cmp_on_tez* 或*mcpwm_comparator_config_t::update_cmp_on_tep* 或*mcpwm_comparator_config_t::update_cm* 配置。
- 请确保已经预先调用*mcpwm_operator_connect_timer()* 将操作器连接至 MCPWM 定时器。否则，将返回*ESP_ERR_INVALID_STATE* 错误。
- 比较值不应超过定时器的计数峰值。否则，将无法触发比较事件。

生成器对事件执行的操作

设置生成器对定时器事件执行的操作 调用*mcpwm_generator_set_actions_on_timer_event()* 并辅以若干操作配置，可以针对不同的定时器事件，为生成器设置不同的操作。操作配置定义在*mcpwm_gen_timer_event_action_t* 中：

- *mcpwm_gen_timer_event_action_t::direction* 指定定时器计数方向，可以调用*mcpwm_timer_direction_t* 查看支持的方向。

- *mcpwm_gen_timer_event_action_t::event* 指定定时器事件，可以调用*mcpwm_timer_event_t* 查看支持的定时器事件。
- *mcpwm_gen_timer_event_action_t::action* 指定随即进行的生成器操作，可以调用*mcpwm_generator_action_t* 查看支持的操作。

可借助辅助宏*MCPWM_GEN_TIMER_EVENT_ACTION* 构建定时器事件操作条目。

需注意，*mcpwm_generator_set_actions_on_timer_event()* 的参数列表**必须**以*MCPWM_GEN_TIMER_EVENT_ACTION_END* 结束。

也可以调用*mcpwm_generator_set_action_on_timer_event()* 逐一设置定时器操作，无需涉及变量参数。

设置生成器对比较器事件执行的操作 调用*mcpwm_generator_set_actions_on_compare_event()* 并辅以若干操作配置，可以针对不同的比较器事件，为生成器设置不同的操作。操作配置定义在*mcpwm_gen_compare_event_action_t* 中：

- *mcpwm_gen_compare_event_action_t::direction* 指定定时器计数方向，可以调用*mcpwm_timer_direction_t* 查看支持的方向。
- *mcpwm_gen_compare_event_action_t::comparator* 指定比较器句柄。有关分配比较器的方法，请参见*MCPWM 比较器*。
- *mcpwm_gen_compare_event_action_t::action* 指定随即进行的生成器操作，可以调用*mcpwm_generator_action_t* 查看支持的操作。

可借助辅助宏*MCPWM_GEN_COMPARE_EVENT_ACTION* 构建比较事件操作条目。

需注意，*mcpwm_generator_set_actions_on_compare_event()* 的参数列表**必须**以*MCPWM_GEN_COMPARE_EVENT_ACTION_END* 结束。

也可以调用*mcpwm_generator_set_action_on_compare_event()* 逐一设置比较器操作，无需涉及变量参数。

设置生成器对故障事件执行的操作 调用*mcpwm_generator_set_action_on_fault_event()* 并辅以操作配置，可以针对故障事件，为生成器设置操作。操作配置定义在*mcpwm_gen_fault_event_action_t* 中：

- *mcpwm_gen_fault_event_action_t::direction* 指定定时器计数方向，可以调用*mcpwm_timer_direction_t* 查看支持的方向。
- *mcpwm_gen_fault_event_action_t::fault* 指定用于触发器的故障。有关分配故障的方法，请参见*MCPWM 故障*。
- *mcpwm_gen_fault_event_action_t::action* 指定随即进行的生成器操作，可以调用*mcpwm_generator_action_t* 查看支持的操作。

当生成器所属的操作器中没有空闲触发器时，将返回*ESP_ERR_NOT_FOUND* 错误。Page 518, 1

触发器支持的故障仅为 GPIO 故障，当传入故障不为 GPIO 故障时，将返回*ESP_ERR_NOT_SUPPORTED* 错误。

可借助辅助宏*MCPWM_GEN_FAULT_EVENT_ACTION* 构建触发事件操作条目。

需注意，故障事件没有类似 `mcpwm_generator_set_actions_on_fault_event()` 这样的可变参数函数。

设置生成器对同步事件执行的操作 调用*mcpwm_generator_set_action_on_sync_event()* 并辅以操作配置，可以针对同步事件，为生成器设置操作。操作配置定义在*mcpwm_gen_sync_event_action_t* 中：

- *mcpwm_gen_sync_event_action_t::direction* 指定定时器计数方向，可以调用*mcpwm_timer_direction_t* 查看支持的方向。
- *mcpwm_gen_sync_event_action_t::sync* 指定用于触发器的同步源。有关分配同步源的方法，请参见*MCPWM 同步源*。

- *mcpwm_gen_sync_event_action_t::action* 指定随即进行的生成器操作，可以调用*mcpwm_generator_action_t* 查看支持的操作。

当生成器所属的操作器中没有空闲触发器时，将返回*ESP_ERR_NOT_FOUND* 错误。Page 518, 1

无论同步为何种类型，触发器仅支持一种同步操作，如果多次设置同步操作，将返回*ESP_ERR_INVALID_STATE* 错误。

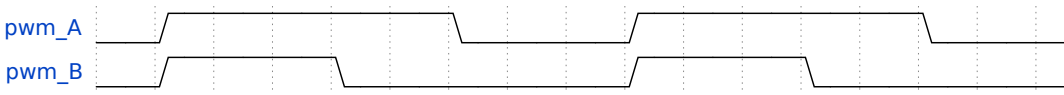
可借助辅助宏*MCPWM_GEN_SYNC_EVENT_ACTION* 构建触发事件操作条目。

需注意，同步事件没有类似 mcpwm_generator_set_actions_on_sync_event() 这样的可变参数函数。

经典 PWM 波形的生成器配置 本节提供了一些生成器支持生成的经典 PWM 波形，同时提供用于生成这些波形的代码片段。总的来说：

- 生成波形为 **对称波形**还是 **不对称波形**取决于 MCPWM 定时器的计数模式。
- 波形对的 **激活电平**取决于占空比较小的 PWM 波形的电平。
- PWM 波形的周期取决于定时器的周期和计数模式。
- PWM 波形的占空比取决于生成器的各种操作配置组合。

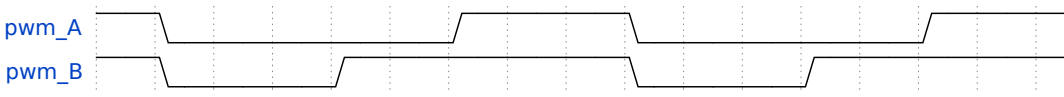
Single Edge Asymmetric Waveform, Active High



单边不对称波形 - 高电平

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_LOW)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(genb,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(genb,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,␣
↪MCPWM_GEN_ACTION_LOW)));
}
```

Single Edge Asymmetric Waveform, Active Low



单边不对称波形 - 低电平

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_FULL, MCPWM_GEN_ACTION_LOW)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(genb,
```

(下页继续)

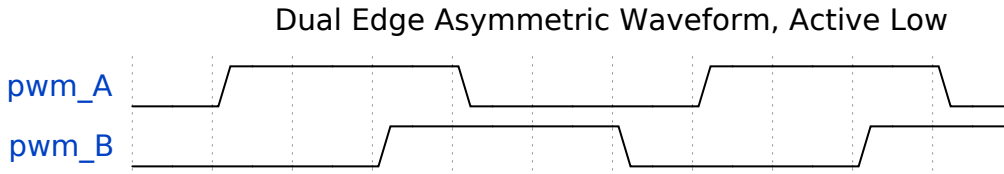
(续上页)

```
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_FULL, MCPWM_GEN_ACTION_LOW)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(genb,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,␣
↪MCPWM_GEN_ACTION_HIGH)));
}
```

脉冲位置不对称波形

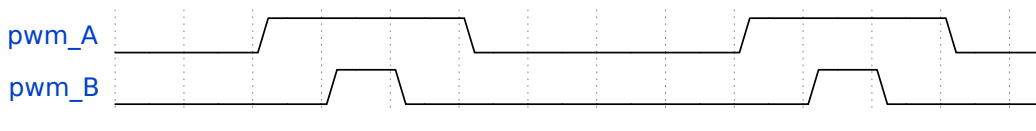
```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,␣
↪MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(genb,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_TOGGLE),
                    MCPWM_GEN_TIMER_EVENT_ACTION_END()));
}
```

双沿不对称波形 - 低电平有效

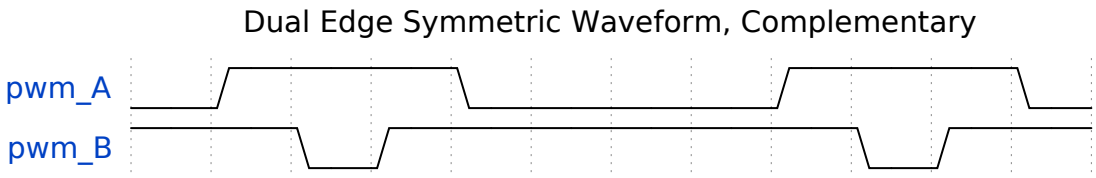



```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN,␣
↪cmpb, MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(genb,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, MCPWM_
↪TIMER_EVENT_FULL, MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_TIMER_EVENT_ACTION_END()));
}
```

双沿对称波形 - 低电平有效



```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN,␣
↪cmpa, MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,␣
↪MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN,␣
↪cmpb, MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}
```



双沿对称波形 - 互补

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN,␣
↪cmpa, MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,␣
↪MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN,␣
↪cmpb, MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}
```

死区 在电力电子学中，常常会用到整流器和逆变器，这就涉及到了整流桥和逆变桥的应用。每个桥臂配有两个功率电子器件，例如 MOSFET、IGBT 等。同一桥臂上的两个 MOSFET 不能同时导通，否则会造成短路。实际应用中，在 PWM 波形显示 MOSFET 开关已关闭后，仍需要一段时间窗口才能完全关闭 MOSFET。因此，需要设置生成器对事件执行的操作，在已生成的 PWM 波形上添加额外延迟。

死区驱动器的工作方式与 **装饰器**类似。在*mcpwm_generator_set_dead_time()* 函数的参数中，驱动接收主要生成器句柄 (in_generator)，并在应用死区后返回一个新的生成器 (out_generator)。需注意，如果 out_generator 和 in_generator 相同，这表示 PWM 波形中的时间延迟是以 “就地” 的方式添加的。反之，如果 out_generator 和 in_generator 不同，则代表在原 in_generator 的基础上派生出了一个新的 PWM 波形。

结构体*mcpwm_dead_time_config_t* 中列出了死区相关的具体配置：

- *mcpwm_dead_time_config_t::posedge_delay_ticks* 和*mcpwm_dead_time_config_t::negedge_de*
  设置 PWM 波形上升沿和下降沿上的延迟时间，以 Tick 为单位。若将这两个参数设置为 0，则代表绕过死区模块。死区的 Tick 分辨率与通过*mcpwm_operator_connect_timer()* 连接操作器的定时器相同。

- *mcpwm_dead_time_config_t::invert_output* 设置是否在应用死区后取反信号，以控制延迟边沿的极性。

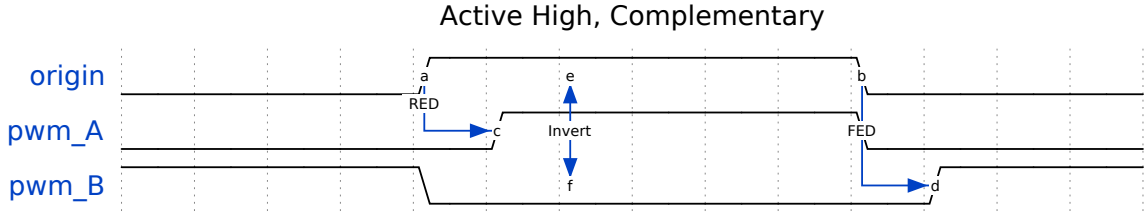
警告： 由于硬件限制，同一种 delay 模块（posedge delay 或者 negedge delay）不能同时被应用在不同的 MCPWM 生成器中。例如，以下配置是无效的：

```
mcpwm_dead_time_config_t dt_config = {
    .posedge_delay_ticks = 10,
};
// 给 generator A 叠加上升沿 delay
mcpwm_generator_set_dead_time(mcpwm_gen_a, mcpwm_gen_a, &dt_config);
// NOTE: 下面的操作是无效的，不能将同一种 delay 应用于不同的 generator 上
mcpwm_generator_set_dead_time(mcpwm_gen_b, mcpwm_gen_b, &dt_config);
```

然而，你可以为生成器 A 设置 posedge delay，为生成器 B 设置 negedge delay。另外，也可以为生成器 A 同时设置 posedge delay 和 negedge delay，而让生成器 B 绕过死区模块。

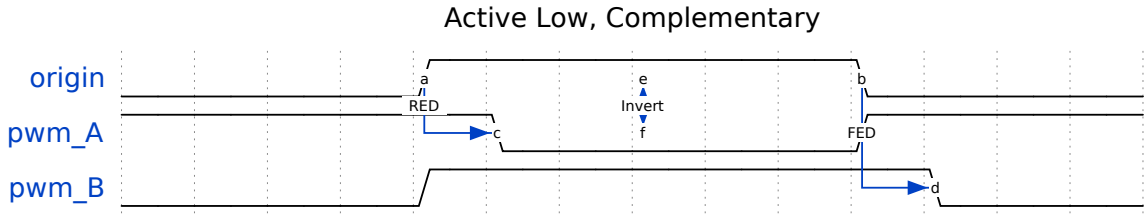
备注： 也可以通过设置生成器对事件执行的操作 来生成所需的死区，通过不同的比较器来控制边沿位置。但是，如果需要使用经典的基于边沿延迟并附带极性控制的死区，则应使用死区子模块。

经典 PWM 波形的死区配置 本节提供了一些死区子模块支持生成的经典 PWM 波形，同时在图片下方提供用于生成这些波形的代码片段。



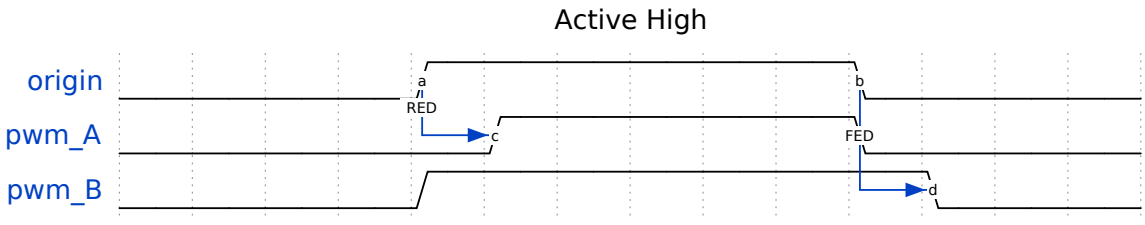
高电平有效互补

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_LOW)));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 50,
        .negedge_delay_ticks = 0
    };
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    dead_time_config.posedge_delay_ticks = 0;
    dead_time_config.negedge_delay_ticks = 100;
    dead_time_config.flags.invert_output = true;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}
```



低电平有效互补

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_LOW)));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 50,
        .negedge_delay_ticks = 0,
        .flags.invert_output = true
    };
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    dead_time_config.posedge_delay_ticks = 0;
    dead_time_config.negedge_delay_ticks = 100;
    dead_time_config.flags.invert_output = false;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}
```



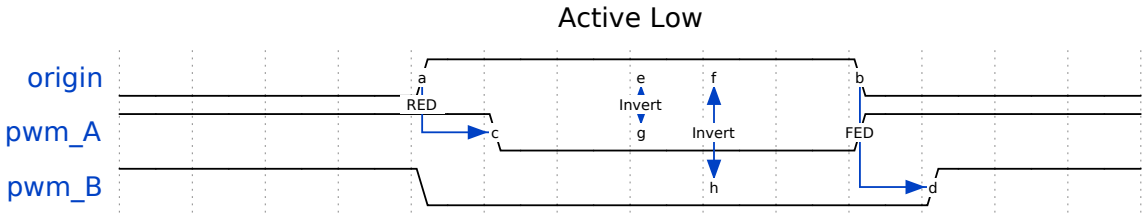
高电平有效

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_LOW)));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 50,
        .negedge_delay_ticks = 0,
    };
```

(下页继续)

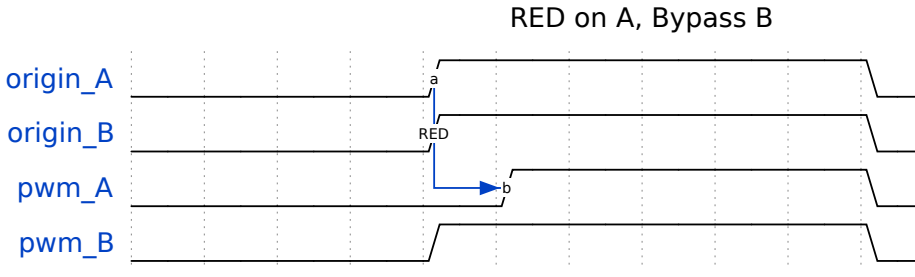
(续上页)

```
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    dead_time_config.posedge_delay_ticks = 0;
    dead_time_config.negedge_delay_ticks = 100;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}
```



低电平有效

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_LOW)));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 50,
        .negedge_delay_ticks = 0,
        .flags.invert_output = true
    };
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    dead_time_config.posedge_delay_ticks = 0;
    dead_time_config.negedge_delay_ticks = 100;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}
```



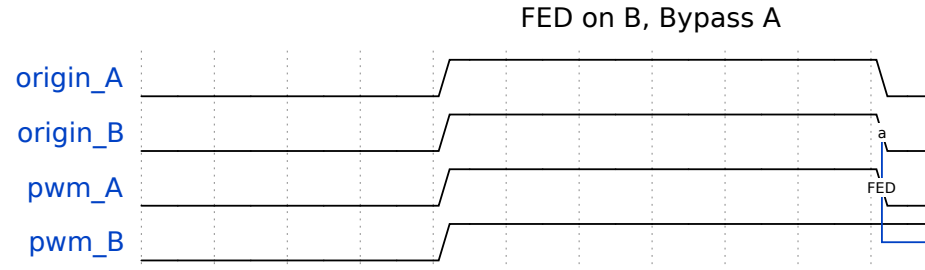
PWMA 上升沿延迟，绕过 PWMB 死区

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_LOW)));
```

(下页继续)

(续上页)

```
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(genb,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(genb,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,␣
↪MCPWM_GEN_ACTION_LOW)));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 50,
        .negedge_delay_ticks = 0,
    };
    // apply deadtime to generator_a
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    // bypass deadtime module for generator_b
    dead_time_config.posedge_delay_ticks = 0;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(genb, genb, &dead_time_config));
}
```



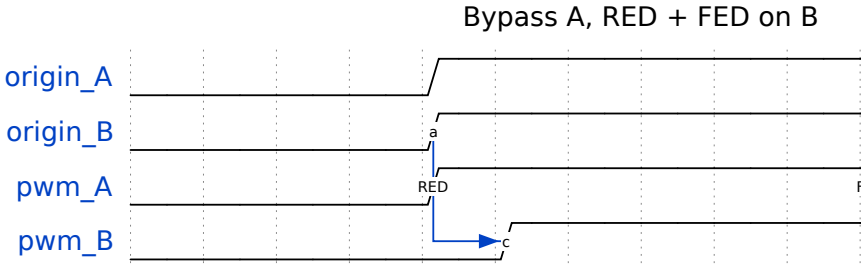
PWMB 下降沿延迟，绕过 PWMA 死区

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_LOW)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(genb,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(genb,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,␣
↪MCPWM_GEN_ACTION_LOW)));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 0,
        .negedge_delay_ticks = 0,
    };
    // generator_a bypass the deadtime module (no delay)
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    // apply dead time to generator_b
    dead_time_config.negedge_delay_ticks = 50;
```

(下页继续)

(续上页)

```
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(genb, genb, &dead_time_config));

}
```



PWMB 上升下降沿延迟，绕过 PWMA 死区

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,␣
↪mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gena,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,␣
↪MCPWM_GEN_ACTION_LOW)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(genb,
                    MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
↪TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
    ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(genb,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,␣
↪MCPWM_GEN_ACTION_LOW)));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 0,
        .negedge_delay_ticks = 0,
    };
    // generator_a bypass the deadtime module (no delay)
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    // apply dead time on both edge for generator_b
    dead_time_config.negedge_delay_ticks = 50;
    dead_time_config.posedge_delay_ticks = 50;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(genb, genb, &dead_time_config));
}
```

载波调制 MCPWM 操作器具有载波子模块，可以根据需要（例如隔离式数字电源应用中）使用变压器传递 PWM 输出信号，实现电机驱动器的电气隔离。在电机需要在全负荷下稳定运行时，各个 PWM 输出信号都将占空比稳定保持在 100% 左右。由于变压器无法直接耦合非交替信号，需要使用载波子模块调制信号，生成交流电波形，从而实现耦合。

调用*mcpwm_operator_apply_carrier()*，并提供配置结构体*mcpwm_carrier_config_t*，配置载波子模块：

- *mcpwm_carrier_config_t::clk_src* 设置载波的时钟源。
- *mcpwm_carrier_config_t::frequency_hz* 表示载波频率，单位为赫兹。内部驱动将根据时钟源和载波频率设置合适的分频器。
- *mcpwm_carrier_config_t::duty_cycle* 表示载波的占空比。需注意，支持的占空比选项并不连续，驱动程序将根据配置查找最接近的占空比。

- *mcpwm_carrier_config_t::first_pulse_duration_us* 表示第一个脉冲的脉宽，单位为微秒。该脉冲的分辨率由*mcpwm_carrier_config_t::frequency_hz* 中的配置决定。第一个脉冲的脉宽不能为零，且至少为一个载波周期。脉宽越长，电感传导越快。
- *mcpwm_carrier_config_t::invert_before_modulate* 和*mcpwm_carrier_config_t::invert_after* 设置是否在调制前和调制后取反载波输出。

具体而言，可调用*mcpwm_operator_apply_carrier()* 并将其配置为 NULL，禁用载波子模块。

故障检测和制动控制 MCPWM 操作器能够感知外部信号，接收有关电机故障、功率驱动器及其他连接设备的信息。这些故障信号封装在 MCPWM 故障对象中。

电机需配置故障模式以及检测到特定故障时的对应操作，例如拉低有刷电机的所有输出，或是锁定步进电机的电流状态等。此操作应使电机重回安全状态，降低故障导致损坏的可能性。

设置故障时操作器的制动模式 MCPWM 操作器对故障的响应方式为 **制动**。可以调用*mcpwm_operator_set_brake_on_fault()*，为每个故障对象配置不同的制动模式。制动的相关配置包含在结构体*mcpwm_brake_config_t* 中：

- *mcpwm_brake_config_t::fault* 设置操作器响应的故障类型。
- *mcpwm_brake_config_t::brake_mode* 设置对应故障的制动模式，可以调用*mcpwm_operator_brake_mode_t* 查看支持的制动模式。在*MCPWM_OPER_BRAKE_MODE_CBC* 模式下，操作器将在故障消失后自行恢复正常，可以通过*mcpwm_brake_config_t::cbc_recover_on_tez* 和*mcpwm_brake_config_t::cbc_recover_on_tep* 配置恢复时间。在*MCPWM_OPER_BRAKE_MODE_OST* 模式下，即使故障消失，操作器也无法恢复正常。此时，需要调用*mcpwm_operator_recover_from_fault()*，手动恢复操作器。

设置发生制动事件时的生成器操作 调用*mcpwm_generator_set_actions_on_brake_event()* 并辅以若干操作配置，可以针对不同的制动事件，为生成器设置不同的对应操作。操作配置定义在*mcpwm_gen_brake_event_action_t* 中：

- *mcpwm_gen_brake_event_action_t::direction* 指定定时器的方向，可以调用*mcpwm_timer_direction_t* 查看支持的方向。
- *mcpwm_gen_brake_event_action_t::brake_mode* 指定制动模式，可以调用*mcpwm_operator_brake_mode_t* 查看支持的制动模式。
- *mcpwm_gen_brake_event_action_t::action* 指定生成器操作，可以调用*mcpwm_generator_action_t* 查看支持的操作。

可借助辅助宏*MCPWM_GEN_BRAKE_EVENT_ACTION* 构建制动事件操作条目。

需注意，*mcpwm_generator_set_actions_on_brake_event()* 的参数列表 **必须** 以*MCPWM_GEN_BRAKE_EVENT_ACTION_END* 结束。

也可以调用*mcpwm_generator_set_action_on_brake_event()* 逐一设置制动操作，无需涉及变量参数。

注册故障事件回调 MCPWM 故障检测器支持在检测到实际故障或故障信号消失时发送通知。若有函数需在特定事件发生时调用，则应预先调用*mcpwm_fault_register_event_callbacks()*，将所需函数挂载至中断服务程序 (ISR) 中。驱动中故障事件回调函数原型声明为*mcpwm_fault_event_cb_t*，其所支持的事件回调类型则列在*mcpwm_fault_event_callbacks_t* 中：

- *mcpwm_fault_event_callbacks_t::on_fault_enter* 设置检测到故障时调用的回调函数。
- *mcpwm_fault_event_callbacks_t::on_fault_exit* 设置故障消失后调用的回调函数。

由于上述回调函数在 ISR 中调用，因此，这些函数 **不应** 涉及 block 操作。可以检查调用 API 的后缀，确保在函数中只调用了后缀为 ISR 的 FreeRTOS API。

函数*mcpwm_fault_register_event_callbacks()* 中的 user_data 参数用于保存用户上下文，将直接传递至各个回调函数。

此函数会延迟安装 MCPWM 故障的中断服务。中断服务只能通过*mcpwm_del_fault* 移除。

寄存器制动事件回调 MCPWM 操作器支持在进行制动操作前发送通知。若有函数需在特定事件发生时调用，则应预先调用*mcpwm_operator_register_event_callbacks()*，将所需函数挂载至中断服务程序 (ISR) 中。驱动中制动事件回调函数原型声明为*mcpwm_brake_event_cb_t*，其所支持的事件回调类型则列在*mcpwm_operator_event_callbacks_t* 中：

- *mcpwm_operator_event_callbacks_t::on_brake_cbc* 设置操作器进行 **逐周期 (CBC)** 操作前调用的回调函数。
- *mcpwm_operator_event_callbacks_t::on_brake_ost* 设置操作器进行 **一次性 (OST)** 操作前调用的回调函数。

由于上述回调函数在 ISR 中调用，因此，这些函数 **不应**涉及 block 操作。可以检查调用 API 的后缀，确保在函数中只调用了后缀为 `ISR` 的 FreeRTOS API。

函数*mcpwm_operator_register_event_callbacks()* 中的 `user_data` 参数用于保存用户上下文，将直接传递至各个回调函数。

此函数会延迟安装 MCPWM 故障的中断服务。中断服务只能通过*mcpwm_del_operator* 移除。

生成器强制操作 调用*mcpwm_generator_set_force_level()*，使能软件强制决定运行时的生成器输出电平。相较于通过*mcpwm_generator_set_actions_on_timer_event()* 配置的其他事件操作，软件强制事件优先级最高。

- 设置 `level` 为 -1，代表禁用强制操作，生成器的输出电平重新交由事件操作控制。
- 设置 `hold_on` 为 true，代表强制输出电平将保持不变，直到设置 `level` 为 -1 来移除该电平。
- 设置 `hole_on` 为 false，代表强制输出电平仅在短时间有效，随后发生的任何事件都可以改变该电平。

同步模块 MCPWM 定时器接收到同步信号后，定时器将强制进入一个预定义的 **相位**，该相位由计数值和计数方向共同决定。调用*mcpwm_timer_set_phase_on_sync()*，设置同步相位。同步相位配置定义在*mcpwm_timer_sync_phase_config_t* 结构体中：

- *mcpwm_timer_sync_phase_config_t::sync_src* 设置同步信号源。创建同步源对象的相关操作，请参见*MCPWM 同步源*。具体来说，当此参数设置为 `NULL` 时，驱动器将禁用 MCPWM 定时器的同步功能。
- *mcpwm_timer_sync_phase_config_t::count_value* 设置接收同步信号后加载至计数器的值。
- *mcpwm_timer_sync_phase_config_t::direction* 设置接收同步信号后的计数方向。

备注： 当 MCPWM 定时器在*MCPWM_TIMER_COUNT_MODE_UP_DOWN* 模式下工作时，需要特别注意。在该模式下，计数器范围 `[0 -> peak-1]` 属于 **递增**阶段，`[peak -> 1]` 属于 **递减**阶段。因此，如果你将*mcpwm_timer_sync_phase_config_t::count_value* 设置为零，则可能还需要将*mcpwm_timer_sync_phase_config_t::direction* 设置为*MCPWM_TIMER_DIRECTION_UP*。否则，计时器将继续维持递减阶段，计数值会下溢至峰值。

同理，*MCPWM 捕获定时器和通道* 也支持同步。调用*mcpwm_capture_timer_set_phase_on_sync()*，设置捕获定时器的同步相位。同步相位配置定义在*mcpwm_capture_timer_sync_phase_config_t* 结构体中：

- *mcpwm_capture_timer_sync_phase_config_t::sync_src* 设置同步信号源。关于如何创建一个同步源对象，请参见*MCPWM 同步源*。具体来说，当此参数设置为 `NULL` 时，驱动器将禁用 MCPWM 捕获定时器的同步功能。
- *mcpwm_capture_timer_sync_phase_config_t::count_value* 设置接收同步信号后加载至计数器的值。
- *mcpwm_capture_timer_sync_phase_config_t::direction* 设置接收同步信号后的计数方向。需注意，不同于 MCPWM 定时器，捕获定时器只支持*MCPWM_TIMER_DIRECTION_UP* 这一个计数方向。

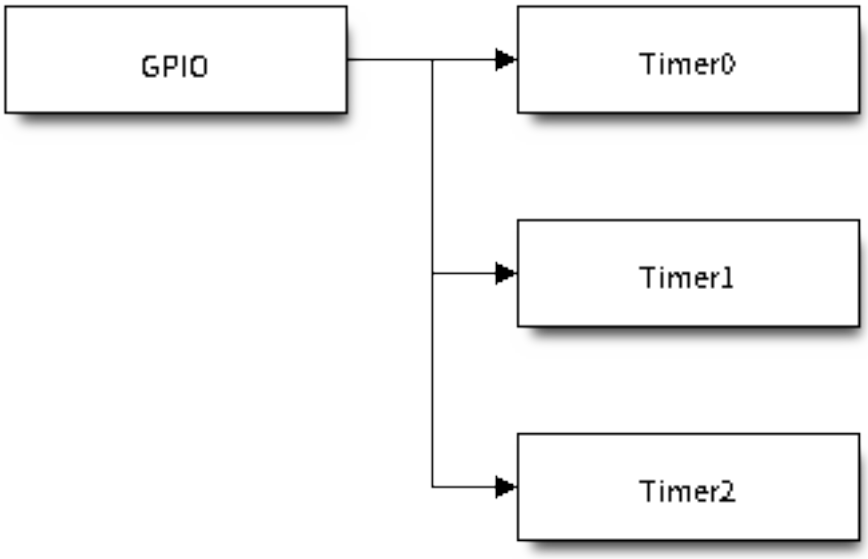


图 18: GPIO Sync All MCPWM Timers

使用 GPIO 同步定时器

```
static void example_setup_sync_strategy(mcpwm_timer_handle_t timers[])
{
    mcpwm_sync_handle_t gpio_sync_source = NULL;
    mcpwm_gpio_sync_src_config_t gpio_sync_config = {
        .group_id = 0,              // GPIO 故障应与以上定时器位于同一组中
        .gpio_num = EXAMPLE_SYNC_GPIO,
        .flags.pull_down = true,
        .flags.active_neg = false,  //␣
↪默认情况下，一个上升沿脉冲可以触发一个同步事件
    };
    ESP_ERROR_CHECK(mcpwm_new_gpio_sync_src(&gpio_sync_config, &gpio_sync_source));

    mcpwm_timer_sync_phase_config_t sync_phase_config = {
        .count_value = 0,                       // 同步相位：目标计数值
        .direction = MCPWM_TIMER_DIRECTION_UP, // 同步相位：计数方向
        .sync_src = gpio_sync_source,           // 同步源
    };
    for (int i = 0; i < 3; i++) {
        ESP_ERROR_CHECK(mcpwm_timer_set_phase_on_sync(timers[i], &sync_phase_
↪config));
    }
}
```

捕获模块 MCPWM 捕获的主要功能是记录捕获信号的脉冲边沿的有效时间。可以通过捕获得到脉宽，随后使用捕获回调函数将脉宽转换为其他物理量，如距离或速度。例如，在下图的无刷直流电机 (BLDC) 方案中，可以使用捕获子模块来确认来自霍尔传感器的转子位置。

通常，捕获定时器连接了数个捕获通道。有关资源分配的相关信息，请参见*MCPWM 捕获定时器和通道*。

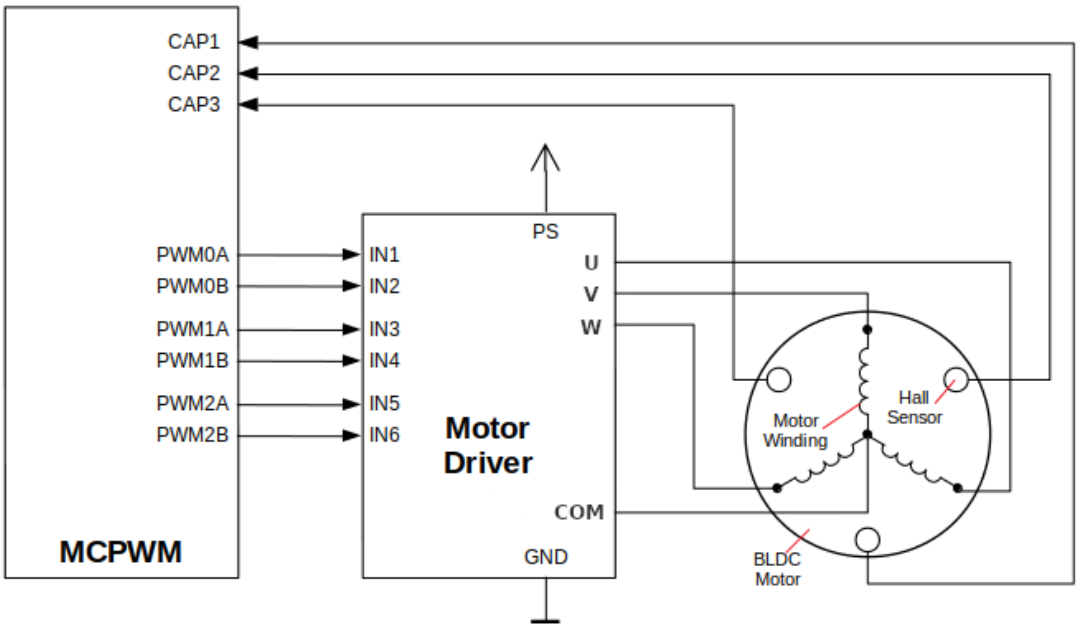


图 19: 带霍尔传感器的 MCPWM 无刷直流电机

注册捕获事件回调 MCPWM 捕获通道支持在信号上检测到有效边沿时发送通知。须调用*mcpwm_capture_channel_register_event_callbacks()*，注册一个回调函数来获得捕获的定时器计数值。回调函数原型声明在*mcpwm_capture_event_cb_t* 中，可以调用*mcpwm_capture_event_callbacks_t* 查看支持的捕获回调：

- *mcpwm_capture_event_callbacks_t::on_cap* 设置检测到有效边沿时捕获通道的回调函数。

回调函数会针对特定事件，提供*mcpwm_capture_event_data_t* 类型的数据，由此，可以通过*mcpwm_capture_event_data_t::cap_edge* 和*mcpwm_capture_event_data_t::cap_value* 分别得到捕获信号的边沿及该捕获的计数值。随后，调用*mcpwm_capture_timer_get_resolution()*，获取捕获定时器的分辨率，以将捕获计数转换为时间戳。

由于上述回调函数在 ISR 中调用，因此，这些函数 **不应**涉及 block 操作。可以检查调用 API 的后缀，确保在函数中只调用了后缀为 `ISR` 的 FreeRTOS API。

函数*mcpwm_capture_channel_register_event_callbacks()* 中的 `user_data` 参数用于保存用户上下文，将直接传递至各个回调函数。

此函数会延迟安装 MCPWM 捕获的中断服务。中断服务只能通过*mcpwm_del_capture_channel* 移除。

启用或禁用捕获通道 调用*mcpwm_new_capture_channel()* 进行分配后，捕获通道不会自动启用。应调用*mcpwm_capture_channel_enable()* 或*mcpwm_capture_channel_disable()* 来启用或禁用该通道。如果在为通道注册事件回调时，由于调用了*mcpwm_capture_channel_register_event_callbacks()*，致使延迟安装中断服务，则调用*mcpwm_capture_channel_enable()* 启用通道时，也将启用中断服务。

启用或禁用捕获定时器 在对捕获定时器进行 IO 控制之前，需要首先调用*mcpwm_capture_timer_enable()*，启用定时器。此函数将进行如下内部操作：

- 将捕获定时器的状态从 **init** 切换到 **enable**。

- 如果选择了一个特定时钟源（例如 APB 时钟），则获取一个对应的电源管理锁。更多信息请参见电源管理。

反之，调用*mcpwm_capture_timer_disable()* 将使定时器驱动程序切换回 **init** 状态，并释放电源管理锁。

启动或停止捕获定时器 通过基本的 IO 控制，即可启动或停止捕获定时器。调用*mcpwm_capture_timer_start()* 启动捕获定时器，或调用*mcpwm_capture_timer_stop()* 立即停止捕获定时器。

触发软件捕获事件 某些场景下，可能存在需要软件触发“虚假”捕获事件的需求。此时，可以调用*mcpwm_capture_channel_trigger_soft_catch()* 实现。需注意，此类“虚假”捕获事件仍然会触发中断，并从而调用捕获事件回调函数。

ETM 事件与任务 MCPWM 比较器可以产生事件，这些事件可以连接到*ETM* 模块。*mcpwm_comparator_etm_event_type_t* 中列出了 MCPWM 比较器能够产生的事件类型。用户可以通过调用*mcpwm_comparator_new_etm_event()* 来获得相应事件的 ETM event 句柄。

关于如何将 MCPWM 比较器事件连接到 ETM 通道中，请参阅*ETM* 文档。

电源管理 启用电源管理（即开启*CONFIG_PM_ENABLE*）时，系统会在进入 Light-sleep 前调整 PLL 和 APB 频率。该操作有可能会改变 MCPWM 定时器的计数步长，导致计时偏差。

不过，驱动程序可以获取*ESP_PM_NO_LIGHT_SLEEP* 类型的电源管理锁，防止系统进入 Light-sleep。每当驱动创建以 PLL 作为时钟源的 MCPWM 定时器实例时，都会在通过*mcpwm_timer_enable()* 启用定时器时获取电源管理锁。反之，调用*mcpwm_timer_disable()* 时，驱动程序释放锁。

同理，每当驱动创建 MCPWM 捕获定时器实例时，都会在通过*mcpwm_capture_timer_enable()* 启用定时器时获取电源管理锁，并在调用*mcpwm_capture_timer_disable()* 时释放锁。

IRAM 安全 默认情况下，禁用 cache 时，写入/擦除 flash 等原因将导致 MCPWM 中断延迟，事件回调函数也将延迟执行。在实时应用程序中，应避免此类情况。

因此，可以启用 Kconfig 选项*CONFIG_MCPWM_ISR_IRAM_SAFE*，该选项：

- 支持在禁用 cache 时启用所需中断
- 支持将 ISR 使用的所有函数存放在 IRAM 中[2]
- 支持将驱动程序存放在 DRAM 中（以防其意外映射到 PSRAM 中）

启用该选项可以保证 cache 禁用时的中断运行，但会相应增加 IRAM 占用。

另一个 Kconfig 选项*CONFIG_MCPWM_CTRL_FUNC_IN_IRAM* 也支持将常用的 IO 控制函数存放在 IRAM 中，以保证在禁用 cache 时可以正常使用函数。IO 控制函数如下所示：

- *mcpwm_comparator_set_compare_value()*
- *mcpwm_timer_set_period()*

线程安全 驱动程序会确保工厂函数（如*mcpwm_new_timer()*）的线程安全，使用时，可以直接从不同的 RTOS 任务中调用此类函数，无需额外锁保护。

驱动程序设置了临界区，以防函数同时在任务和 ISR 中调用。因此，以下函数支持在 ISR 上下文运行：

- *mcpwm_comparator_set_compare_value()*
- *mcpwm_timer_set_period()*

资源配置及初始化 中尚未提及的函数并非线程安全。在没有设置互斥锁保护的任务中，应避免调用这些函数。

[2] 回调函数及其调用的子函数需手动存放进 IRAM 中。

Kconfig 选项

- *CONFIG_MCPWM_ISR_IRAM_SAFE* 控制默认 ISR 处理程序能否在禁用 cache 的情况下工作。更多信息请参见*IRAM 安全*。
- *CONFIG_MCPWM_CTRL_FUNC_IN_IRAM* 控制 MCPWM 控制函数的存放位置（IRAM 或 flash）。更多信息请参见*IRAM 安全*。
- *CONFIG_MCPWM_ENABLE_DEBUG_LOG* 用于启用调试日志输出。启用此选项将增加固件的二进制文件大小。

应用示例

- 通过 PID 算法控制有刷直流电机速度：peripherals/mcpwm/mcpwm_bdc_speed_control
- 控制带霍尔传感器反馈的无刷直流电机：peripherals/mcpwm/mcpwm_bldc_hall_control
- 使用超声波传感器 (HC-SR04) 测量距离：peripherals/mcpwm/mcpwm_capture_hc_sr04
- 控制伺服电机角度：peripherals/mcpwm/mcpwm_servo_control
- 定时器之间的 MCPWM 同步：peripherals/mcpwm/mcpwm_sync

API Reference

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_timer.h
- This header file can be included with:

```
#include "driver/mcpwm_timer.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Functions

esp_err_t **mcpwm_new_timer**(const *mcpwm_timer_config_t* *config, *mcpwm_timer_handle_t* *ret_timer)

Create MCPWM timer.

参数

- **config** -- **[in]** MCPWM timer configuration
- **ret_timer** -- **[out]** Returned MCPWM timer handle

返回

- ESP_OK: Create MCPWM timer successfully
- ESP_ERR_INVALID_ARG: Create MCPWM timer failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM timer failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM timer failed because all hardware timers are used up and no more free one
- ESP_FAIL: Create MCPWM timer failed because of other error

esp_err_t **mcpwm_del_timer**(*mcpwm_timer_handle_t* timer)

Delete MCPWM timer.

参数 **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer()`

返回

- ESP_OK: Delete MCPWM timer successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM timer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete MCPWM timer failed because timer is not in init state
- ESP_FAIL: Delete MCPWM timer failed because of other error

esp_err_t **mcpwm_timer_set_period**(*mcpwm_timer_handle_t* timer, uint32_t period_ticks)

Set a new period for MCPWM timer.

备　注: If *mcpwm_timer_config_t::update_period_on_empty* and *mcpwm_timer_config_t::update_period_on_sync* are not set, the new period will take effect immediately. Otherwise, the new period will take effect when timer counts to zero or on sync event.

备注: You may need to use `mcpwm_comparator_set_compare_value` to set a new compare value for MCPWM comparator in order to keep the same PWM duty cycle.

参数

- **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer`
- **period_ticks** -- **[in]** New period in count ticks

返回

- ESP_OK: Set new period for MCPWM timer successfully
- ESP_ERR_INVALID_ARG: Set new period for MCPWM timer failed because of invalid argument
- ESP_FAIL: Set new period for MCPWM timer failed because of other error

esp_err_t **mcpwm_timer_enable**(*mcpwm_timer_handle_t* timer)

Enable MCPWM timer.

参数 **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer()`

返回

- ESP_OK: Enable MCPWM timer successfully
- ESP_ERR_INVALID_ARG: Enable MCPWM timer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable MCPWM timer failed because timer is enabled already
- ESP_FAIL: Enable MCPWM timer failed because of other error

esp_err_t **mcpwm_timer_disable**(*mcpwm_timer_handle_t* timer)

Disable MCPWM timer.

参数 **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer()`

返回

- ESP_OK: Disable MCPWM timer successfully
- ESP_ERR_INVALID_ARG: Disable MCPWM timer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable MCPWM timer failed because timer is disabled already
- ESP_FAIL: Disable MCPWM timer failed because of other error

esp_err_t **mcpwm_timer_start_stop**(*mcpwm_timer_handle_t* timer, *mcpwm_timer_start_stop_cmd_t* command)

Send specific start/stop commands to MCPWM timer.

参数

- **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer()`
- **command** -- **[in]** Supported command list for MCPWM timer

返回

- ESP_OK: Start or stop MCPWM timer successfully
- ESP_ERR_INVALID_ARG: Start or stop MCPWM timer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Start or stop MCPWM timer failed because timer is not enabled
- ESP_FAIL: Start or stop MCPWM timer failed because of other error

esp_err_t **mcpwm_timer_register_event_callbacks**(*mcpwm_timer_handle_t* timer, const *mcpwm_timer_event_callbacks_t* *cbs, void *user_data)

Set event callbacks for MCPWM timer.

备注: The first call to this function needs to be before the call to `mcpwm_timer_enable`

备注: User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

参数

- **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer()`
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_ERR_INVALID_STATE: Set event callbacks failed because timer is not in init state
- ESP_FAIL: Set event callbacks failed because of other error

esp_err_t **mcpwm_timer_set_phase_on_sync**(*mcpwm_timer_handle_t* timer, const *mcpwm_timer_sync_phase_config_t* *config)

Set sync phase for MCPWM timer.

参数

- **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer()`
- **config** -- **[in]** MCPWM timer sync phase configuration

返回

- ESP_OK: Set sync phase for MCPWM timer successfully
- ESP_ERR_INVALID_ARG: Set sync phase for MCPWM timer failed because of invalid argument
- ESP_FAIL: Set sync phase for MCPWM timer failed because of other error

Structures

struct **mcpwm_timer_event_callbacks_t**

Group of supported MCPWM timer event callbacks.

备注: The callbacks are all running under ISR environment

Public Members

mcpwm_timer_event_cb_t **on_full**

callback function when MCPWM timer counts to peak value

mcpwm_timer_event_cb_t **on_empty**

callback function when MCPWM timer counts to zero

mcpwm_timer_event_cb_t **on_stop**

callback function when MCPWM timer stops

struct **mcpwm_timer_config_t**

MCPWM timer configuration.

Public Members

int **group_id**

Specify from which group to allocate the MCPWM timer

mcpwm_timer_clock_source_t **clk_src**

MCPWM timer clock source

uint32_t **resolution_hz**

Counter resolution in Hz The step size of each count tick equals to (1 / resolution_hz) seconds

mcpwm_timer_count_mode_t **count_mode**

Count mode

uint32_t **period_ticks**

Number of count ticks within a period

int **intr_priority**

MCPWM timer interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **update_period_on_empty**

Whether to update period when timer counts to zero

uint32_t **update_period_on_sync**

Whether to update period on sync event

struct *mcpwm_timer_config_t*::[anonymous] **flags**

Extra configuration flags for timer

struct **mcpwm_timer_sync_phase_config_t**

MCPWM Timer sync phase configuration.

Public Members

mcpwm_sync_handle_t **sync_src**

The sync event source. Set to NULL will disable the timer being synced by others

uint32_t **count_value**

The count value that should lock to upon sync event

mcpwm_timer_direction_t **direction**

The count direction that should lock to upon sync event

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_oper.h
- This header file can be included with:

```
#include "driver/mcpwm_oper.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Functions

esp_err_t **mcpwm_new_operator**(const *mcpwm_operator_config_t* *config, *mcpwm_oper_handle_t* *ret_oper)

Create MCPWM operator.

参数

- **config** -- **[in]** MCPWM operator configuration
- **ret_oper** -- **[out]** Returned MCPWM operator handle

返回

- ESP_OK: Create MCPWM operator successfully
- ESP_ERR_INVALID_ARG: Create MCPWM operator failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM operator failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM operator failed because can't find free resource
- ESP_FAIL: Create MCPWM operator failed because of other error

esp_err_t **mcpwm_del_operator**(*mcpwm_oper_handle_t* oper)

Delete MCPWM operator.

参数 **oper** -- **[in]** MCPWM operator, allocated by `mcpwm_new_operator()`

返回

- ESP_OK: Delete MCPWM operator successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM operator failed because of invalid argument
- ESP_FAIL: Delete MCPWM operator failed because of other error

esp_err_t **mcpwm_operator_connect_timer**(*mcpwm_oper_handle_t* oper, *mcpwm_timer_handle_t* timer)

Connect MCPWM operator and timer, so that the operator can be driven by the timer.

参数

- **oper** -- **[in]** MCPWM operator handle, allocated by `mcpwm_new_operator()`
- **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer()`

返回

- ESP_OK: Connect MCPWM operator and timer successfully
- ESP_ERR_INVALID_ARG: Connect MCPWM operator and timer failed because of invalid argument
- ESP_FAIL: Connect MCPWM operator and timer failed because of other error

esp_err_t **mcpwm_operator_set_brake_on_fault**(*mcpwm_oper_handle_t* oper, const *mcpwm_brake_config_t* *config)

Set brake method for MCPWM operator.

参数

- **oper** -- **[in]** MCPWM operator, allocated by `mcpwm_new_operator()`
- **config** -- **[in]** MCPWM brake configuration

返回

- ESP_OK: Set trip for operator successfully
- ESP_ERR_INVALID_ARG: Set trip for operator failed because of invalid argument
- ESP_FAIL: Set trip for operator failed because of other error

esp_err_t **mcpwm_operator_recover_from_fault**(*mcpwm_oper_handle_t* oper, *mcpwm_fault_handle_t* fault)

Try to make the operator recover from fault.

备注： To recover from fault or escape from trip, you make sure the fault signal has disappeared already. Otherwise the recovery can't succeed.

参数

- **oper** -- **[in]** MCPWM operator, allocated by `mcpwm_new_operator()`
- **fault** -- **[in]** MCPWM fault handle

返回

- ESP_OK: Recover from fault successfully
- ESP_ERR_INVALID_ARG: Recover from fault failed because of invalid argument
- ESP_ERR_INVALID_STATE: Recover from fault failed because the fault source is still active
- ESP_FAIL: Recover from fault failed because of other error

esp_err_t **mcpwm_operator_register_event_callbacks**(*mcpwm_oper_handle_t* oper, const *mcpwm_operator_event_callbacks_t* *cbs, void *user_data)

Set event callbacks for MCPWM operator.

备注： User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

参数

- **oper** -- **[in]** MCPWM operator handle, allocated by `mcpwm_new_operator()`
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

esp_err_t **mcpwm_operator_apply_carrier**(*mcpwm_oper_handle_t* oper, const *mcpwm_carrier_config_t* *config)

Apply carrier feature for MCPWM operator.

参数

- **oper** -- **[in]** MCPWM operator, allocated by `mcpwm_new_operator()`
- **config** -- **[in]** MCPWM carrier specific configuration

返回

- ESP_OK: Set carrier for operator successfully
- ESP_ERR_INVALID_ARG: Set carrier for operator failed because of invalid argument
- ESP_FAIL: Set carrier for operator failed because of other error

Structures

struct **mcpwm_operator_config_t**

MCPWM operator configuration.

Public Members

int **group_id**

Specify from which group to allocate the MCPWM operator

int **intr_priority**

MCPWM operator interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **update_gen_action_on_tez**

Whether to update generator action when timer counts to zero

uint32_t **update_gen_action_on_tep**

Whether to update generator action when timer counts to peak

uint32_t **update_gen_action_on_sync**

Whether to update generator action on sync event

uint32_t **update_dead_time_on_tez**

Whether to update dead time when timer counts to zero

uint32_t **update_dead_time_on_tep**

Whether to update dead time when timer counts to peak

uint32_t **update_dead_time_on_sync**

Whether to update dead time on sync event

struct *mcpwm_operator_config_t*::[anonymous] **flags**

Extra configuration flags for operator

struct **mcpwm_brake_config_t**

MCPWM brake configuration structure.

Public Members

mcpwm_fault_handle_t **fault**

Which fault causes the operator to brake

mcpwm_operator_brake_mode_t **brake_mode**

Brake mode

uint32_t **cbc_recover_on_tez**

Recovery CBC brake state on tez event

uint32_t **cbc_recover_on_tep**

Recovery CBC brake state on tep event

struct *mcpwm_brake_config_t*::[anonymous] **flags**

Extra flags for brake configuration

struct **mcpwm_operator_event_callbacks_t**

Group of supported MCPWM operator event callbacks.

备注: The callbacks are all running under ISR environment

Public Members

mcpwm_brake_event_cb_t **on_brake_cbc**

callback function when mcpwm operator brakes in CBC

mcpwm_brake_event_cb_t **on_brake_ost**

callback function when mcpwm operator brakes in OST

struct **mcpwm_carrier_config_t**

MCPWM carrier configuration structure.

Public Members

mcpwm_carrier_clock_source_t **clk_src**

MCPWM carrier clock source

uint32_t **frequency_hz**

Carrier frequency in Hz

uint32_t **first_pulse_duration_us**

The duration of the first PWM pulse, in us

float **duty_cycle**

Carrier duty cycle

uint32_t **invert_before_modulate**

Invert the raw signal

uint32_t **invert_after_modulate**

Invert the modulated signal

struct *mcpwm_carrier_config_t*::[anonymous] **flags**

Extra flags for carrier configuration

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_cmpr.h
- This header file can be included with:

```
#include "driver/mcpwm_cmpr.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Functions

esp_err_t **mcpwm_new_comparator**(*mcpwm_oper_handle_t* oper, const *mcpwm_comparator_config_t* *config, *mcpwm_cmpr_handle_t* *ret_cmpr)

Create MCPWM comparator.

参数

- **oper** -- **[in]** MCPWM operator, allocated by `mcpwm_new_operator()`, the new comparator will be allocated from this operator
- **config** -- **[in]** MCPWM comparator configuration
- **ret_cmpr** -- **[out]** Returned MCPWM comparator

返回

- ESP_OK: Create MCPWM comparator successfully
- ESP_ERR_INVALID_ARG: Create MCPWM comparator failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM comparator failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM comparator failed because can't find free resource
- ESP_FAIL: Create MCPWM comparator failed because of other error

esp_err_t **mcpwm_del_comparator**(*mcpwm_cmpr_handle_t* cmpr)

Delete MCPWM comparator.

参数 **cmpr** -- **[in]** MCPWM comparator handle, allocated by `mcpwm_new_comparator()`

返回

- ESP_OK: Delete MCPWM comparator successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM comparator failed because of invalid argument
- ESP_FAIL: Delete MCPWM comparator failed because of other error

esp_err_t **mcpwm_new_event_comparator**(*mcpwm_oper_handle_t* oper, const *mcpwm_event_comparator_config_t* *config, *mcpwm_cmpr_handle_t* *ret_cmpr)

Create MCPWM event comparator.

参数

- **oper** -- **[in]** MCPWM operator, allocated by `mcpwm_new_operator()`, the new event comparator will be allocated from this operator
- **config** -- **[in]** MCPWM comparator configuration
- **ret_cmpr** -- **[out]** Returned MCPWM event comparator

返回

- ESP_OK: Create MCPWM event comparator successfully
- ESP_ERR_INVALID_ARG: Create MCPWM event comparator failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM event comparator failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM event comparator failed because can't find free resource
- ESP_FAIL: Create MCPWM event comparator failed because of other error

esp_err_t **mcpwm_comparator_register_event_callbacks**(*mcpwm_cmpr_handle_t* cmpr, const *mcpwm_comparator_event_callbacks_t* *cbs, void *user_data)

Set event callbacks for MCPWM comparator.

备注： User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

参数

- **`cmpr`** -- **[in]** MCPWM comparator handle, allocated by `mcpwm_new_comparator()`
- **`cbs`** -- **[in]** Group of callback functions
- **`user_data`** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

esp_err_t **`mcpwm_comparator_set_compare_value`**(*mcpwm_cmpr_handle_t* cmpr, uint32_t cmp_ticks)

Set MCPWM comparator's compare value.

参数

- **`cmpr`** -- **[in]** MCPWM comparator handle, allocated by `mcpwm_new_comparator()`
- **`cmp_ticks`** -- **[in]** The new compare value

返回

- ESP_OK: Set MCPWM compare value successfully
- ESP_ERR_INVALID_ARG: Set MCPWM compare value failed because of invalid argument (e.g. the cmp_ticks is out of range)
- ESP_ERR_INVALID_STATE: Set MCPWM compare value failed because the operator doesn't have a timer connected
- ESP_FAIL: Set MCPWM compare value failed because of other error

Structures

struct **`mcpwm_comparator_config_t`**

MCPWM comparator configuration.

Public Members

int **`intr_priority`**

MCPWM comparator interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **`update_cmp_on_tez`**

Whether to update compare value when timer count equals to zero (tez)

uint32_t **`update_cmp_on_tep`**

Whether to update compare value when timer count equals to peak (tep)

uint32_t **`update_cmp_on_sync`**

Whether to update compare value on sync event

struct *mcpwm_comparator_config_t*::[anonymous] **`flags`**

Extra configuration flags for comparator

struct **mcpwm_event_comparator_config_t**

MCPWM event comparator configuration.

struct **mcpwm_comparator_event_callbacks_t**

Group of supported MCPWM compare event callbacks.

备注： The callbacks are all running under ISR environment

Public Members

mcpwm_compare_event_cb_t **on_reach**

ISR callback function which would be invoked when counter reaches compare value

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_gen.h
- This header file can be included with:

```
#include "driver/mcpwm_gen.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Functions

esp_err_t **mcpwm_new_generator**(*mcpwm_oper_handle_t* oper, const *mcpwm_generator_config_t* *config, *mcpwm_gen_handle_t* *ret_gen)

Allocate MCPWM generator from given operator.

参数

- **oper** -- **[in]** MCPWM operator, allocated by `mcpwm_new_operator()`
- **config** -- **[in]** MCPWM generator configuration
- **ret_gen** -- **[out]** Returned MCPWM generator

返回

- ESP_OK: Create MCPWM generator successfully
- ESP_ERR_INVALID_ARG: Create MCPWM generator failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM generator failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM generator failed because can't find free resource
- ESP_FAIL: Create MCPWM generator failed because of other error

esp_err_t **mcpwm_del_generator**(*mcpwm_gen_handle_t* gen)

Delete MCPWM generator.

参数 **gen** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`

返回

- ESP_OK: Delete MCPWM generator successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM generator failed because of invalid argument

- ESP_FAIL: Delete MCPWM generator failed because of other error

esp_err_t **mcpwm_generator_set_force_level**(*mcpwm_gen_handle_t* gen, int level, bool hold_on)

Set force level for MCPWM generator.

备注： The force level will be applied to the generator immediately, regardless any other events that would change the generator's behaviour.

备注： If the `hold_on` is true, the force level will retain forever, until user removes the force level by setting the force level to `-1`.

备注： If the `hold_on` is false, the force level can be overridden by the next event action.

备注： The force level set by this function can be inverted by GPIO matrix or dead-time module. So the level set here doesn't equal to the final output level.

参数

- **gen** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **level** -- **[in]** GPIO level to be applied to MCPWM generator, specially, -1 means to remove the force level
- **hold_on** -- **[in]** Whether the forced PWM level should retain (i.e. will remain unchanged until manually remove the force level)

返回

- ESP_OK: Set force level for MCPWM generator successfully
- ESP_ERR_INVALID_ARG: Set force level for MCPWM generator failed because of invalid argument
- ESP_FAIL: Set force level for MCPWM generator failed because of other error

esp_err_t **mcpwm_generator_set_action_on_timer_event**(*mcpwm_gen_handle_t* gen, *mcpwm_gen_timer_event_action_t* ev_act)

Set generator action on MCPWM timer event.

参数

- **gen** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **ev_act** -- **[in]** MCPWM timer event action, can be constructed by `MCPWM_GEN_TIMER_EVENT_ACTION` helper macro

返回

- ESP_OK: Set generator action successfully
- ESP_ERR_INVALID_ARG: Set generator action failed because of invalid argument
- ESP_ERR_INVALID_STATE: Set generator action failed because of timer is not connected to operator
- ESP_FAIL: Set generator action failed because of other error

esp_err_t **mcpwm_generator_set_actions_on_timer_event**(*mcpwm_gen_handle_t* gen, *mcpwm_gen_timer_event_action_t* ev_act, ...)

Set generator actions on multiple MCPWM timer events.

备注： This is an aggregation version of `mcpwm_generator_set_action_on_timer_event`, which allows user to set multiple actions in one call.

参数

- **gen** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **ev_act** -- **[in]** MCPWM timer event action list, must be terminated by `MCPWM_GEN_TIMER_EVENT_ACTION_END()`

返回

- ESP_OK: Set generator actions successfully
- ESP_ERR_INVALID_ARG: Set generator actions failed because of invalid argument
- ESP_ERR_INVALID_STATE: Set generator actions failed because of timer is not connected to operator
- ESP_FAIL: Set generator actions failed because of other error

esp_err_t **mcpwm_generator_set_action_on_compare_event**(*mcpwm_gen_handle_t* generator, *mcpwm_gen_compare_event_action_t* ev_act)

Set generator action on MCPWM compare event.

参数

- **generator** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **ev_act** -- **[in]** MCPWM compare event action, can be constructed by `MCPWM_GEN_COMPARE_EVENT_ACTION` helper macro

返回

- ESP_OK: Set generator action successfully
- ESP_ERR_INVALID_ARG: Set generator action failed because of invalid argument
- ESP_FAIL: Set generator action failed because of other error

esp_err_t **mcpwm_generator_set_actions_on_compare_event**(*mcpwm_gen_handle_t* generator, *mcpwm_gen_compare_event_action_t* ev_act, ...)

Set generator actions on multiple MCPWM compare events.

备注: This is an aggregation version of `mcpwm_generator_set_action_on_compare_event`, which allows user to set multiple actions in one call.

参数

- **generator** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **ev_act** -- **[in]** MCPWM compare event action list, must be terminated by `MCPWM_GEN_COMPARE_EVENT_ACTION_END()`

返回

- ESP_OK: Set generator actions successfully
- ESP_ERR_INVALID_ARG: Set generator actions failed because of invalid argument
- ESP_FAIL: Set generator actions failed because of other error

esp_err_t **mcpwm_generator_set_action_on_brake_event**(*mcpwm_gen_handle_t* generator, *mcpwm_gen_brake_event_action_t* ev_act)

Set generator action on MCPWM brake event.

参数

- **generator** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **ev_act** -- **[in]** MCPWM brake event action, can be constructed by `MCPWM_GEN_BRAKE_EVENT_ACTION` helper macro

返回

- ESP_OK: Set generator action successfully
- ESP_ERR_INVALID_ARG: Set generator action failed because of invalid argument

- ESP_FAIL: Set generator action failed because of other error

esp_err_t **mcpwm_generator_set_actions_on_brake_event**(*mcpwm_gen_handle_t* generator, *mcpwm_gen_brake_event_action_t* ev_act, ...)

Set generator actions on multiple MCPWM brake events.

备注： This is an aggregation version of `mcpwm_generator_set_action_on_brake_event`, which allows user to set multiple actions in one call.

参数

- **generator** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **ev_act** -- **[in]** MCPWM brake event action list, must be terminated by `MCPWM_GEN_BRAKE_EVENT_ACTION_END()`

返回

- ESP_OK: Set generator actions successfully
- ESP_ERR_INVALID_ARG: Set generator actions failed because of invalid argument
- ESP_FAIL: Set generator actions failed because of other error

esp_err_t **mcpwm_generator_set_action_on_fault_event**(*mcpwm_gen_handle_t* generator, *mcpwm_gen_fault_event_action_t* ev_act)

Set generator action on MCPWM Fault event.

参数

- **generator** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **ev_act** -- **[in]** MCPWM trigger event action, can be constructed by `MCPWM_GEN_FAULT_EVENT_ACTION` helper macro

返回

- ESP_OK: Set generator action successfully
- ESP_ERR_INVALID_ARG: Set generator action failed because of invalid argument
- ESP_FAIL: Set generator action failed because of other error

esp_err_t **mcpwm_generator_set_action_on_sync_event**(*mcpwm_gen_handle_t* generator, *mcpwm_gen_sync_event_action_t* ev_act)

Set generator action on MCPWM Sync event.

备注： The trigger only support one sync action, regardless of the kinds. Should not call this function more than once.

参数

- **generator** -- **[in]** MCPWM generator handle, allocated by `mcpwm_new_generator()`
- **ev_act** -- **[in]** MCPWM trigger event action, can be constructed by `MCPWM_GEN_SYNC_EVENT_ACTION` helper macro

返回

- ESP_OK: Set generator action successfully
- ESP_ERR_INVALID_ARG: Set generator action failed because of invalid argument
- ESP_FAIL: Set generator action failed because of other error

esp_err_t **mcpwm_generator_set_dead_time**(*mcpwm_gen_handle_t* in_generator, *mcpwm_gen_handle_t* out_generator, const *mcpwm_dead_time_config_t* *config)

Set dead time for MCPWM generator.

备注： Due to a hardware limitation, you can't set rising edge delay for both MCPWM generator 0 and 1 at the same time, otherwise, there will be a conflict inside the dead time module. The same goes for the falling edge setting. But you can set both the rising edge and falling edge delay for the same MCPWM generator.

参数

- **in_generator** -- **[in]** MCPWM generator, before adding the dead time
- **out_generator** -- **[in]** MCPWM generator, after adding the dead time
- **config** -- **[in]** MCPWM dead time configuration

返回

- ESP_OK: Set dead time for MCPWM generator successfully
- ESP_ERR_INVALID_ARG: Set dead time for MCPWM generator failed because of invalid argument
- ESP_ERR_INVALID_STATE: Set dead time for MCPWM generator failed because of invalid state (e.g. delay module is already in use by other generator)
- ESP_FAIL: Set dead time for MCPWM generator failed because of other error

Structures

struct **mcpwm_generator_config_t**

MCPWM generator configuration.

Public Members

int **gen_gpio_num**

The GPIO number used to output the PWM signal

uint32_t **invert_pwm**

Whether to invert the PWM signal (done by GPIO matrix)

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

uint32_t **io_od_mode**

Configure the GPIO as open-drain mode

uint32_t **pull_up**

Whether to pull up internally

uint32_t **pull_down**

Whether to pull down internally

struct *mcpwm_generator_config_t*::[anonymous] **flags**

Extra configuration flags for generator

struct **mcpwm_gen_timer_event_action_t**

Generator action on specific timer event.

Public Members

mcpwm_timer_direction_t **direction**

Timer direction

mcpwm_timer_event_t **event**

Timer event

mcpwm_generator_action_t **action**

Generator action should perform

struct **mcpwm_gen_compare_event_action_t**

Generator action on specific comparator event.

Public Members

mcpwm_timer_direction_t **direction**

Timer direction

mcpwm_cmpr_handle_t **comparator**

Comparator handle

mcpwm_generator_action_t **action**

Generator action should perform

struct **mcpwm_gen_brake_event_action_t**

Generator action on specific brake event.

Public Members

mcpwm_timer_direction_t **direction**

Timer direction

mcpwm_operator_brake_mode_t **brake_mode**

Brake mode

mcpwm_generator_action_t **action**

Generator action should perform

struct **mcpwm_gen_fault_event_action_t**

Generator action on specific fault event.

Public Members

mcpwm_timer_direction_t **direction**

Timer direction

mcpwm_fault_handle_t **fault**

Which fault as the trigger. Only support GPIO fault

mcpwm_generator_action_t **action**

Generator action should perform

struct **mcpwm_gen_sync_event_action_t**

Generator action on specific sync event.

Public Members

mcpwm_timer_direction_t **direction**

Timer direction

mcpwm_sync_handle_t **sync**

Which sync as the trigger

mcpwm_generator_action_t **action**

Generator action should perform

struct **mcpwm_dead_time_config_t**

MCPWM dead time configuration structure.

Public Members

uint32_t **posedge_delay_ticks**

delay time applied to rising edge, 0 means no rising delay time

uint32_t **negedge_delay_ticks**

delay time applied to falling edge, 0 means no falling delay time

uint32_t **invert_output**

Invert the signal after applied the dead time

struct *mcpwm_dead_time_config_t*::[anonymous] **flags**

Extra flags for dead time configuration

Macros

MCPWM_GEN_TIMER_EVENT_ACTION(dir, ev, act)

Help macros to construct a *mcpwm_gen_timer_event_action_t* entry.

MCPWM_GEN_TIMER_EVENT_ACTION_END()

MCPWM_GEN_COMPARE_EVENT_ACTION(dir, cmp, act)

Help macros to construct a *mcpwm_gen_compare_event_action_t* entry.

MCPWM_GEN_COMPARE_EVENT_ACTION_END()

MCPWM_GEN_BRAKE_EVENT_ACTION(dir, mode, act)

Help macros to construct a *mcpwm_gen_brake_event_action_t* entry.

MCPWM_GEN_BRAKE_EVENT_ACTION_END()

MCPWM_GEN_FAULT_EVENT_ACTION(dir, flt, act)

Help macros to construct a *mcpwm_gen_fault_event_action_t* entry.

MCPWM_GEN_SYNC_EVENT_ACTION(dir, syn, act)

Help macros to construct a *mcpwm_gen_sync_event_action_t* entry.

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_fault.h
- This header file can be included with:

```
#include "driver/mcpwm_fault.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Functions

esp_err_t **mcpwm_new_gpio_fault**(const *mcpwm_gpio_fault_config_t* *config, *mcpwm_fault_handle_t* *ret_fault)

Create MCPWM GPIO fault.

参数

- **config** -- **[in]** MCPWM GPIO fault configuration
- **ret_fault** -- **[out]** Returned GPIO fault handle

返回

- ESP_OK: Create MCPWM GPIO fault successfully
- ESP_ERR_INVALID_ARG: Create MCPWM GPIO fault failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM GPIO fault failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM GPIO fault failed because can't find free resource
- ESP_FAIL: Create MCPWM GPIO fault failed because of other error

esp_err_t **mcpwm_new_soft_fault**(const *mcpwm_soft_fault_config_t* *config, *mcpwm_fault_handle_t* *ret_fault)

Create MCPWM software fault.

参数

- **config** -- **[in]** MCPWM software fault configuration
- **ret_fault** -- **[out]** Returned software fault handle

返回

- ESP_OK: Create MCPWM software fault successfully
- ESP_ERR_INVALID_ARG: Create MCPWM software fault failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM software fault failed because out of memory
- ESP_FAIL: Create MCPWM software fault failed because of other error

esp_err_t **mcpwm_del_fault**(*mcpwm_fault_handle_t* fault)

Delete MCPWM fault.

参数 **fault** -- **[in]** MCPWM fault handle allocated by `mcpwm_new_gpio_fault()` or `mcpwm_new_soft_fault()`

返回

- ESP_OK: Delete MCPWM fault successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM fault failed because of invalid argument
- ESP_FAIL: Delete MCPWM fault failed because of other error

esp_err_t **mcpwm_soft_fault_activate**(*mcpwm_fault_handle_t* fault)

Activate the software fault, trigger the fault event for once.

参数 **fault** -- **[in]** MCPWM soft fault, allocated by `mcpwm_new_soft_fault()`

返回

- ESP_OK: Trigger MCPWM software fault event successfully
- ESP_ERR_INVALID_ARG: Trigger MCPWM software fault event failed because of invalid argument
- ESP_FAIL: Trigger MCPWM software fault event failed because of other error

esp_err_t **mcpwm_fault_register_event_callbacks**(*mcpwm_fault_handle_t* fault, const *mcpwm_fault_event_callbacks_t* *cbs, void *user_data)

Set event callbacks for MCPWM fault.

备注: User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

参数

- **fault** -- **[in]** MCPWM GPIO fault handle, allocated by `mcpwm_new_gpio_fault()`
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

Structures

struct **mcpwm_gpio_fault_config_t**

MCPWM GPIO fault configuration structure.

Public Members

int **group_id**

In which MCPWM group that the GPIO fault belongs to

int **intr_priority**

MCPWM GPIO fault interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

int **gpio_num**

GPIO used by the fault signal

uint32_t **active_level**

On which level the fault signal is treated as active

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

uint32_t **pull_up**

Whether to pull up internally

uint32_t **pull_down**

Whether to pull down internally

struct *mcpwm_gpio_fault_config_t*::[anonymous] **flags**

Extra configuration flags for GPIO fault

struct **mcpwm_soft_fault_config_t**

MCPWM software fault configuration structure.

struct **mcpwm_fault_event_callbacks_t**

Group of supported MCPWM fault event callbacks.

备注： The callbacks are all running under ISR environment

Public Members

mcpwm_fault_event_cb_t **on_fault_enter**

ISR callback function that would be invoked when fault signal becomes active

mcpwm_fault_event_cb_t **on_fault_exit**

ISR callback function that would be invoked when fault signal becomes inactive

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_sync.h
- This header file can be included with:

```
#include "driver/mcpwm_sync.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Functions

esp_err_t **mcpwm_new_timer_sync_src**(*mcpwm_timer_handle_t* timer, const *mcpwm_timer_sync_src_config_t* *config, *mcpwm_sync_handle_t* *ret_sync)

Create MCPWM timer sync source.

参数

- **timer** -- **[in]** MCPWM timer handle, allocated by `mcpwm_new_timer()`

- **config** -- **[in]** MCPWM timer sync source configuration
- **ret_sync** -- **[out]** Returned MCPWM sync handle

返回

- ESP_OK: Create MCPWM timer sync source successfully
- ESP_ERR_INVALID_ARG: Create MCPWM timer sync source failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM timer sync source failed because out of memory
- ESP_ERR_INVALID_STATE: Create MCPWM timer sync source failed because the timer has created a sync source before
- ESP_FAIL: Create MCPWM timer sync source failed because of other error

esp_err_t **mcpwm_new_gpio_sync_src**(const *mcpwm_gpio_sync_src_config_t* *config, *mcpwm_sync_handle_t* *ret_sync)

Create MCPWM GPIO sync source.

参数

- **config** -- **[in]** MCPWM GPIO sync source configuration
- **ret_sync** -- **[out]** Returned MCPWM GPIO sync handle

返回

- ESP_OK: Create MCPWM GPIO sync source successfully
- ESP_ERR_INVALID_ARG: Create MCPWM GPIO sync source failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM GPIO sync source failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM GPIO sync source failed because can't find free resource
- ESP_FAIL: Create MCPWM GPIO sync source failed because of other error

esp_err_t **mcpwm_new_soft_sync_src**(const *mcpwm_soft_sync_config_t* *config, *mcpwm_sync_handle_t* *ret_sync)

Create MCPWM software sync source.

参数

- **config** -- **[in]** MCPWM software sync source configuration
- **ret_sync** -- **[out]** Returned software sync handle

返回

- ESP_OK: Create MCPWM software sync successfully
- ESP_ERR_INVALID_ARG: Create MCPWM software sync failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM software sync failed because out of memory
- ESP_FAIL: Create MCPWM software sync failed because of other error

esp_err_t **mcpwm_del_sync_src**(*mcpwm_sync_handle_t* sync)

Delete MCPWM sync source.

参数 **sync** -- **[in]** MCPWM sync handle, allocated by `mcpwm_new_timer_sync_src()` or `mcpwm_new_gpio_sync_src()` or `mcpwm_new_soft_sync_src()`

返回

- ESP_OK: Delete MCPWM sync source successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM sync source failed because of invalid argument
- ESP_FAIL: Delete MCPWM sync source failed because of other error

esp_err_t **mcpwm_soft_sync_activate**(*mcpwm_sync_handle_t* sync)

Activate the software sync, trigger the sync event for once.

参数 **sync** -- **[in]** MCPWM soft sync handle, allocated by `mcpwm_new_soft_sync_src()`

返回

- ESP_OK: Trigger MCPWM software sync event successfully
- ESP_ERR_INVALID_ARG: Trigger MCPWM software sync event failed because of invalid argument
- ESP_FAIL: Trigger MCPWM software sync event failed because of other error

Structures

struct **mcpwm_timer_sync_src_config_t**

MCPWM timer sync source configuration.

Public Members

mcpwm_timer_event_t **timer_event**

Timer event, upon which MCPWM timer will generate the sync signal

uint32_t **propagate_input_sync**

The input sync signal would be routed to its sync output

struct *mcpwm_timer_sync_src_config_t*::[anonymous] **flags**

Extra configuration flags for timer sync source

struct **mcpwm_gpio_sync_src_config_t**

MCPWM GPIO sync source configuration.

Public Members

int **group_id**

MCPWM group ID

int **gpio_num**

GPIO used by sync source

uint32_t **active_neg**

Whether the sync signal is active on negedge, by default, the sync signal's posedge is treated as active

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

uint32_t **pull_up**

Whether to pull up internally

uint32_t **pull_down**

Whether to pull down internally

struct *mcpwm_gpio_sync_src_config_t*::[anonymous] **flags**

Extra configuration flags for GPIO sync source

struct **mcpwm_soft_sync_config_t**

MCPWM software sync configuration structure.

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_cap.h
- This header file can be included with:

```
#include "driver/mcpwm_cap.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Functions

esp_err_t **mcpwm_new_capture_timer**(const *mcpwm_capture_timer_config_t* *config, *mcpwm_cap_timer_handle_t* *ret_cap_timer)

Create MCPWM capture timer.

参数

- **config** -- **[in]** MCPWM capture timer configuration
- **ret_cap_timer** -- **[out]** Returned MCPWM capture timer handle

返回

- ESP_OK: Create MCPWM capture timer successfully
- ESP_ERR_INVALID_ARG: Create MCPWM capture timer failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM capture timer failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM capture timer failed because can't find free resource
- ESP_FAIL: Create MCPWM capture timer failed because of other error

esp_err_t **mcpwm_del_capture_timer**(*mcpwm_cap_timer_handle_t* cap_timer)

Delete MCPWM capture timer.

参数 **cap_timer** -- **[in]** MCPWM capture timer, allocated by `mcpwm_new_capture_timer()`

返回

- ESP_OK: Delete MCPWM capture timer successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM capture timer failed because of invalid argument
- ESP_FAIL: Delete MCPWM capture timer failed because of other error

esp_err_t **mcpwm_capture_timer_enable**(*mcpwm_cap_timer_handle_t* cap_timer)

Enable MCPWM capture timer.

参数 **cap_timer** -- **[in]** MCPWM capture timer handle, allocated by `mcpwm_new_capture_timer()`

返回

- ESP_OK: Enable MCPWM capture timer successfully
- ESP_ERR_INVALID_ARG: Enable MCPWM capture timer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable MCPWM capture timer failed because timer is enabled already
- ESP_FAIL: Enable MCPWM capture timer failed because of other error

esp_err_t **mcpwm_capture_timer_disable**(*mcpwm_cap_timer_handle_t* cap_timer)

Disable MCPWM capture timer.

参数 **cap_timer** -- **[in]** MCPWM capture timer handle, allocated by `mcpwm_new_capture_timer()`

返回

- ESP_OK: Disable MCPWM capture timer successfully
- ESP_ERR_INVALID_ARG: Disable MCPWM capture timer failed because of invalid argument

- ESP_ERR_INVALID_STATE: Disable MCPWM capture timer failed because timer is disabled already
- ESP_FAIL: Disable MCPWM capture timer failed because of other error

esp_err_t **mcpwm_capture_timer_start**(*mcpwm_cap_timer_handle_t* cap_timer)

Start MCPWM capture timer.

参数 **cap_timer** -- **[in]** MCPWM capture timer, allocated by `mcpwm_new_capture_timer()`

返回

- ESP_OK: Start MCPWM capture timer successfully
- ESP_ERR_INVALID_ARG: Start MCPWM capture timer failed because of invalid argument
- ESP_FAIL: Start MCPWM capture timer failed because of other error

esp_err_t **mcpwm_capture_timer_stop**(*mcpwm_cap_timer_handle_t* cap_timer)

Stop MCPWM capture timer.

参数 **cap_timer** -- **[in]** MCPWM capture timer, allocated by `mcpwm_new_capture_timer()`

返回

- ESP_OK: Stop MCPWM capture timer successfully
- ESP_ERR_INVALID_ARG: Stop MCPWM capture timer failed because of invalid argument
- ESP_FAIL: Stop MCPWM capture timer failed because of other error

esp_err_t **mcpwm_capture_timer_get_resolution**(*mcpwm_cap_timer_handle_t* cap_timer, uint32_t *out_resolution)

Get MCPWM capture timer resolution, in Hz.

参数

- **cap_timer** -- **[in]** MCPWM capture timer, allocated by `mcpwm_new_capture_timer()`
- **out_resolution** -- **[out]** Returned capture timer resolution, in Hz

返回

- ESP_OK: Get capture timer resolution successfully
- ESP_ERR_INVALID_ARG: Get capture timer resolution failed because of invalid argument
- ESP_FAIL: Get capture timer resolution failed because of other error

esp_err_t **mcpwm_capture_timer_set_phase_on_sync**(*mcpwm_cap_timer_handle_t* cap_timer, const *mcpwm_capture_timer_sync_phase_config_t* *config)

Set sync phase for MCPWM capture timer.

参数

- **cap_timer** -- **[in]** MCPWM capture timer, allocated by `mcpwm_new_capture_timer()`
- **config** -- **[in]** MCPWM capture timer sync phase configuration

返回

- ESP_OK: Set sync phase for MCPWM capture timer successfully
- ESP_ERR_INVALID_ARG: Set sync phase for MCPWM capture timer failed because of invalid argument
- ESP_FAIL: Set sync phase for MCPWM capture timer failed because of other error

esp_err_t **mcpwm_new_capture_channel**(*mcpwm_cap_timer_handle_t* cap_timer, const *mcpwm_capture_channel_config_t* *config, *mcpwm_cap_channel_handle_t* *ret_cap_channel)

Create MCPWM capture channel.

备注： The created capture channel won't be enabled until calling `mcpwm_capture_channel_enable`

参数

- **cap_timer** -- **[in]** MCPWM capture timer, allocated by `mcpwm_new_capture_timer()`, will be connected to the new capture channel
- **config** -- **[in]** MCPWM capture channel configuration
- **ret_cap_channel** -- **[out]** Returned MCPWM capture channel

返回

- ESP_OK: Create MCPWM capture channel successfully
- ESP_ERR_INVALID_ARG: Create MCPWM capture channel failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM capture channel failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM capture channel failed because can't find free resource
- ESP_FAIL: Create MCPWM capture channel failed because of other error

esp_err_t **mcpwm_del_capture_channel**(*mcpwm_cap_channel_handle_t* cap_channel)

Delete MCPWM capture channel.

参数 **cap_channel** -- **[in]** MCPWM capture channel handle, allocated by `mcpwm_new_capture_channel()`

返回

- ESP_OK: Delete MCPWM capture channel successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM capture channel failed because of invalid argument
- ESP_FAIL: Delete MCPWM capture channel failed because of other error

esp_err_t **mcpwm_capture_channel_enable**(*mcpwm_cap_channel_handle_t* cap_channel)

Enable MCPWM capture channel.

备注： This function will transit the channel state from init to enable.

备 注： This function will enable the interrupt service, if it's lazy installed in `mcpwm_capture_channel_register_event_callbacks()`.

参数 **cap_channel** -- **[in]** MCPWM capture channel handle, allocated by `mcpwm_new_capture_channel()`

返回

- ESP_OK: Enable MCPWM capture channel successfully
- ESP_ERR_INVALID_ARG: Enable MCPWM capture channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable MCPWM capture channel failed because the channel is already enabled
- ESP_FAIL: Enable MCPWM capture channel failed because of other error

esp_err_t **mcpwm_capture_channel_disable**(*mcpwm_cap_channel_handle_t* cap_channel)

Disable MCPWM capture channel.

参数 **cap_channel** -- **[in]** MCPWM capture channel handle, allocated by `mcpwm_new_capture_channel()`

返回

- ESP_OK: Disable MCPWM capture channel successfully

- ESP_ERR_INVALID_ARG: Disable MCPWM capture channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable MCPWM capture channel failed because the channel is not enabled yet
- ESP_FAIL: Disable MCPWM capture channel failed because of other error

esp_err_t **mcpwm_capture_channel_register_event_callbacks**(*mcpwm_cap_channel_handle_t* cap_channel, const *mcpwm_capture_event_callbacks_t* *cbs, void *user_data)

Set event callbacks for MCPWM capture channel.

备注: The first call to this function needs to be before the call to `mcpwm_capture_channel_enable`

备注: User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

参数

- **cap_channel** -- **[in]** MCPWM capture channel handle, allocated by `mcpwm_new_capture_channel()`
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_ERR_INVALID_STATE: Set event callbacks failed because the channel is not in init state
- ESP_FAIL: Set event callbacks failed because of other error

esp_err_t **mcpwm_capture_channel_trigger_soft_catch**(*mcpwm_cap_channel_handle_t* cap_channel)

Trigger a catch by software.

参数 **cap_channel** -- **[in]** MCPWM capture channel handle, allocated by `mcpwm_new_capture_channel()`

返回

- ESP_OK: Trigger software catch successfully
- ESP_ERR_INVALID_ARG: Trigger software catch failed because of invalid argument
- ESP_ERR_INVALID_STATE: Trigger software catch failed because the channel is not enabled yet
- ESP_FAIL: Trigger software catch failed because of other error

Structures

struct **mcpwm_capture_timer_config_t**

MCPWM capture timer configuration structure.

Public Members

int **group_id**

Specify from which group to allocate the capture timer

mcpwm_capture_clock_source_t **clk_src**

MCPWM capture timer clock source

uint32_t **resolution_hz**

Resolution of capture timer

struct **mcpwm_capture_timer_sync_phase_config_t**

MCPWM Capture timer sync phase configuration.

Public Members

mcpwm_sync_handle_t **sync_src**

The sync event source

uint32_t **count_value**

The count value that should lock to upon sync event

mcpwm_timer_direction_t **direction**

The count direction that should lock to upon sync event

struct **mcpwm_capture_channel_config_t**

MCPWM capture channel configuration structure.

Public Members

int **gpio_num**

GPIO used capturing input signal

int **intr_priority**

MCPWM capture interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **prescale**

Prescale of input signal, effective frequency = cap_input_clk/prescale

struct *mcpwm_capture_channel_config_t*::*extra_capture_channel_flags* **flags**

Extra configuration flags for capture channel

struct **extra_capture_channel_flags**

Extra configuration flags for capture channel.

Public Members

uint32_t **pos_edge**

Whether to capture on positive edge

uint32_t **neg_edge**

Whether to capture on negative edge

uint32_t **pull_up**

Whether to pull up internally

uint32_t **pull_down**

Whether to pull down internally

uint32_t **invert_cap_signal**

Invert the input capture signal

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

uint32_t **keep_io_conf_at_exit**

For debug/test, whether to keep the GPIO configuration when capture channel is deleted. By default, driver will reset the GPIO pin at exit.

struct **mcpwm_capture_event_callbacks_t**

Group of supported MCPWM capture event callbacks.

备注： The callbacks are all running under ISR environment

Public Members

mcpwm_capture_event_cb_t **on_cap**

Callback function that would be invoked when capture event occurred

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_etm.h
- This header file can be included with:

```
#include "driver/mcpwm_etm.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Functions

esp_err_t **mcpwm_comparator_new_etm_event**(*mcpwm_cmpr_handle_t* cmpr, const *mcpwm_cmpr_etm_event_config_t* *config, *esp_etm_event_handle_t* *out_event)

Get the ETM event for MCPWM comparator.

备注： The created ETM event object can be deleted later by calling `esp_etm_del_event`

参数

- **`cmpr`** -- **[in]** MCPWM comparator, allocated by `mcpwm_new_comparator()` or `mcpwm_new_event_comparator()`
- **`config`** -- **[in]** MCPWM ETM comparator event configuration
- **`out_event`** -- **[out]** Returned ETM event handle

返回

- ESP_OK: Get ETM event successfully
- ESP_ERR_INVALID_ARG: Get ETM event failed because of invalid argument
- ESP_FAIL: Get ETM event failed because of other error

Structures

struct **`mcpwm_cmpr_etm_event_config_t`**

MCPWM event comparator ETM event configuration.

Public Members

mcpwm_comparator_etm_event_type_t **`event_type`**

MCPWM comparator ETM event type

Header File

- components/esp_driver_mcpwm/include/driver/mcpwm_types.h
- This header file can be included with:

```
#include "driver/mcpwm_types.h"
```

- This header file is a part of the API provided by the `esp_driver_mcpwm` component. To declare that your component depends on `esp_driver_mcpwm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_mcpwm
```

or

```
PRIV_REQUIRES esp_driver_mcpwm
```

Structures

struct **`mcpwm_timer_event_data_t`**

MCPWM timer event data.

Public Members

uint32_t **`count_value`**

MCPWM timer count value

mcpwm_timer_direction_t **`direction`**

MCPWM timer count direction

struct **mcpwm_brake_event_data_t**

MCPWM brake event data.

struct **mcpwm_fault_event_data_t**

MCPWM fault event data.

struct **mcpwm_compare_event_data_t**

MCPWM compare event data.

Public Members

uint32_t **compare_ticks**

Compare value

mcpwm_timer_direction_t **direction**

Count direction

struct **mcpwm_capture_event_data_t**

MCPWM capture event data.

Public Members

uint32_t **cap_value**

Captured value

mcpwm_capture_edge_t **cap_edge**

Capture edge

Type Definitions

typedef struct mcpwm_timer_t ***mcpwm_timer_handle_t**

Type of MCPWM timer handle.

typedef struct mcpwm_oper_t ***mcpwm_oper_handle_t**

Type of MCPWM operator handle.

typedef struct mcpwm_cmpr_t ***mcpwm_cmpr_handle_t**

Type of MCPWM comparator handle.

typedef struct mcpwm_gen_t ***mcpwm_gen_handle_t**

Type of MCPWM generator handle.

typedef struct mcpwm_fault_t ***mcpwm_fault_handle_t**

Type of MCPWM fault handle.

typedef struct mcpwm_sync_t ***mcpwm_sync_handle_t**

Type of MCPWM sync handle.

typedef struct mcpwm_cap_timer_t ***mcpwm_cap_timer_handle_t**

Type of MCPWM capture timer handle.

typedef struct mcpwm_cap_channel_t ***mcpwm_cap_channel_handle_t**

Type of MCPWM capture channel handle.

typedef bool (***mcpwm_timer_event_cb_t**)(*mcpwm_timer_handle_t* timer, const *mcpwm_timer_event_data_t* *edata, void *user_ctx)

MCPWM timer event callback function.

Param timer **[in]** MCPWM timer handle
Param edata **[in]** MCPWM timer event data, fed by driver
Param user_ctx **[in]** User data, set in `mcpwm_timer_register_event_callbacks()`
Return Whether a high priority task has been waken up by this function

typedef bool (***mcpwm_brake_event_cb_t**)(*mcpwm_oper_handle_t* oper, const *mcpwm_brake_event_data_t* *edata, void *user_ctx)

MCPWM operator brake event callback function.

Param oper **[in]** MCPWM operator handle
Param edata **[in]** MCPWM brake event data, fed by driver
Param user_ctx **[in]** User data, set in `mcpwm_operator_register_event_callbacks()`
Return Whether a high priority task has been waken up by this function

typedef bool (***mcpwm_fault_event_cb_t**)(*mcpwm_fault_handle_t* fault, const *mcpwm_fault_event_data_t* *edata, void *user_ctx)

MCPWM fault event callback function.

Param fault MCPWM fault handle
Param edata MCPWM fault event data, fed by driver
Param user_ctx User data, set in `mcpwm_fault_register_event_callbacks()`
Return whether a task switch is needed after the callback returns

typedef bool (***mcpwm_compare_event_cb_t**)(*mcpwm_cmpr_handle_t* comparator, const *mcpwm_compare_event_data_t* *edata, void *user_ctx)

MCPWM comparator event callback function.

Param comparator MCPWM comparator handle
Param edata MCPWM comparator event data, fed by driver
Param user_ctx User data, set in `mcpwm_comparator_register_event_callbacks()`
Return Whether a high priority task has been waken up by this function

typedef bool (***mcpwm_capture_event_cb_t**)(*mcpwm_cap_channel_handle_t* cap_channel, const *mcpwm_capture_event_data_t* *edata, void *user_ctx)

MCPWM capture event callback function.

Param cap_channel MCPWM capture channel handle
Param edata MCPWM capture event data, fed by driver
Param user_ctx User data, set in `mcpwm_capture_channel_register_event_callbacks()`
Return Whether a high priority task has been waken up by this function

Header File

- components/hal/include/hal/mcpwm_types.h
- This header file can be included with:

```
#include "hal/mcpwm_types.h"
```

Type Definitions

typedef *soc_periph_mcpwm_timer_clk_src_t* **mcpwm_timer_clock_source_t**

MCPWM timer clock source.

typedef *soc_periph_mcpwm_capture_clk_src_t* **mcpwm_capture_clock_source_t**

MCPWM capture clock source.

typedef *soc_periph_mcpwm_carrier_clk_src_t* **mcpwm_carrier_clock_source_t**

MCPWM carrier clock source.

Enumerations

enum **mcpwm_timer_direction_t**

MCPWM timer count direction.

Values:

enumerator **MCPWM_TIMER_DIRECTION_UP**

Counting direction: Increase

enumerator **MCPWM_TIMER_DIRECTION_DOWN**

Counting direction: Decrease

enum **mcpwm_timer_event_t**

MCPWM timer events.

Values:

enumerator **MCPWM_TIMER_EVENT_EMPTY**

MCPWM timer counts to zero (i.e. counter is empty)

enumerator **MCPWM_TIMER_EVENT_FULL**

MCPWM timer counts to peak (i.e. counter is full)

enumerator **MCPWM_TIMER_EVENT_INVALID**

MCPWM timer invalid event

enum **mcpwm_timer_count_mode_t**

MCPWM timer count modes.

Values:

enumerator **MCPWM_TIMER_COUNT_MODE_PAUSE**

MCPWM timer paused

enumerator **MCPWM_TIMER_COUNT_MODE_UP**

MCPWM timer counting up

enumerator **MCPWM_TIMER_COUNT_MODE_DOWN**

MCPWM timer counting down

enumerator **MCPWM_TIMER_COUNT_MODE_UP_DOWN**

MCPWM timer counting up and down

enum **mcpwm_timer_start_stop_cmd_t**

MCPWM timer commands, specify the way to start or stop the timer.

Values:

enumerator **MCPWM_TIMER_STOP_EMPTY**

MCPWM timer stops when next count reaches zero

enumerator **MCPWM_TIMER_STOP_FULL**

MCPWM timer stops when next count reaches peak

enumerator **MCPWM_TIMER_START_NO_STOP**

MCPWM timer starts couting, and don't stop until received stop command

enumerator **MCPWM_TIMER_START_STOP_EMPTY**

MCPWM timer starts counting and stops when next count reaches zero

enumerator **MCPWM_TIMER_START_STOP_FULL**

MCPWM timer starts counting and stops when next count reaches peak

enum **mcpwm_generator_action_t**

MCPWM generator actions.

Values:

enumerator **MCPWM_GEN_ACTION_KEEP**

Generator action: Keep the same level

enumerator **MCPWM_GEN_ACTION_LOW**

Generator action: Force to low level

enumerator **MCPWM_GEN_ACTION_HIGH**

Generator action: Force to high level

enumerator **MCPWM_GEN_ACTION_TOGGLE**

Generator action: Toggle level

enum **mcpwm_operator_brake_mode_t**

MCPWM operator brake mode.

Values:

enumerator **MCPWM_OPER_BRAKE_MODE_CBC**

Brake mode: CBC (cycle by cycle)

enumerator **MCPWM_OPER_BRAKE_MODE_OST**

Brake mode: OST (one shot)

enumerator **MCPWM_OPER_BRAKE_MODE_INVALID**

MCPWM operator invalid brake mode

enum **mcpwm_capture_edge_t**

MCPWM capture edge.

Values:

enumerator **MCPWM_CAP_EDGE_POS**

Capture on the positive edge

enumerator **MCPWM_CAP_EDGE_NEG**

Capture on the negative edge

enum **mcpwm_comparator_etm_event_type_t**

MCPWM comparator specific events that supported by the ETM module.

Values:

enumerator **MCPWM_CMPR_ETM_EVENT_EQUAL**

The count value equals the value of comparator

enumerator **MCPWM_CMPR_ETM_EVENT_MAX**

Maximum number of comparator events

警告： This document is not updated for ESP32P4 yet, so some of the content may not be correct.

This warning was automatically inserted due to the source file being in the *add_warnings_pages* list.

2.5.20 并行 IO

简介

并行 IO 外设是一个通用的并行接口，可用于连接外部设备，如 LED 矩阵、LCD 显示器、打印机和摄像头。该外设具有独立的 TX 和 RX 单元。每个单元最多可以具有 8 或 16 个数据信号以及 1 或 2 个时钟信号。[1]

并行 IO 外设的 TX 和 RX 驱动程序有各自独立的设计，可分别通过包含头文件 `driver/parlio_tx.h` 或 `driver/parlio_rx.h` 来使用。

应用示例

- 带有 HUB75 接口的简单 REG LED 矩阵：peripherals/parlio/parlio_tx/simple_rgb_led_matrix。
- 逻辑分析仪：peripherals/parlio/parlio_rx/logic_analyzer。

[1] 不同的 ESP 系列芯片可能具有不同数量的并行 IO TX/RX 实例，数据总线的最大宽度也会有所不同。更多信息，请参阅 **ESP32-P4 技术参考手册 > 并行 IO 控制器 (PARLIO) 章节** [PDF]。驱动程序允许你创建更多的驱动程序对象，但当所有可用的硬件资源都被用完时，将会显示返回错误。在进行资源分配时，请时刻注意返回值 (e.g., *parlio_new_tx_unit()*)。

API 参考

Header File

- components/esp_driver_parlio/include/driver/parlio_tx.h
- This header file can be included with:

```
#include "driver/parlio_tx.h"
```

- This header file is a part of the API provided by the `esp_driver_parlio` component. To declare that your component depends on `esp_driver_parlio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_parlio
```

or

```
PRIV_REQUIRES esp_driver_parlio
```

Functions

esp_err_t **parlio_new_tx_unit** (const *parlio_tx_unit_config_t* *config, *parlio_tx_unit_handle_t* *ret_unit)

Create a Parallel IO TX unit.

参数

- **config** -- **[in]** Parallel IO TX unit configuration
- **ret_unit** -- **[out]** Returned Parallel IO TX unit handle

返回

- ESP_OK: Create Parallel IO TX unit successfully
- ESP_ERR_INVALID_ARG: Create Parallel IO TX unit failed because of invalid argument
- ESP_ERR_NO_MEM: Create Parallel IO TX unit failed because of out of memory
- ESP_ERR_NOT_FOUND: Create Parallel IO TX unit failed because all TX units are used up and no more free one
- ESP_ERR_NOT_SUPPORTED: Create Parallel IO TX unit failed because some feature is not supported by hardware, e.g. clock gating
- ESP_FAIL: Create Parallel IO TX unit failed because of other error

esp_err_t **parlio_del_tx_unit** (*parlio_tx_unit_handle_t* unit)

Delete a Parallel IO TX unit.

参数 **unit** -- **[in]** Parallel IO TX unit that created by `parlio_new_tx_unit`

返回

- ESP_OK: Delete Parallel IO TX unit successfully
- ESP_ERR_INVALID_ARG: Delete Parallel IO TX unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete Parallel IO TX unit failed because it is still in working
- ESP_FAIL: Delete Parallel IO TX unit failed because of other error

esp_err_t **parlio_tx_unit_enable** (*parlio_tx_unit_handle_t* unit)

Enable the Parallel IO TX unit.

备注: This function will transit the driver state from init to enable

备注: This function will acquire a PM lock that might be installed during channel allocation

备注: If there're transaction pending in the queue, this function will pick up the first one and start the transfer

参数 **unit** -- **[in]** Parallel IO TX unit that created by `parlio_new_tx_unit`

返回

- ESP_OK: Enable Parallel IO TX unit successfully
- ESP_ERR_INVALID_ARG: Enable Parallel IO TX unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable Parallel IO TX unit failed because it is already enabled
- ESP_FAIL: Enable Parallel IO TX unit failed because of other error

esp_err_t **parlio_tx_unit_disable**(*parlio_tx_unit_handle_t* unit)

Disable the Parallel IO TX unit.

备注: This function will transit the driver state from enable to init

备注: This function will release the PM lock that might be installed during channel allocation

备注: If one transaction is undergoing, this function will terminate it immediately

参数 **unit** -- **[in]** Parallel IO TX unit that created by `parlio_new_tx_unit`

返回

- ESP_OK: Disable Parallel IO TX unit successfully
- ESP_ERR_INVALID_ARG: Disable Parallel IO TX unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable Parallel IO TX unit failed because it's not enabled yet
- ESP_FAIL: Disable Parallel IO TX unit failed because of other error

esp_err_t **parlio_tx_unit_register_event_callbacks**(*parlio_tx_unit_handle_t* tx_unit, const *parlio_tx_event_callbacks_t* *cbs, void *user_data)

Set event callbacks for Parallel IO TX unit.

备注: User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

备注: When CONFIG_PARLIO_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. The `user_data` should also reside in SRAM.

参数

- **tx_unit** -- **[in]** Parallel IO TX unit that created by `parlio_new_tx_unit`
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

esp_err_t **parlio_tx_unit_transmit**(*parlio_tx_unit_handle_t* tx_unit, const void *payload, size_t payload_bits, const *parlio_transmit_config_t* *config)

Transmit data on by Parallel IO TX unit.

备注： After the function returns, it doesn't mean the transaction is finished. This function only constructs a transaction structure and push into a queue.

参数

- **tx_unit** -- **[in]** Parallel IO TX unit that created by `parlio_new_tx_unit`
- **payload** -- **[in]** Pointer to the data to be transmitted
- **payload_bits** -- **[in]** Length of the data to be transmitted, in bits
- **config** -- **[in]** Transmit configuration

返回

- ESP_OK: Transmit data successfully
- ESP_ERR_INVALID_ARG: Transmit data failed because of invalid argument
- ESP_ERR_INVALID_STATE: Transmit data failed because the Parallel IO TX unit is not enabled
- ESP_FAIL: Transmit data failed because of other error

esp_err_t **parlio_tx_unit_wait_all_done** (*parlio_tx_unit_handle_t* tx_unit, int timeout_ms)

Wait for all pending TX transactions done.

参数

- **tx_unit** -- **[in]** Parallel IO TX unit that created by `parlio_new_tx_unit`
- **timeout_ms** -- **[in]** Timeout in milliseconds, `-1` means to wait forever

返回

- ESP_OK: All pending TX transactions is finished and recycled
- ESP_ERR_INVALID_ARG: Wait for all pending TX transactions done failed because of invalid argument
- ESP_ERR_TIMEOUT: Wait for all pending TX transactions done timeout
- ESP_FAIL: Wait for all pending TX transactions done failed because of other error

Structures

struct **parlio_tx_unit_config_t**

Parallel IO TX unit configuration.

Public Members

parlio_clock_source_t **clk_src**

Parallel IO internal clock source

gpio_num_t **clk_in_gpio_num**

If the clock source is input from external, set the corresponding GPIO number. Otherwise, set to `-1` and the driver will use the internal `clk_src` as clock source. This option has higher priority than `clk_src`

uint32_t **input_clk_src_freq_hz**

Frequency of the input clock source, valid only if `clk_in_gpio_num` is not `-1`

uint32_t **output_clk_freq_hz**

Frequency of the output clock. It's divided from either internal `clk_src` or external clock source

size_t **data_width**

Parallel IO data width, can set to 1/2/4/8/..., but can't bigger than PARLIO_TX_UNIT_MAX_DATA_WIDTH

gpio_num_t **data_gpio_nums**[PARLIO_TX_UNIT_MAX_DATA_WIDTH]

Parallel IO data GPIO numbers, if any GPIO is not used, you can set it to `-1`

gpio_num_t **clk_out_gpio_num**

GPIO number of the output clock signal, the clock is synced with TX data

gpio_num_t **valid_gpio_num**

GPIO number of the valid signal, which stays high when transferring data. Note that, the valid signal will always occupy the MSB data bit

size_t **trans_queue_depth**

Depth of internal transaction queue

size_t **max_transfer_size**

Maximum transfer size in one transaction, in bytes. This decides the number of DMA nodes will be used for each transaction

size_t **dma_burst_size**

DMA burst size, in bytes

parlio_sample_edge_t **sample_edge**

Parallel IO sample edge

parlio_bit_pack_order_t **bit_pack_order**

Set the order of packing the bits into bytes (only works when `data_width` < 8)

uint32_t **clk_gate_en**

Enable TX clock gating, the output clock will be controlled by the MSB bit of the data bus, i.e. by data_gpio_nums[PARLIO_TX_UNIT_MAX_DATA_WIDTH-1]. High level to enable the clock output, low to disable

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

struct *parlio_tx_unit_config_t*::[anonymous] **flags**

Extra configuration flags

struct **parlio_tx_done_event_data_t**

Type of Parallel IO TX done event data.

struct **parlio_tx_event_callbacks_t**

Group of Parallel IO TX callbacks.

备注: The callbacks are all running under ISR environment

备注： When CONFIG_PARLIO_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

Public Members

parlio_tx_done_callback_t **on_trans_done**

Event callback, invoked when one transmission is finished

struct **parlio_transmit_config_t**

Parallel IO transmit configuration.

Public Members

uint32_t **idle_value**

The value on the data line when the parallel IO is in idle state

uint32_t **queue_nonblocking**

If set, when the transaction queue is full, driver will not block the thread but return directly

struct *parlio_transmit_config_t*::[anonymous] **flags**

Transmit specific config flags

Type Definitions

typedef bool (***parlio_tx_done_callback_t**)(*parlio_tx_unit_handle_t* tx_unit, const *parlio_tx_done_event_data_t* *edata, void *user_ctx)

Prototype of parlio tx event callback.

Param tx_unit **[in]** Parallel IO TX unit that created by `parlio_new_tx_unit`

Param edata **[in]** Point to Parallel IO TX event data. The lifecycle of this pointer memory is inside this function, user should copy it into static memory if used outside this function.

Param user_ctx **[in]** User registered context, passed from `parlio_tx_unit_register_event_callbacks`

Return Whether a high priority task has been waken up by this callback function

Header File

- components/esp_driver_parlio/include/driver/parlio_rx.h
- This header file can be included with:

```
#include "driver/parlio_rx.h"
```

- This header file is a part of the API provided by the `esp_driver_parlio` component. To declare that your component depends on `esp_driver_parlio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_parlio
```

or

```
PRIV_REQUIRES esp_driver_parlio
```

Functions

esp_err_t **parlio_new_rx_unit**(const *parlio_rx_unit_config_t* *config, *parlio_rx_unit_handle_t* *ret_unit)

Create a Parallel IO RX unit.

参数

- **config** -- **[in]** Parallel IO RX unit configuration
- **ret_unit** -- **[out]** Returned Parallel IO RX unit handle

返回

- ESP_ERR_INVALID_ARG Invalid arguments in the parameter list or the rx unit configuration
- ESP_ERR_NOT_FOUND No available rx unit found
- ESP_ERR_NO_MEM No enough memory for the rx unit resources
- ESP_OK Success to allocate the rx unit

esp_err_t **parlio_del_rx_unit**(*parlio_rx_unit_handle_t* rx_unit)

Delete a Parallel IO RX unit.

参数 **rx_unit** -- **[in]** Parallel IO RX unit handle that created by `parlio_new_rx_unit`

返回

- ESP_ERR_INVALID_ARG rx_unit is NULL
- ESP_ERR_INVALID_STATE The rx unit is enabled, can't delete an enabled rx unit
- ESP_OK Success to delete the rx unit

esp_err_t **parlio_new_rx_level_delimiter**(const *parlio_rx_level_delimiter_config_t* *config, *parlio_rx_delimiter_handle_t* *ret_delimiter)

Create a level delimiter.

备注: This function only allocate the software resources, the hardware configurations will lazy installed while the transaction that using this delimiter start processing

---

备注: The enable signal must be aligned with the valid data.

备注: There're at most `SOC_PARLIO_RX_UNIT_MAX_DATA_WIDTH - 1` IO pins left for RXD

参数

- **config** -- **[in]** Level delimiter configuration
- **ret_delimiter** -- **[out]** Returned delimiter handle

返回

- ESP_ERR_INVALID_ARG Invalid arguments in the parameter list or the level delimiter configuration
- ESP_ERR_NO_MEM No enough memory for the level delimiter resources
- ESP_OK Success to allocate the level delimiter

esp_err_t **parlio_new_rx_pulse_delimiter**(const *parlio_rx_pulse_delimiter_config_t* *config, *parlio_rx_delimiter_handle_t* *ret_delimiter)

Create a pulse delimiter.

备注: This function only allocate the software resources, the hardware configurations will lazy installed while the transaction that using this delimiter start processing

---

备注: There're at most `SOC_PARLIO_RX_UNIT_MAX_DATA_WIDTH - 1` IO pins left for RXD

参数

- **config** -- **[in]** Pulse delimiter configuration
- **ret_delimiter** -- **[out]** Returned delimiter handle

返回

- ESP_ERR_INVALID_ARG Invalid arguments in the parameter list or the pulse delimiter configuration
- ESP_ERR_NO_MEM No enough memory for the pulse delimiter resources
- ESP_OK Success to allocate the pulse delimiter

esp_err_t **parlio_new_rx_soft_delimiter**(const *parlio_rx_soft_delimiter_config_t* *config, *parlio_rx_delimiter_handle_t* *ret_delimiter)

Create a pulse delimiter.

备注: This function only allocate the software resources, the hardware configurations will lazy installed while the transaction that using this delimiter start processing

---

参数

- **config** -- **[in]** Soft delimiter configuration
- **ret_delimiter** -- **[out]** Returned delimiter handle

返回

- ESP_ERR_INVALID_ARG Invalid arguments in the parameter list or the soft delimiter configuration
- ESP_ERR_NO_MEM No enough memory for the soft delimiter resources
- ESP_OK Success to allocate the soft delimiter

esp_err_t **parlio_rx_soft_delimiter_start_stop**(*parlio_rx_unit_handle_t* rx_unit, *parlio_rx_delimiter_handle_t* delimiter, bool start_stop)

Start/stop the soft delimiter.

备注: Soft delimiter need to start or stop manually because it has no validating/enabling signal to indicate the data has started or stopped

---

参数

- **rx_unit** -- **[in]** Parallel IO RX unit handle that created by `parlio_new_rx_unit`
- **delimiter** -- **[in]** Delimiter handle
- **start_stop** -- **[in]** Set true to start, set false to stop

返回

- ESP_ERR_INVALID_ARG Invalid arguments in the parameter list or not soft delimiter
- ESP_ERR_INVALID_STATE The rx unit not enabled
- ESP_OK Success to start or stop the soft delimiter

esp_err_t **parlio_del_rx_delimiter**(*parlio_rx_delimiter_handle_t* delimiter)

Delete the delimiter.

备注: To delete the delimiter safely, please delete it after disable all the RX units

---

参数 **delimiter** -- **[in]** Delimiter handle

返回

- ESP_ERR_INVALID_ARG The input delimiter is NULL
- ESP_ERR_INVALID_STATE The delimiter is on receiving
- ESP_OK Success to delete the delimiter

esp_err_t **parlio_rx_unit_enable**(*parlio_rx_unit_handle_t* rx_unit, bool reset_queue)

Enable the Parallel IO RX unit.

参数

- **rx_unit** -- **[in]** Parallel IO RX unit handle that created by `parlio_new_rx_unit`
- **reset_queue** -- **[in]** Whether to reset the receiving queue. If set to false, the legacy receive transactions in the queue are still available, If set to true, the legacy receive transactions in the queue are dropped.

返回

- ESP_ERR_INVALID_ARG The input rx_unit is NULL
- ESP_ERR_INVALID_STATE The rx unit has been enabled
- ESP_OK Success to enable the rx unit

esp_err_t **parlio_rx_unit_disable**(*parlio_rx_unit_handle_t* rx_unit)

Disable the Parallel IO RX unit.

参数 **rx_unit** -- **[in]** Parallel IO RX unit handle that created by `parlio_new_rx_unit`

返回

- ESP_ERR_INVALID_ARG The input rx_unit is NULL
- ESP_ERR_INVALID_STATE The rx unit has been disabled
- ESP_OK Success to disable the rx unit

esp_err_t **parlio_rx_unit_receive**(*parlio_rx_unit_handle_t* rx_unit, void *payload, size_t payload_size, const *parlio_receive_config_t* *recv_cfg)

Receive data by Parallel IO RX unit.

备注: This is a non-blocking and asynchronous function. To block or realize synchronous receive, please call `parlio_rx_unit_wait_all_done` after this function

---

备注: The receive transaction will start immediately when there is not other transaction on receiving, Otherwise it will be sent to the transaction queue to wait for the bus.

参数

- **rx_unit** -- **[in]** Parallel IO RX unit handle that created by `parlio_new_rx_unit`
- **payload** -- **[in]** The payload buffer pointer
- **payload_size** -- **[in]** The size of the payload buffer, in bytes.
- **recv_cfg** -- **[in]** The configuration of this receive transaction

返回

- ESP_ERR_INVALID_ARG Invalid arguments in the parameter list or the receive configuration
- ESP_ERR_NO_MEM No memory for the internal DMA buffer (only when *parlio_receive_config_t::indirect_mount* enabled)
- ESP_ERR_INVALID_STATE Transaction queue is full, failed to queue the current transaction. Or the internal buffer is under using by an infinite transaction, can't allocate a new one
- ESP_OK Success to queue the current receiving transaction

esp_err_t **parlio_rx_unit_wait_all_done**(*parlio_rx_unit_handle_t* rx_unit, int timeout_ms)

Wait for all pending RX transactions done.

备注: This function will block until all receiving transactions done or timeout. When timeout occurs, either the timeout limitation too short for all transactions done, or the peripheral got stuck and no more interrupts trigger (e.g., external clock stopped).

参数

- **rx_unit** -- **[in]** Parallel IO RX unit handle that created by `parlio_new_rx_unit`
- **timeout_ms** -- **[in]** Timeout in milliseconds, `-1` means to wait forever (software timeout)

返回

- ESP_ERR_INVALID_ARG The input rx_unit is NULL
- ESP_ERR_TIMEOUT Wait for all transactions done timeout
- ESP_OK All transaction done

esp_err_t **parlio_rx_unit_register_event_callbacks**(*parlio_rx_unit_handle_t* rx_unit, const *parlio_rx_event_callbacks_t* *cbs, void *user_data)

Register event callbacks for Parallel IO RX unit.

参数

- **rx_unit** -- **[in]** Parallel IO RX unit handle that created by `parlio_new_rx_unit`
- **cbs** -- **[in]** Callback group, set callback to NULL to deregister the corresponding callback (callback group pointer shouldn't be NULL)
- **user_data** -- **[in]** User specified data that will be transported to the callbacks

返回

- ESP_ERR_INVALID_ARG The input rx_unit is NULL
- ESP_ERR_INVALID_STATE The rx unit has been enabled, callback should be registered before enabling the unit
- ESP_OK Success to register the callbacks

Structures

struct **parlio_rx_unit_config_t**

Parallel IO RX unit configuration.

Public Members

size_t **trans_queue_depth**

Depth of internal transaction queue

size_t **max_recv_size**

Maximum receive size in one transaction, in bytes. This decides the number of DMA nodes will be used for each transaction

size_t **data_width**

Parallel IO data width, can set to 1/2/4/8/..., but can't be greater than PARLIO_RX_UNIT_MAX_DATA_WIDTH

parlio_clock_source_t **clk_src**

Parallel IO clock source

uint32_t **ext_clk_freq_hz**

The external source clock frequency. Only be valid when select PARLIO_CLK_SRC_EXTERNAL as clock source

uint32_t **exp_clk_freq_hz**

The expected sample/bit clock frequency, which is divided from the internal or external clock regarding the clock source

gpio_num_t **clk_in_gpio_num**

The the external clock input pin. Only be valid when select PARLIO_CLK_SRC_EXTERNAL as clock source. Set to -1 if not needed

gpio_num_t **clk_out_gpio_num**

The sample/bit clock output pin. Set to -1 if not needed

gpio_num_t **valid_gpio_num**

GPIO number of the valid signal. The signal on this pin is used to indicate whether the data on the data lines are valid. Only takes effect when using level or pulse delimiter, set to `-1` if only use the soft delimiter

gpio_num_t **data_gpio_nums**[PARLIO_RX_UNIT_MAX_DATA_WIDTH]

Parallel IO data GPIO numbers, set to `-1` if it's not used, The driver will take [0 .. (data_width - 1)] as the data pins

uint32_t **free_clk**

Whether the input external clock is a free-running clock. A free-running clock will always keep running (e.g. I2S bclk), a non-free-running clock will start when there are data transporting and stop when the bus idle (e.g. SPI). This flag only takes effect when select PARLIO_CLK_SRC_EXTERNAL as clock source

uint32_t **clk_gate_en**

Enable RX clock gating, only available when the clock direction is output(not supported on ESP32-C6) the output clock will be controlled by the valid gpio, i.e. high level of valid gpio to enable the clock output, low to disable

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

uint32_t **io_no_init**

Set to skip initializing the GPIO, but only attach the pralio rx signals to those GPIOs via IO Matrix. So that the signals that have attached to those GPIO won't be overwritten. Mainly used for self communication or self monitoring

struct *parlio_rx_unit_config_t*::[anonymous] **flags**

RX driver flags

struct **parlio_rx_level_delimiter_config_t**

Configuration of level delimiter.

Public Members

uint32_t **valid_sig_line_id**

The data line id of valid/enable signal. The selected data line will be used as the valid/enable signal (i.e. level signal) in this delimiter. As the line of valid/enable signal is shared with the data line, this line_id will be conflict with the data line if set the id within 'data_width', therefore the range is (data_width, PARLIO_RX_UNIT_MAX_DATA_WIDTH].

parlio_sample_edge_t **sample_edge**

Parallel IO sample edge

parlio_bit_pack_order_t **bit_pack_order**

Set how we pack the bits into one bytes

uint32_t **eof_data_len**

Set the data length to trigger the End Of Frame (EOF, i.e. transaction done) interrupt, if the data length is set to `0`, that mean the EOF will only triggers when the enable signal inactivated

uint32_t **timeout_ticks**

The number of source clock ticks to trigger timeout interrupt. Set 0 to disable the receive timeout interrupt The timeout counter starts when the valid/enable signal is invalid/disabled.

uint32_t **active_low_en**

Set true to set the valid signal active when the level is low, otherwise, the valid signal becomes active when its level is high

struct *parlio_rx_level_delimiter_config_t*::[anonymous] **flags**

Extra flags

struct **parlio_rx_pulse_delimiter_config_t**

Configuration of pulse delimiter.

Public Members

uint32_t **valid_sig_line_id**

The data line id of valid/enable signal. The selected data line will be used as the valid/enable signal (i.e. pulse signal) in this delimiter. As the line of valid/enable signal is shared with the data line, this line_id will be conflict with the data line if set the id within 'data_width', therefore the range is (data_width, PARLIO_RX_UNIT_MAX_DATA_WIDTH].

parlio_sample_edge_t **sample_edge**

Parallel IO sample edge

parlio_bit_pack_order_t **bit_pack_order**

Set how we pack the bits into one bytes

uint32_t **eof_data_len**

Set the data length to trigger the End Of Frame (EOF, i.e. transaction done) interrupt, if the data length is set to `0`, that mean the EOF will only triggers when the end pulse detected, please ensure there is an end pulse for a frame and `has_end_pulse` flag is set

uint32_t **timeout_ticks**

The number of source clock ticks to trigger timeout interrupt. Set 0 to disable the receive timeout interrupt The timeout counter starts when the valid/enable signal is invalid/disabled.

uint32_t **start_bit_included**

Whether data bit is included in the start pulse

uint32_t **end_bit_included**

Whether data bit is included in the end pulse, only valid when `has_end_pulse` is true

uint32_t **has_end_pulse**

Whether there's an end pulse to terminate the transaction, if no, the transaction will be terminated by user configured transaction length

uint32_t **pulse_invert**

Whether to invert the pulse

struct *parlio_rx_pulse_delimiter_config_t*::[anonymous] **flags**

Extra flags

struct **parlio_rx_soft_delimiter_config_t**

Configuration of soft delimiter.

Public Members

parlio_sample_edge_t **sample_edge**

Parallel IO sample edge

parlio_bit_pack_order_t **bit_pack_order**

Set how we pack the bits into one bytes, set 1 to pack the bits into a byte from LSB, otherwise from MSB

uint32_t **eof_data_len**

Set the data length to trigger the End Of Frame (EOF, i.e. transaction done) interrupt, if the data length is set to `0`, that mean the EOF will only triggers when the end pulse detected, please ensure there is an end pulse for a frame and *`parlio_rx_pulse_delimiter_config_t::has_end_pulse`* flag is set

uint32_t **timeout_ticks**

The number of APB clock ticks to trigger timeout interrupt. Set 0 to disable the receive timeout interrupt

struct **parlio_receive_config_t**

Configuration of a receive transaction.

Public Members

parlio_rx_delimiter_handle_t **delimiter**

The delimiter of this receiving transaction

uint32_t **partial_rx_en**

Whether this is an infinite transaction that supposed to receive continuously and partially

uint32_t **indirect_mount**

This flag only take effect when `partial_rx_en` is enabled. Enable this flag, an INTERNAL DMA buffer will be mounted to the DMA descriptor instead, The data will be copy to the payload in every interrupt. So that to guarantee the payload buffer is valid during the `on_receive_done` callback. Either `partial_rx_en` or `indirect_mount` is disabled, the user given finite payload will be mounted to the DMA descriptor directly. By default, the user given receive payload will be mounted to the DMA descriptor directly.

struct *parlio_receive_config_t*::[anonymous] **flags**

Extra flags

struct **parlio_rx_event_data_t**

Event callback data.

Public Members

parlio_rx_delimiter_handle_t **delimiter**

The current delimiter of this receiving event

void ***data**

The data buffer address that just finished receiving

size_t **recv_bytes**

The number of received bytes in the data buffer

struct **parlio_rx_event_callbacks_t**

Parallel IO RX event callbacks.

Public Members

parlio_rx_callback_t **on_partial_receive**

Callback of received partial data

parlio_rx_callback_t **on_receive_done**

Callback of receiving transaction done

parlio_rx_callback_t **on_timeout**

Callback of hardware receiving timeout

Type Definitions

typedef bool (***parlio_rx_callback_t**)(*parlio_rx_unit_handle_t* rx_unit, const *parlio_rx_event_data_t* *edata, void *user_data)

The template of the Parallel IO RX callback function.

Param rx_unit **[in]** Parallel IO RX unit handle that given from ISR
Param edata **[in]** The event data that given from ISR
Param user_data **[in]** The user specified data that given while registering the callbacks
Return

- True: to awoke high priority tasks
- False: not to awoke high priority tasks

Header File

- components/esp_driver_parlio/include/driver/parlio_types.h
- This header file can be included with:

```
#include "driver/parlio_types.h"
```

- This header file is a part of the API provided by the `esp_driver_parlio` component. To declare that your component depends on `esp_driver_parlio`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_parlio
```

or

```
PRIV_REQUIRES esp_driver_parlio
```

Type Definitions

typedef struct parlio_tx_unit_t ***parlio_tx_unit_handle_t**

Type of Parallel IO TX unit handle.

typedef struct parlio_rx_unit_t ***parlio_rx_unit_handle_t**

Type of Parallel IO RX unit handle.

typedef struct parlio_rx_delimiter_t ***parlio_rx_delimiter_handle_t**

Type of Parallel IO RX frame delimiter handle.

Header File

- components/hal/include/hal/parlio_types.h
- This header file can be included with:

```
#include "hal/parlio_types.h"
```

Macros

PARLIO_TX_UNIT_MAX_DATA_WIDTH

Maximum data width of TX unit.

PARLIO_RX_UNIT_MAX_DATA_WIDTH

Maximum data width of RX unit.

Type Definitions

typedef *soc_periph_parlio_clk_src_t* **parlio_clock_source_t**

Parallel IO clock source.

> **备注：** User should select the clock source based on the power and resolution requirement

Enumerations

enum **parlio_sample_edge_t**

Parallel IO sample edge.

Values:

enumerator **PARLIO_SAMPLE_EDGE_NEG**

Sample data on falling edge of clock

enumerator **PARLIO_SAMPLE_EDGE_POS**

Sample data on rising edge of clock

enum **parlio_bit_pack_order_t**

Parallel IO bit packing order.

Data in memory: Byte 0: MSB < B0.7 B0.6 B0.5 B0.4 B0.3 B0.2 B0.1 B0.0 > LSB Byte 1: MSB < B1.7 B1.6 B1.5 B1.4 B1.3 B1.2 B1.1 B1.0 > LSB

Output on line (PARLIO_BIT_PACK_ORDER_LSB): Cycle 0 Cycle 1 Cycle 2 —> time GPIO 0: B0.0 B0.4 B1.0 GPIO 1: B0.1 B0.5 B1.1 GPIO 2: B0.2 B0.6 B1.2 GPIO 3: B0.3 B0.7 B1.3

Output on line (PARLIO_BIT_PACK_ORDER_MSB): Cycle 0 Cycle 1 Cycle 2 —> time GPIO 0: B0.4 B0.0 B1.4 GPIO 1: B0.5 B0.1 B1.5 GPIO 2: B0.6 B0.2 B1.6 GPIO 3: B0.7 B0.3 B1.7

Values:

enumerator **PARLIO_BIT_PACK_ORDER_LSB**

Bit pack order: LSB

enumerator **PARLIO_BIT_PACK_ORDER_MSB**

Bit pack order: MSB

2.5.21 脉冲计数器 (PCNT)

概述

PCNT 用于统计输入信号的上升沿和/或下降沿的数量。ESP32-P4 集成了多个脉冲计数单元,[1] 每个单元都是包含多个通道的独立计数器。通道可独立配置为统计上升沿或下降沿数量的递增计数器或递减计数器。

PCNT 通道可检测 **边沿**信号及 **电平**信号。对于比较简单的应用，检测边沿信号就足够了。PCNT 通道可检测上升沿信号、下降沿信号，同时也能设置为递增计数，递减计数，或停止计数。电平信号就是所谓的 **控制信号**，可用来控制边沿信号的计数模式。通过设置电平信号与边沿信号的检测模式，PCNT 单元可用作 **正交解码器**。

每个 PCNT 单元还包含一个滤波器，用于滤除线路毛刺。

PCNT 模块通常用于：

- 对一段时间内的脉冲计数，进而计算得到周期信号的频率；
- 对正交信号进行解码，进而获得速度和方向信息。

功能描述

PCNT 的功能从以下几个方面进行说明：

- 分配资源 - 说明如何通过配置分配 PCNT 单元和通道，以及在相应操作完成之后，如何回收单元和通道。
- 设置通道操作 - 说明如何设置通道针对不同信号沿和电平进行操作。
- *PCNT* 观察点 - 说明如何配置观察点，即当计数达到某个数值时，命令 PCNT 单元触发某个事件。
- 注册事件回调函数 - 说明如何将您的代码挂载到观察点事件的回调函数上。
- 设置毛刺滤波器 - 说明如何使能毛刺滤波器并设置其时序参数。
- 使用外部清零信号 - 说明如何使能外部清零信号并设置其参数。
- 使能和禁用单元 - 说明如何使能和关闭 PCNT 单元。

[1] 在不同的 ESP 芯片系列中，PCNT 单元和通道的数量可能会有差异，具体信息请参考 [TRM]。驱动程序对通道申请数量不做限制，但当硬件资源用尽时，驱动程序将返回错误。因此分配资源时，应注意检查返回值，如`pcnt_new_unit()`。

- 控制单元 *IO* 操作 - 说明 PCNT 单元的 IO 控制功能，例如使能毛刺滤波器，开启和停用 PCNT 单元，获取和清除计数。
- 电源管理 - 说明哪些功能会阻止芯片进入低功耗模式。
- 支持 *IRAM* 安全中断 - 说明在缓存禁用的情况下，如何执行 PCNT 中断和 IO 控制功能。
- 支持线程安全 - 列出线程安全的 API。
- 支持的 *Kconfig* 选项 - 列出了支持的 Kconfig 选项，这些选项可实现不同的驱动效果。

分配资源 PCNT 单元和通道分别用*pcnt_unit_handle_t* 与*pcnt_channel_handle_t* 表示。所有的可用单元和通道都由驱动在资源池中进行维护，无需了解底层实例 ID。

安装 PCNT 单元 安装 PCNT 单元时，需要先完成配置*pcnt_unit_config_t*：

- *pcnt_unit_config_t::low_limit* 与*pcnt_unit_config_t::high_limit* 用于指定内部计数器的最小值和最大值。当计数器超过任一限值时，计数器将归零。
- *pcnt_unit_config_t::accum_count* 用于设置是否需要软件在硬件计数值溢出的时候进行累加保存，这有助于“拓宽”计数器的实际位宽。默认情况下，计数器的位宽最高只有 16 比特。请参考计数溢出补偿 了解如何利用此功能来补偿硬件计数器的溢出损失。
- *pcnt_unit_config_t::intr_priority* 设置中断的优先级。如果设置为 0，则会分配一个默认优先级的中断，否则会使用指定的优先级。

备注: 由于所有 PCNT 单元共享一个中断源，安装多个 PCNT 单元时请确保每个单元的中断优先级*pcnt_unit_config_t::intr_priority* 一致。

调用函数*pcnt_new_unit()* 并将*pcnt_unit_config_t* 作为其输入值，可对 PCNT 单元进行分配和初始化。该函数正常运行时，会返回一个 PCNT 单元句柄。没有可用的 PCNT 单元时（即 PCNT 单元全部被占用），该函数会返回错误*ESP_ERR_NOT_FOUND*。可用的 PCNT 单元总数记录在*SOC_PCNT_UNITS_PER_GROUP* 中，以供参考。

如果不再需要之前创建的某个 PCNT 单元，建议通过调用*pcnt_del_unit()* 来回收该单元，从而该单元可用于其他用途。删除某个 PCNT 单元之前，需要满足以下条件：

- 该单元处于初始状态，即该单元要么已经被*pcnt_unit_disable()* 禁用，要么尚未使能。
- 附属于该单元的通道已全部被*pcnt_del_channel()* 删除。

```
#define EXAMPLE_PCNT_HIGH_LIMIT 100
#define EXAMPLE_PCNT_LOW_LIMIT  -100

pcnt_unit_config_t unit_config = {
    .high_limit = EXAMPLE_PCNT_HIGH_LIMIT,
    .low_limit = EXAMPLE_PCNT_LOW_LIMIT,
};
pcnt_unit_handle_t pcnt_unit = NULL;
ESP_ERROR_CHECK(pcnt_new_unit(&unit_config, &pcnt_unit));
```

安装 PCNT 通道 安装 PCNT 通道时，需要先初始化*pcnt_chan_config_t*，然后调用*pcnt_new_channel()*。对*pcnt_chan_config_t* 配置如下所示：

- *pcnt_chan_config_t::edge_gpio_num* 与*pcnt_chan_config_t::level_gpio_num* 用于指定 **边沿**信号和 **电平**信号对应的 GPIO 编号。请注意，这两个参数未被使用时，可以设置为 *-1*，即成为 **虚拟 IO** 。对于一些简单的脉冲计数应用，电平信号或边沿信号是固定的（即不会发生改变），可将其设置为虚拟 IO，然后该信号会被连接到一个固定的高/低逻辑电平，这样就可以在通道分配时回收一个 GPIO，节省一个 GPIO 管脚资源。
- *pcnt_chan_config_t::virt_edge_io_level* 与*pcnt_chan_config_t::virt_level_io_level* 用于指定 **边沿**信号和 **电平**信号的虚拟 IO 电平，以保证这些控制信号处于确定状态。请注意，只有在*pcnt_chan_config_t::edge_gpio_num* 或*pcnt_chan_config_t::level_gpio_num* 设置为 *-1* 时，这两个参数才有效。
- *pcnt_chan_config_t::invert_edge_input* 与*pcnt_chan_config_t::invert_level_input* 用于确定信号在输入 PCNT 之前是否需要被翻转，信号翻转由 GPIO 矩阵（不是 PCNT 单元）执行。

- *pcnt_chan_config_t::io_loop_back* 仅用于调试，它可以使能 GPIO 的输入和输出路径。这样，就可以通过调用位于同一 GPIO 上的函数*gpio_set_level()* 来模拟脉冲信号。

调用函数*pcnt_new_channel()*，将*pcnt_chan_config_t* 作为输入值并调用*pcnt_new_unit()* 返回的 PCNT 单元句柄，可对 PCNT 通道进行分配和初始化。如果该函数正常运行，会返回一个 PCNT 通道句柄。如果没有可用的 PCNT 通道（PCNT 通道资源全部被占用），该函数会返回错误*ESP_ERR_NOT_FOUND*。可用的 PCNT 通道总数记录在*SOC_PCNT_CHANNELS_PER_UNIT*，以供参考。注意，为某个单元安装 PCNT 通道时，应确保该单元处于初始状态，否则函数*pcnt_new_channel()* 会返回错误*ESP_ERR_INVALID_STATE*。

如果不再需要之前创建的某个 PCNT 通道，建议通过调用*pcnt_del_channel()* 回收该通道，从而该通道可用于其他用途。

```
#define EXAMPLE_CHAN_GPIO_A 0
#define EXAMPLE_CHAN_GPIO_B 2

pcnt_chan_config_t chan_config = {
    .edge_gpio_num = EXAMPLE_CHAN_GPIO_A,
    .level_gpio_num = EXAMPLE_CHAN_GPIO_B,
};
pcnt_channel_handle_t pcnt_chan = NULL;
ESP_ERROR_CHECK(pcnt_new_channel(pcnt_unit, &chan_config, &pcnt_chan));
```

设置通道操作 当输入脉冲信号切换时，PCNT 通道会增加，减少或停止计数。边沿信号及电平信号可设置为不同的计数器操作。

- *pcnt_channel_set_edge_action()* 为输入到*pcnt_chan_config_t::edge_gpio_num* 的信号上升沿和下降沿设置操作，*pcnt_channel_edge_action_t* 中列出了支持的操作。
- *pcnt_channel_set_level_action()* 为输入到*pcnt_chan_config_t::level_gpio_num* 的信号高电平和低电平设置操作，*pcnt_channel_level_action_t* 中列出了支持的操作。使用*pcnt_new_channel()* 分配 PCNT 通道时，如果*pcnt_chan_config_t::level_gpio_num* 被设置为 *-1*，就无需对该函数进行设置了。

```
// decrease the counter on rising edge, increase the counter on falling edge
ESP_ERROR_CHECK(pcnt_channel_set_edge_action(pcnt_chan, PCNT_CHANNEL_EDGE_ACTION_
↪DECREASE, PCNT_CHANNEL_EDGE_ACTION_INCREASE));
// keep the counting mode when the control signal is high level, and reverse the␣
↪counting mode when the control signal is low level
ESP_ERROR_CHECK(pcnt_channel_set_level_action(pcnt_chan, PCNT_CHANNEL_LEVEL_ACTION_
↪KEEP, PCNT_CHANNEL_LEVEL_ACTION_INVERSE));
```

PCNT 观察点 PCNT 单元可被设置为观察几个特定的数值，这些被观察的数值被称为 **观察点**。观察点不能超过*pcnt_unit_config_t* 设置的范围，最小值和最大值分别为*pcnt_unit_config_t::low_limit* 和*pcnt_unit_config_t::high_limit*。当计数器到达任一观察点时，会触发一个观察事件，如果在*pcnt_unit_register_event_callbacks()* 注册过事件回调函数，该事件就会通过中断发送通知。关于如何注册事件回调函数，请参考注册事件回调函数。

观察点分别可以通过*pcnt_unit_add_watch_point()* 和*pcnt_unit_remove_watch_point()* 进行添加和删除。常用的观察点包括 **过零**、**最大/最小计数** 以及其他的阈值。可用的观察点是有限的，如果*pcnt_unit_add_watch_point()* 无法获得空闲硬件资源来存储观察点，会返回错误*ESP_ERR_NOT_FOUND*。不能多次添加同一个观察点，否则将返回错误*ESP_ERR_INVALID_STATE*。

建议通过*pcnt_unit_remove_watch_point()* 删除未使用的观察点来回收资源。

```
// add zero across watch point
ESP_ERROR_CHECK(pcnt_unit_add_watch_point(pcnt_unit, 0));
// add high limit watch point
ESP_ERROR_CHECK(pcnt_unit_add_watch_point(pcnt_unit, EXAMPLE_PCNT_HIGH_LIMIT));
```

注册事件回调函数 当 PCNT 单元的数值达到任一使能的观察点的数值时，会触发相应的事件并通过中断通知 CPU。如果要在事件触发时执行相关函数，可通过调用*pcnt_unit_register_event_callbacks()* 将函数挂载到中断服务程序 (ISR) 上。*pcnt_event_callbacks_t* 列出了所有支持的事件回调函数：

- *pcnt_event_callbacks_t::on_reach* 用于为观察点事件设置回调函数。由于该回调函数是在 ISR 的上下文中被调用的，必须确保该函数不会阻塞调用的任务，（例如，可确保只有以 `ISR` 为后缀的 FreeRTOS API 才能在函数中调用）。*pcnt_watch_cb_t* 中声明了该回调函数的原型。

可通过 `user_ctx` 将函数上下文保存到*pcnt_unit_register_event_callbacks()* 中，这些数据会直接传递给回调函数。

驱动程序会将特定事件的数据写入回调函数中，例如，观察点事件数据被声明为*pcnt_watch_event_data_t*：

- *pcnt_watch_event_data_t::watch_point_value* 用于保存触发该事件的观察点数值。
- *pcnt_watch_event_data_t::zero_cross_mode* 用于保存上一次 PCNT 单元的过零模式，*pcnt_unit_zero_cross_mode_t* 中列出了所有可能的过零模式。通常，不同的过零模式意味着不同的 **计数方向**和 **计数步长**。

注册回调函数会导致中断服务延迟安装，因此回调函数只能在 PCNT 单元被*pcnt_unit_enable()* 使能之前调用。否则，回调函数会返回错误*ESP_ERR_INVALID_STATE*。

```
static bool example_pcnt_on_reach(pcnt_unit_handle_t unit, const pcnt_watch_event_
↪data_t *edata, void *user_ctx)
{
    BaseType_t high_task_wakeup;
    QueueHandle_t queue = (QueueHandle_t)user_ctx;
    // send watch point to queue, from this interrupt callback
    xQueueSendFromISR(queue, &(edata->watch_point_value), &high_task_wakeup);
    // return whether a high priority task has been waken up by this function
    return (high_task_wakeup == pdTRUE);
}

pcnt_event_callbacks_t cbs = {
    .on_reach = example_pcnt_on_reach,
};
QueueHandle_t queue = xQueueCreate(10, sizeof(int));
ESP_ERROR_CHECK(pcnt_unit_register_event_callbacks(pcnt_unit, &cbs, queue));
```

设置毛刺滤波器 PCNT 单元的滤波器可滤除信号中的短时毛刺，*pcnt_glitch_filter_config_t* 中列出了毛刺滤波器的配置参数：

- *pcnt_glitch_filter_config_t::max_glitch_ns* 设置了最大的毛刺宽度，单位为纳秒。如果一个信号脉冲的宽度小于该数值，则该信号会被认定为噪声而不会触发计数器操作。

可通过调用*pcnt_unit_set_glitch_filter()* 来使能毛刺滤波器，并对上述参数进行配置。之后，还可通过调用*pcnt_unit_set_glitch_filter()* 来关闭毛刺滤波器，并将上述参数设置为 *NULL*。

调用该函数时，PCNT 单元应处于初始状态。否则，函数将返回错误*ESP_ERR_INVALID_STATE*。

备注： 毛刺滤波器的时钟信息来自 APB。为确保 PCNT 单元不会滤除脉冲信号，最大毛刺宽度应大于一个 APB_CLK 周期（如果 APB 的频率为 80 MHz，则最大毛刺宽度为 12.5 ns）。使能动态频率缩放 (DFS) 后，APB 的频率会发生变化，从而最大毛刺宽度也会发生变化，这会导致计数器无法正常工作。因此，驱动会为每个 PCNT 单元安装电源锁。关于 PCNT 驱动的电源管理的更多信息，请参考电源管理。

```
pcnt_glitch_filter_config_t filter_config = {
    .max_glitch_ns = 1000,
};
ESP_ERROR_CHECK(pcnt_unit_set_glitch_filter(pcnt_unit, &filter_config));
```

使用外部清零信号 PCNT 单元的可以接收来自 GPIO 的清零信号，*pcnt_clear_signal_config_t* 中列出了清零信号的配置参数：

- *pcnt_clear_signal_config_t::clear_signal_gpio_num* 用于指定 **清零** 信号对应的 GPIO 编号。默认有效电平为高，使能下拉输入。
- *pcnt_clear_signal_config_t::invert_clear_signal* 用于确定信号在输入 PCNT 之前是否需要被翻转，信号翻转由 GPIO 矩阵 (不是 PCNT 单元) 执行。驱动会使能上拉输入，以确保信号在未连接时保持高电平。
- *pcnt_clear_signal_config_t::io_loop_back* 仅用于调试，它可以使能 GPIO 的输入和输出路径。这样，就可以通过*gpio_set_level()* 函数来模拟外部输入的清零信号。

该输入信号的作用与调用*pcnt_unit_clear_count()* 函数相同，但它不受软件延迟的限制，更适用于需要低延迟的场合。请注意，该信号的翻转频率不能太高。

```
pcnt_clear_signal_config_t clear_signal_config = {
    .clear_signal_gpio_num = PCNT_CLEAR_SIGNAL_GPIO,
};
ESP_ERROR_CHECK(pcnt_unit_set_clear_signal(pcnt_unit, &clear_signal_config));
```

使能和禁用单元 在对 PCNT 单元进行 IO 控制之前，需要通过调用函数*pcnt_unit_enable()* 来使能该 PCNT 单元。该函数将完成以下操作：

- 将 PCNT 单元的驱动状态从 **初始**切换到 **使能**。
- 如果中断服务已经在*pcnt_unit_register_event_callbacks()* 延迟安装，使能中断服务。
- 如果电源管理锁已经安装，获取该电源管理锁。请参考电源管理 获取更多信息。

调用函数*pcnt_unit_disable()* 会进行相反的操作，即将 PCNT 单元的驱动状态切换回 **初始**状态，禁用中断服务并释放电源管理锁。

控制单元 IO 操作

启用/停用及清零 通过调用*pcnt_unit_start()* 可启用 PCNT 单元，根据不同脉冲信号进行递增或递减计数；通过调用*pcnt_unit_stop()* 可停用 PCNT 单元，当前的计数值会保留；通过调用*pcnt_unit_clear_count()* 可将计数器清零。

注意*pcnt_unit_start()* 和*pcnt_unit_stop()* 应该在 PCNT 单元被*pcnt_unit_enable()* 使能后调用，否则将返回错误*ESP_ERR_INVALID_STATE*。

获取计数器数值 调用*pcnt_unit_get_count()* 可随时获取当前计数器的数值。返回的计数值是一个 **带符号**的整型数，其符号反映了计数的方向。

```
int pulse_count = 0;
ESP_ERROR_CHECK(pcnt_unit_get_count(pcnt_unit, &pulse_count));
```

计数溢出补偿 PCNT 内部的硬件计数器会在计数达到高/低门限的时候自动清零。如果你想补偿该计数值的溢出损失，以期进一步拓宽计数器的实际位宽，你可以：

1. 在安装 PCNT 计数单元的时候使能*pcnt_unit_config_t::accum_count* 选项。
2. 将高/低计数门限设置为*PCNT 观察点*.
3. 现在，*pcnt_unit_get_count()* 函数返回的计数值就会包含硬件计数器当前的计数值，累加上计数器溢出造成的损失。

备注: *pcnt_unit_clear_count()* 会复位该软件累加器。

电源管理 当电源管理使能（即*CONFIG_PM_ENABLE* 开启）时，系统会在进入 Light-sleep 模式之前调整 APB 的频率，这可能导致 PCNT 毛刺滤波器将有效信号误认为噪声。

为了防止这种情况发生，驱动程序可以获取类型为*ESP_PM_APB_FREQ_MAX* 的电源管理锁，以确保 APB 频率保持不变。该锁在通过*pcnt_unit_enable()* 使能 PCNT 单元时获取，并在通过*pcnt_unit_disable()* 禁用单元时释放。

支持 IRAM 安全中断 当缓存由于写入/擦除 flash 等原因被禁用时，PCNT 中断会默认被延迟。这会导致报警中断无法及时执行，从而无法满足实时性应用的要求。

Konfig 选项*CONFIG_PCNT_ISR_IRAM_SAFE* 可以实现以下功能：

1. 即使缓存被禁用也可以使能中断服务
2. 将 ISR 使用的所有函数都放入 IRAM 中[2]
3. 将驱动对象放入 DRAM（防止驱动对象被意外映射到 PSRAM 中）

这样，在缓存被禁用时，中断也可运行，但是这也会增加 IRAM 的消耗。

另外一个 Konfig 选项*CONFIG_PCNT_CTRL_FUNC_IN_IRAM* 也可以把常用的 IO 控制函数放在 IRAM 中。这样，当缓存禁用时，这些函数仍然可以执行。这些 IO 控制函数如下所示：

- *pcnt_unit_start()*
- *pcnt_unit_stop()*
- *pcnt_unit_clear_count()*
- *pcnt_unit_get_count()*

支持线程安全 驱动保证工厂函数*pcnt_new_unit()* 与*pcnt_new_channel()* 是线程安全的，因此可以从 RTOS 任务中调用这些函数，而无需使用额外的电源管理锁。

以下函数可以在 ISR 上下文中运行，驱动可以防止这些函数在任务和 ISR 中同时被调用。

- *pcnt_unit_start()*
- *pcnt_unit_stop()*
- *pcnt_unit_clear_count()*
- *pcnt_unit_get_count()*

其他以*pcnt_unit_handle_t* 和*pcnt_channel_handle_t* 作为第一个参数的函数被视为线程不安全函数，在多任务场景下应避免调用这些函数。

支持的 Kconfig 选项

- *CONFIG_PCNT_CTRL_FUNC_IN_IRAM* 用于确定 PCNT 控制函数的位置（放在 IRAM 还是 flash 中），请参考*支持 IRAM 安全中断* 获取更多信息。
- *CONFIG_PCNT_ISR_IRAM_SAFE* 用于控制当缓存禁用时，默认的 ISR 句柄是否可以工作，请参考*支持 IRAM 安全中断* 获取更多信息。
- *CONFIG_PCNT_ENABLE_DEBUG_LOG* 用于使能调试日志输出，而这会增大固件二进制文件。

应用实例

- 对旋转编码器的正交信号进行解码的实例请参考：peripherals/pcnt/rotary_encoder。

API 参考

Header File

- components/esp_driver_pcnt/include/driver/pulse_cnt.h
- This header file can be included with:

[2] *pcnt_event_callbacks_t::on_reach* 回调函数和其调用的函数也应该放在 IRAM 中。

```
#include "driver/pulse_cnt.h"
```

- This header file is a part of the API provided by the `esp_driver_pcnt` component. To declare that your component depends on `esp_driver_pcnt`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_pcnt
```

or

```
PRIV_REQUIRES esp_driver_pcnt
```

Functions

esp_err_t **pcnt_new_unit**(const *pcnt_unit_config_t* *config, *pcnt_unit_handle_t* *ret_unit)

Create a new PCNT unit, and return the handle.

备注: The newly created PCNT unit is put in the init state.

参数

- **config** -- **[in]** PCNT unit configuration
- **ret_unit** -- **[out]** Returned PCNT unit handle

返回

- ESP_OK: Create PCNT unit successfully
- ESP_ERR_INVALID_ARG: Create PCNT unit failed because of invalid argument (e.g. high/low limit value out of the range)
- ESP_ERR_NO_MEM: Create PCNT unit failed because out of memory
- ESP_ERR_NOT_FOUND: Create PCNT unit failed because all PCNT units are used up and no more free one
- ESP_FAIL: Create PCNT unit failed because of other error

esp_err_t **pcnt_del_unit**(*pcnt_unit_handle_t* unit)

Delete the PCNT unit handle.

备注: A PCNT unit can't be in the enable state when this function is invoked. See also `pcnt_unit_disable()` for how to disable a unit.

参数 **unit** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`

返回

- ESP_OK: Delete the PCNT unit successfully
- ESP_ERR_INVALID_ARG: Delete the PCNT unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete the PCNT unit failed because the unit is not in init state or some PCNT channel is still in working
- ESP_FAIL: Delete the PCNT unit failed because of other error

esp_err_t **pcnt_unit_set_glitch_filter**(*pcnt_unit_handle_t* unit, const *pcnt_glitch_filter_config_t* *config)

Set glitch filter for PCNT unit.

备注: The glitch filter module is clocked from APB, and APB frequency can be changed during DFS, which in return make the filter out of action. So this function will lazy-install a PM lock internally when the power management is enabled. With this lock, the APB frequency won't be changed. The PM lock can be uninstalled in `pcnt_del_unit()`.

备注: This function should be called when the PCNT unit is in the init state (i.e. before calling `pcnt_unit_enable()`)

参数

- **`unit`** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`
- **`config`** -- **[in]** PCNT filter configuration, set config to NULL means disabling the filter function

返回

- ESP_OK: Set glitch filter successfully
- ESP_ERR_INVALID_ARG: Set glitch filter failed because of invalid argument (e.g. glitch width is too big)
- ESP_ERR_INVALID_STATE: Set glitch filter failed because the unit is not in the init state
- ESP_FAIL: Set glitch filter failed because of other error

esp_err_t **`pcnt_unit_set_clear_signal`**(*pcnt_unit_handle_t* unit, const *pcnt_clear_signal_config_t* *config)

Set clear signal for PCNT unit.

备注: The function of clear signal is the same as `pcnt_unit_clear_count()`. High-level Active

参数

- **`unit`** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`
- **`config`** -- **[in]** PCNT clear signal configuration, set config to NULL means disabling the clear signal

返回

- ESP_OK: Set clear signal successfully
- ESP_ERR_INVALID_ARG: Set clear signal failed because of invalid argument
- ESP_ERR_INVALID_STATE: Set clear signal failed because set clear signal repeatedly or disable clear signal before set it
- ESP_FAIL: Set clear signal failed because of other error

esp_err_t **`pcnt_unit_enable`**(*pcnt_unit_handle_t* unit)

Enable the PCNT unit.

备注: This function will transit the unit state from init to enable.

备 注: This function will enable the interrupt service, if it's lazy installed in `pcnt_unit_register_event_callbacks()`.

备 注: This function will acquire the PM lock if it's lazy installed in `pcnt_unit_set_glitch_filter()`.

备注: Enable a PCNT unit doesn't mean to start it. See also `pcnt_unit_start()` for how to start the PCNT counter.

参数 **`unit`** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`

返回

- ESP_OK: Enable PCNT unit successfully
- ESP_ERR_INVALID_ARG: Enable PCNT unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable PCNT unit failed because the unit is already enabled
- ESP_FAIL: Enable PCNT unit failed because of other error

esp_err_t **pcnt_unit_disable**(*pcnt_unit_handle_t* unit)

Disable the PCNT unit.

备注: This function will do the opposite work to the `pcnt_unit_enable()`

备注: Disable a PCNT unit doesn't mean to stop it. See also `pcnt_unit_stop()` for how to stop the PCNT counter.

参数 **unit** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`

返回

- ESP_OK: Disable PCNT unit successfully
- ESP_ERR_INVALID_ARG: Disable PCNT unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable PCNT unit failed because the unit is not enabled yet
- ESP_FAIL: Disable PCNT unit failed because of other error

esp_err_t **pcnt_unit_start**(*pcnt_unit_handle_t* unit)

Start the PCNT unit, the counter will start to count according to the edge and/or level input signals.

备注: This function should be called when the unit is in the enable state (i.e. after calling `pcnt_unit_enable()`)

备注: This function is allowed to run within ISR context

备注: This function will be placed into IRAM if `CONFIG_PCNT_CTRL_FUNC_IN_IRAM` is on, so that it's allowed to be executed when Cache is disabled

参数 **unit** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`

返回

- ESP_OK: Start PCNT unit successfully
- ESP_ERR_INVALID_ARG: Start PCNT unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Start PCNT unit failed because the unit is not enabled yet
- ESP_FAIL: Start PCNT unit failed because of other error

esp_err_t **pcnt_unit_stop**(*pcnt_unit_handle_t* unit)

Stop PCNT from counting.

备注: This function should be called when the unit is in the enable state (i.e. after calling `pcnt_unit_enable()`)

备注: The stop operation won't clear the counter. Also see `pcnt_unit_clear_count()` for how to clear pulse count value.

备注: This function is allowed to run within ISR context

备注: This function will be placed into IRAM if `CONFIG_PCNT_CTRL_FUNC_IN_IRAM`, so that it is allowed to be executed when Cache is disabled

参数 **`unit`** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`

返回

- ESP_OK: Stop PCNT unit successfully
- ESP_ERR_INVALID_ARG: Stop PCNT unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Stop PCNT unit failed because the unit is not enabled yet
- ESP_FAIL: Stop PCNT unit failed because of other error

esp_err_t **`pcnt_unit_clear_count`**(*pcnt_unit_handle_t* unit)

Clear PCNT pulse count value to zero.

备注: It's recommended to call this function after adding a watch point by `pcnt_unit_add_watch_point()`, so that the newly added watch point is effective immediately.

备注: This function is allowed to run within ISR context

备注: This function will be placed into IRAM if `CONFIG_PCNT_CTRL_FUNC_IN_IRAM`, so that it's allowed to be executed when Cache is disabled

参数 **`unit`** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`

返回

- ESP_OK: Clear PCNT pulse count successfully
- ESP_ERR_INVALID_ARG: Clear PCNT pulse count failed because of invalid argument
- ESP_FAIL: Clear PCNT pulse count failed because of other error

esp_err_t **`pcnt_unit_get_count`**(*pcnt_unit_handle_t* unit, int *value)

Get PCNT count value.

备注: This function is allowed to run within ISR context

备注: This function will be placed into IRAM if `CONFIG_PCNT_CTRL_FUNC_IN_IRAM`, so that it's allowed to be executed when Cache is disabled

参数

- **`unit`** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`
- **`value`** -- **[out]** Returned count value

返回

- ESP_OK: Get PCNT pulse count successfully

- ESP_ERR_INVALID_ARG: Get PCNT pulse count failed because of invalid argument
- ESP_FAIL: Get PCNT pulse count failed because of other error

esp_err_t **pcnt_unit_register_event_callbacks**(*pcnt_unit_handle_t* unit, const *pcnt_event_callbacks_t* *cbs, void *user_data)

Set event callbacks for PCNT unit.

备注: User registered callbacks are expected to be runnable within ISR context

备注: The first call to this function needs to be before the call to `pcnt_unit_enable`

备注: User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

参数

- **unit** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_ERR_INVALID_STATE: Set event callbacks failed because the unit is not in init state
- ESP_FAIL: Set event callbacks failed because of other error

esp_err_t **pcnt_unit_add_watch_point**(*pcnt_unit_handle_t* unit, int watch_point)

Add a watch point for PCNT unit, PCNT will generate an event when the counter value reaches the watch point value.

参数

- **unit** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`
- **watch_point** -- **[in]** Value to be watched

返回

- ESP_OK: Add watch point successfully
- ESP_ERR_INVALID_ARG: Add watch point failed because of invalid argument (e.g. the value to be watched is out of the limitation set in *pcnt_unit_config_t*)
- ESP_ERR_INVALID_STATE: Add watch point failed because the same watch point has already been added
- ESP_ERR_NOT_FOUND: Add watch point failed because no more hardware watch point can be configured
- ESP_FAIL: Add watch point failed because of other error

esp_err_t **pcnt_unit_remove_watch_point**(*pcnt_unit_handle_t* unit, int watch_point)

Remove a watch point for PCNT unit.

参数

- **unit** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`
- **watch_point** -- **[in]** Watch point value

返回

- ESP_OK: Remove watch point successfully
- ESP_ERR_INVALID_ARG: Remove watch point failed because of invalid argument
- ESP_ERR_INVALID_STATE: Remove watch point failed because the watch point was not added by `pcnt_unit_add_watch_point()` yet
- ESP_FAIL: Remove watch point failed because of other error

esp_err_t **pcnt_new_channel** (*pcnt_unit_handle_t* unit, const *pcnt_chan_config_t* *config, *pcnt_channel_handle_t* *ret_chan)

Create PCNT channel for specific unit, each PCNT has several channels associated with it.

备 注: This function should be called when the unit is in init state (i.e. before calling `pcnt_unit_enable()`)

参数

- **unit** -- **[in]** PCNT unit handle created by `pcnt_new_unit()`
- **config** -- **[in]** PCNT channel configuration
- **ret_chan** -- **[out]** Returned channel handle

返回

- ESP_OK: Create PCNT channel successfully
- ESP_ERR_INVALID_ARG: Create PCNT channel failed because of invalid argument
- ESP_ERR_NO_MEM: Create PCNT channel failed because of insufficient memory
- ESP_ERR_NOT_FOUND: Create PCNT channel failed because all PCNT channels are used up and no more free one
- ESP_ERR_INVALID_STATE: Create PCNT channel failed because the unit is not in the init state
- ESP_FAIL: Create PCNT channel failed because of other error

esp_err_t **pcnt_del_channel** (*pcnt_channel_handle_t* chan)

Delete the PCNT channel.

参数 **chan** -- **[in]** PCNT channel handle created by `pcnt_new_channel()`

返回

- ESP_OK: Delete the PCNT channel successfully
- ESP_ERR_INVALID_ARG: Delete the PCNT channel failed because of invalid argument
- ESP_FAIL: Delete the PCNT channel failed because of other error

esp_err_t **pcnt_channel_set_edge_action** (*pcnt_channel_handle_t* chan, *pcnt_channel_edge_action_t* pos_act, *pcnt_channel_edge_action_t* neg_act)

Set channel actions when edge signal changes (e.g. falling or rising edge occurred). The edge signal is input from the `edge_gpio_num` configured in *pcnt_chan_config_t*. We use these actions to control when and how to change the counter value.

参数

- **chan** -- **[in]** PCNT channel handle created by `pcnt_new_channel()`
- **pos_act** -- **[in]** Action on posedge signal
- **neg_act** -- **[in]** Action on negedge signal

返回

- ESP_OK: Set edge action for PCNT channel successfully
- ESP_ERR_INVALID_ARG: Set edge action for PCNT channel failed because of invalid argument
- ESP_FAIL: Set edge action for PCNT channel failed because of other error

esp_err_t **pcnt_channel_set_level_action** (*pcnt_channel_handle_t* chan, *pcnt_channel_level_action_t* high_act, *pcnt_channel_level_action_t* low_act)

Set channel actions when level signal changes (e.g. signal level goes from high to low). The level signal is input from the `level_gpio_num` configured in *pcnt_chan_config_t*. We use these actions to control when and how to change the counting mode.

参数

- **chan** -- **[in]** PCNT channel handle created by `pcnt_new_channel()`
- **high_act** -- **[in]** Action on high level signal
- **low_act** -- **[in]** Action on low level signal

返回

- ESP_OK: Set level action for PCNT channel successfully
- ESP_ERR_INVALID_ARG: Set level action for PCNT channel failed because of invalid argument
- ESP_FAIL: Set level action for PCNT channel failed because of other error

Structures

struct **pcnt_watch_event_data_t**

PCNT watch event data.

Public Members

int **watch_point_value**

Watch point value that triggered the event

pcnt_unit_zero_cross_mode_t **zero_cross_mode**

Zero cross mode

struct **pcnt_event_callbacks_t**

Group of supported PCNT callbacks.

备注： The callbacks are all running under ISR environment

备注： When CONFIG_PCNT_ISR_IRAM_SAFE is enabled, the callback itself and functions callbed by it should be placed in IRAM.

Public Members

pcnt_watch_cb_t **on_reach**

Called when PCNT unit counter reaches any watch point

struct **pcnt_unit_config_t**

PCNT unit configuration.

Public Members

int **low_limit**

Low limitation of the count unit, should be lower than 0

int **high_limit**

High limitation of the count unit, should be higher than 0

int **intr_priority**

PCNT interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **accum_count**

Whether to accumulate the count value when overflows at the high/low limit

struct *pcnt_unit_config_t*::[anonymous] **flags**

Extra flags

struct **pcnt_chan_config_t**

PCNT channel configuration.

Public Members

int **edge_gpio_num**

GPIO number used by the edge signal, input mode with pull up enabled. Set to -1 if unused

int **level_gpio_num**

GPIO number used by the level signal, input mode with pull up enabled. Set to -1 if unused

uint32_t **invert_edge_input**

Invert the input edge signal

uint32_t **invert_level_input**

Invert the input level signal

uint32_t **virt_edge_io_level**

Virtual edge IO level, 0: low, 1: high. Only valid when edge_gpio_num is set to -1

uint32_t **virt_level_io_level**

Virtual level IO level, 0: low, 1: high. Only valid when level_gpio_num is set to -1

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

struct *pcnt_chan_config_t*::[anonymous] **flags**

Channel config flags

struct **pcnt_glitch_filter_config_t**

PCNT glitch filter configuration.

Public Members

uint32_t **max_glitch_ns**

Pulse width smaller than this threshold will be treated as glitch and ignored, in the unit of ns

struct **pcnt_clear_signal_config_t**

PCNT clear signal configuration.

Public Members

int **clear_signal_gpio_num**

GPIO number used by the clear signal, the default active level is high, input mode with pull down enabled

uint32_t **invert_clear_signal**

Invert the clear input signal and set input mode with pull up

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

struct *pcnt_clear_signal_config_t*::[anonymous] **flags**

clear signal config flags

Type Definitions

typedef struct pcnt_unit_t ***pcnt_unit_handle_t**

Type of PCNT unit handle.

typedef struct pcnt_chan_t ***pcnt_channel_handle_t**

Type of PCNT channel handle.

typedef bool (***pcnt_watch_cb_t**)(*pcnt_unit_handle_t* unit, const *pcnt_watch_event_data_t* *edata, void *user_ctx)

PCNT watch event callback prototype.

备注： The callback function is invoked from an ISR context, so it should meet the restrictions of not calling any blocking APIs when implementing the callback. e.g. must use ISR version of FreeRTOS APIs.

Param unit **[in]** PCNT unit handle
Param edata **[in]** PCNT event data, fed by the driver
Param user_ctx **[in]** User data, passed from `pcnt_unit_register_event_callbacks()`
Return Whether a high priority task has been woken up by this function

Header File

- components/hal/include/hal/pcnt_types.h
- This header file can be included with:

```
#include "hal/pcnt_types.h"
```

Enumerations

enum **pcnt_channel_level_action_t**

PCNT channel action on control level.

Values:

enumerator **PCNT_CHANNEL_LEVEL_ACTION_KEEP**

Keep current count mode

enumerator **PCNT_CHANNEL_LEVEL_ACTION_INVERSE**

Invert current count mode (increase -> decrease, decrease -> increase)

enumerator **PCNT_CHANNEL_LEVEL_ACTION_HOLD**

Hold current count value

enum **pcnt_channel_edge_action_t**

PCNT channel action on signal edge.

Values:

enumerator **PCNT_CHANNEL_EDGE_ACTION_HOLD**

Hold current count value

enumerator **PCNT_CHANNEL_EDGE_ACTION_INCREASE**

Increase count value

enumerator **PCNT_CHANNEL_EDGE_ACTION_DECREASE**

Decrease count value

enum **pcnt_unit_zero_cross_mode_t**

PCNT unit zero cross mode.

Values:

enumerator **PCNT_UNIT_ZERO_CROSS_POS_ZERO**

start from positive value, end to zero, i.e. +N->0

enumerator **PCNT_UNIT_ZERO_CROSS_NEG_ZERO**

start from negative value, end to zero, i.e. -N->0

enumerator **PCNT_UNIT_ZERO_CROSS_NEG_POS**

start from negative value, end to positive value, i.e. -N->+M

enumerator **PCNT_UNIT_ZERO_CROSS_POS_NEG**

start from positive value, end to negative value, i.e. +N->-M

2.5.22 Pixel-Processing Accelerator (PPA)

Introduction

ESP32-P4 includes a pixel-processing accelerator (PPA) module, to realize hardware-level acceleration of image algorithms, such as image rotation, scaling, mirroring, and blending.

Terminology

The terms used in relation to the PPA driver are given in the table and the diagram below.

Term	Definition
Picture (pic)	A complete image stored in the system memory.
Block	A portion cropped from a picture at a certain size, with the maximum size equivalent to the entire picture.
Pixel	The unit to be used in the PPA context.
PPA Operation	Types of image algorithm accelerations, includes scale-rotate-mirror (SRM), blend, and fill.
PPA Client	Who wants to do the PPA operations. Typically, every PPA client is hold by a specific task.
PPA Transaction	One request from a PPA client to do a PPA operation is one PPA transaction.

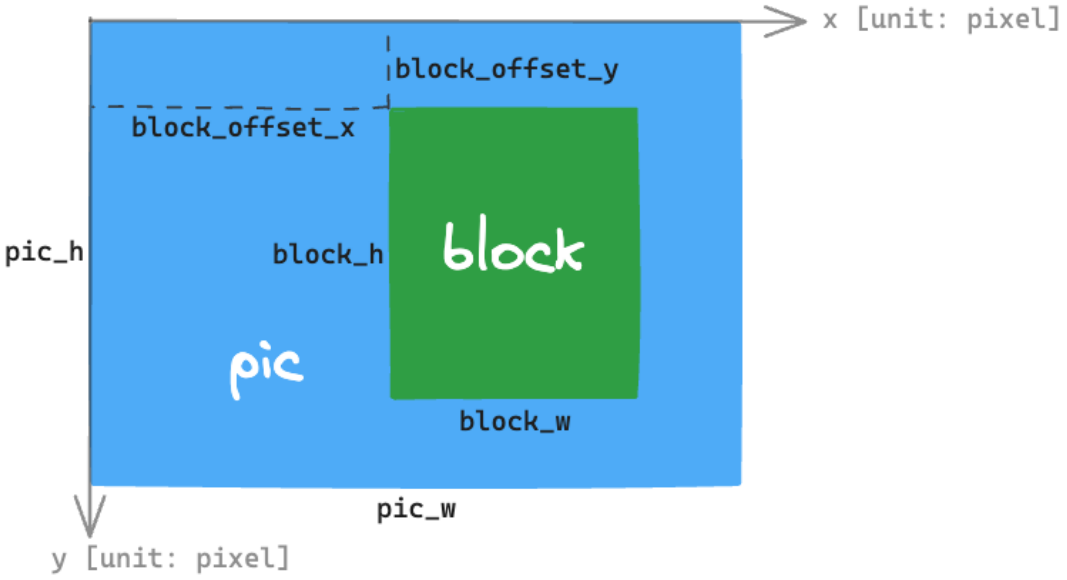


图 20: PPA picture/block terminology

Functional Overview

The following sections detail the design of the PPA driver:

- *Register PPA Client* - Covers how to register a PPA client to perform any PPA operations.
- *Register PPA Event Callbacks* - Covers how to hook user specific code to PPA driver event callback function.
- *Perform PPA Operations* - Covers how to perform a PPA operation.
- *Thread Safety* - Covers the usage of the PPA operation APIs in thread safety aspect.
- *Performance Overview* - Covers the performance of PPA operations.

Register PPA Client Requests to perform PPA operations are made by PPA clients. Therefore, PPA clients need to be registered first before doing any PPA operations. Call `ppa_register_client()` function to register a new client. `ppa_client_config_t` structure is used to specific the properties of the client.

- `ppa_client_config_t::oper_type` - Each PPA operation type corresponds to one PPA client type, a registered PPA client can only request one specific type of PPA operations.
- `ppa_client_config_t::max_pending_trans_num` - Decides the maximum number of pending PPA transactions the client can hold.

It is recommended that every task to register its own PPA clients. For example, an application contains two tasks: Task A requires both the PPA SRM and the PPA fill functionalities, so one PPA SRM client and one PPA fill client should be registered in Task A; While Task B also requires the PPA SRM functionality, then another PPA SRM client should be registered in Task B.

If the task no longer needs to do PPA operations, the corresponding PPA clients can be deregistered with *ppa_unregister_client()* function.

Register PPA Event Callbacks When an event occurs (e.g., a PPA transaction is completed), the CPU is notified of this event via an interrupt. If some specific functions need to be called when a particular event occurs, a callback can be registered for that event by calling *ppa_client_register_event_callbacks()*. This can be specifically useful when PPA_TRANS_MODE_NON_BLOCKING mode is selected to perform the PPA operations. It is worth noticing that the event callbacks are bound to PPA clients, but the user context is provided per transaction in the call to the PPA operation APIs. This allows the maximum flexibility in utilizing the event callbacks.

The registered callback functions are called in the interrupt context, therefore, the callback functions should follow common ISR (Interrupt Service Routine) rules.

Perform PPA Operations Once the PPA client is registered, a PPA operation can be requested with the returned *ppa_client_handle_t*.

PPA operations includes:

Scale, Rotate, Mirror (SRM) Call *ppa_do_scale_rotate_mirror()* to apply one or more of the scaling, rotation, mirroring operations to the target block inside a picture.

Some notes to avoid confusion in configuring *ppa_srm_oper_config_t*:

- *ppa_in_pic_blk_config_t::buffer* and *ppa_out_pic_blk_config_t::buffer* have to be the pointers to different picture buffers for a SRM operation.
- The precision of *ppa_srm_oper_config_t::scale_x* and *ppa_srm_oper_config_t::scale_y* will be truncated to a step size of 1/16.
- Output block's width/height is totally determined by the input block's width/height, scaling factor, and rotation angle, so output block's width/height does not need to be configured. However, please make sure the output block can fit at the offset location in the output picture.
- If the color mode of the input or output picture is PPA_SRM_COLOR_MODE_YUV420, then its pic_w, pic_h, block_w, block_h, block_offset_x, block_offset_y fields must be even.

Blend Call *ppa_do_blend()* to blend the two target blocks of two so-called foreground (FG) and background (BG) pictures.

Blend follows the normal Alpha Blending formula:

$A_{out} = A_b + A_f - A_b \times A_f$

$C_{out} = (C_b \times A_b \times (1 - A_f) + C_f \times A_f)/(A_b + A_f - A_b \times A_f)$

where A_b is the Alpha channel of the background layer, A_f is the Alpha channel of the foreground layer, C_b corresponds to the R, G, B components of the background layer, and C_f corresponds to the R, G, B components of the foreground layer.

Note that this formula is not symmetric to FG and BG. When $A_f = 1$, it calculates $C_{out} = C_f$, $A_{out} = 1$, which means if the color mode of the FG picture is PPA_BLEND_COLOR_MODE_RGB565 or PPA_BLEND_COLOR_MODE_RGB888, since a Alpha value of 255 will be filled by the PPA hardware (i.e. $A_f = 1$), the blended result will be identical to the FG block.

If *ppa_blend_oper_config_t::bg_ck_en* or *ppa_blend_oper_config_t::fg_ck_en* is set to true, the pixels fall into the color-key (also known as Chroma-key) range does not follow Alpha Blending process. Please check **ESP32-P4 Technical Reference Manual > Pixel-Processing Accelerator (PPA) > Functional Description > Layer Blending (BLEND)** [PDF] for the detailed rules.

Similarly, some notes to avoid confusion in configuring *ppa_blend_oper_config_t*:

- *ppa_out_pic_blk_config_t::buffer* can be the same pointer to one of the input's *ppa_in_pic_blk_config_t::buffer* for a blend operation.
- The blocks' width/height of FG and BG should be identical, and are the width/height values for the output block.
- If the color mode of the input picture is `PPA_BLEND_COLOR_MODE_A4`, then its `block_w` and `block_offset_x` fields must be even.

Fill Call *ppa_do_fill()* to fill a target block inside a picture.

ppa_trans_mode_t is a field configurable to all the PPA operation APIs. It decides whether you want the call to the PPA operation API to block until the transaction finishes or to return immediately after the transaction is pushed to the internal queue.

Thread Safety The PPA driver has guaranteed the thread safety of calling the PPA operation APIs in all following situations:

- Among clients of different types in one task
- Among clients of same type in different tasks
- Among clients of different types in different tasks

Performance Overview The PPA operations are acted on the target block of an input picture. Therefore, the time it takes to complete a PPA transaction is proportional to the amount of the data in the block. The size of the entire picture has no influence on the performance. More importantly, the PPA performance highly relies on the PSRAM bandwidth if the pictures are located in the PSRAM section. When there are quite a few peripherals reading and writing to the PSRAM at the same time, the performance of PPA operation will be greatly reduced.

API Reference

Header File

- components/esp_driver_ppa/include/driver/ppa.h
- This header file can be included with:

```
#include "driver/ppa.h"
```

- This header file is a part of the API provided by the `esp_driver_ppa` component. To declare that your component depends on `esp_driver_ppa`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_ppa
```

or

```
PRIV_REQUIRES esp_driver_ppa
```

Functions

esp_err_t **ppa_register_client**(const *ppa_client_config_t* *config, *ppa_client_handle_t* *ret_client)

Register a PPA client to do a specific PPA operation.

参数

- **config** -- **[in]** Pointer to a collection of configurations for the client
- **ret_client** -- **[out]** Returned client handle

返回

- ESP_OK: Register the PPA client successfully
- ESP_ERR_INVALID_ARG: Register the PPA client failed because of invalid argument
- ESP_ERR_NO_MEM: Register the PPA client failed because out of memory
- ESP_FAIL: Register the PPA client failed because of other error

esp_err_t **ppa_unregister_client** (*ppa_client_handle_t* ppa_client)

Unregister a PPA client.

备注： This will also free the resources occupied by the client

参数 **ppa_client** -- **[in]** PPA client handle, allocated by `ppa_register_client`

返回

- ESP_OK: Unregister the PPA client successfully
- ESP_ERR_INVALID_ARG: Unregister the PPA client failed because of invalid argument
- ESP_ERR_INVALID_STATE: Unregister the PPA client failed because there are unfinished transactions

esp_err_t **ppa_client_register_event_callbacks** (*ppa_client_handle_t* ppa_client, const *ppa_event_callbacks_t* *cbs)

Register event callbacks for a PPA client.

备注： Any user private data that wants to be passed directly to callback's user_data is provided per PPA transaction. Please check the `user_data` field in `ppa_xxx_oper_config_t` structure.

参数

- **ppa_client** -- **[in]** PPA client handle
- **cbs** -- **[in]** Structure with all PPA callbacks

返回

- ESP_OK: Register event callbacks for the PPA client successfully
- ESP_ERR_INVALID_ARG: Register event callbacks for the PPA client failed because of invalid argument

esp_err_t **ppa_do_scale_rotate_mirror** (*ppa_client_handle_t* ppa_client, const *ppa_srm_oper_config_t* *config)

Perform a scaling-rotating-mirroring (SRM) operation to a picture.

参数

- **ppa_client** -- **[in]** PPA client handle that has been registered to do SRM operations
- **config** -- **[in]** Pointer to a collection of configurations for the SRM operation transaction, *ppa_srm_oper_config_t*

返回

- ESP_OK: Perform a SRM operation successfully
- ESP_ERR_INVALID_ARG: Perform a SRM operation failed because of invalid argument
- ESP_FAIL: Perform a SRM operation failed because the client's pending transactions has reached its maximum capacity

esp_err_t **ppa_do_blend** (*ppa_client_handle_t* ppa_client, const *ppa_blend_oper_config_t* *config)

Perform a blending operation to a picture.

参数

- **ppa_client** -- **[in]** PPA client handle that has been registered to do blend operations
- **config** -- **[in]** Pointer to a collection of configurations for the blend operation transaction, *ppa_blend_oper_config_t*

返回

- ESP_OK: Perform a blend operation successfully
- ESP_ERR_INVALID_ARG: Perform a blend operation failed because of invalid argument
- ESP_FAIL: Perform a blend operation failed because the client's pending transactions has reached its maximum capacity

esp_err_t **ppa_do_fill** (*ppa_client_handle_t* ppa_client, const *ppa_fill_oper_config_t* *config)

Perform a filling operation to a picture.

参数

- **ppa_client** -- **[in]** PPA client handle that has been registered to do fill operations
- **config** -- **[in]** Pointer to a collection of configurations for the fill operation transaction, *ppa_fill_oper_config_t*

返回

- ESP_OK: Perform a fill operation successfully
- ESP_ERR_INVALID_ARG: Perform a fill operation failed because of invalid argument
- ESP_FAIL: Perform a fill operation failed because the client's pending transactions has reached its maximum capacity

Structures

struct **ppa_client_config_t**

A collection of configuration items that used for registering a PPA client.

Public Members

ppa_operation_t **oper_type**

The desired PPA operation for the client

uint32_t **max_pending_trans_num**

The maximum number of pending transactions for the client. By default, it will be 1, which is sufficient if all transactions are performed with `PPA_TRANS_MODE_BLOCKING`

ppa_data_burst_length_t **data_burst_length**

The desired data burst length for all the transactions of the client. Use a small burst length will decrease PPA performance, but can save burst bandwidth for other peripheral usages. By default, it will be at the maximum burst length, `PPA_DATA_BURST_LENGTH_128`

struct **ppa_event_data_t**

Type of PPA event data.

struct **ppa_event_callbacks_t**

Group of supported PPA callbacks.

Public Members

ppa_event_callback_t **on_trans_done**

Invoked when a PPA transaction finishes

struct **ppa_in_pic_blk_config_t**

A collection of configuration items for an input picture and the target block inside the picture.

Public Members

const void ***buffer**

Pointer to the input picture buffer

uint32_t **pic_w**

Input picture width (unit: pixel)

uint32_t **pic_h**

Input picture height (unit: pixel)

uint32_t **block_w**

Target block width (unit: pixel)

uint32_t **block_h**

Target block height (unit: pixel)

uint32_t **block_offset_x**

Target block offset in x direction in the picture (unit: pixel)

uint32_t **block_offset_y**

Target block offset in y direction in the picture (unit: pixel)

ppa_srm_color_mode_t **srm_cm**

Color mode of the picture in a PPA SRM operation. Supported color mode in `ppa_srm_color_mode_t`

ppa_blend_color_mode_t **blend_cm**

Color mode of the picture in a PPA blend operation. Supported color mode in `ppa_blend_color_mode_t`

ppa_fill_color_mode_t **fill_cm**

Color mode of the picture in a PPA fill operation. Supported color mode in `ppa_fill_color_mode_t`

ppa_color_range_t **yuv_range**

When the color mode is any YUV color space, this field is to describe its color range

ppa_color_conv_std_rgb_yuv_t **yuv_std**

When the color mode is any YUV color space, this field is to describe its YUV<->RGB conversion standard

struct **ppa_out_pic_blk_config_t**

A collection of configuration items for an output picture and the target block inside the picture.

Public Members

void ***buffer**

Pointer to the output picture buffer (requires alignment: internal memory needs align to L1 cache line size, external memory needs align to L1 and L2 cache line size)

uint32_t **buffer_size**

Size of the output picture buffer (requires alignment: internal memory needs align to L1 cache line size, external memory needs align to L1 and L2 cache line size)

uint32_t **pic_w**

Output picture width (unit: pixel)

uint32_t **pic_h**

Output picture height (unit: pixel)

uint32_t **block_offset_x**

Target block offset in x direction in the picture (unit: pixel)

uint32_t **block_offset_y**

Target block offset in y direction in the picture (unit: pixel)

ppa_srm_color_mode_t **srm_cm**

Color mode of the picture in a PPA SRM operation. Supported color mode in `ppa_srm_color_mode_t`

ppa_blend_color_mode_t **blend_cm**

Color mode of the picture in a PPA blend operation. Supported color mode in `ppa_blend_color_mode_t`

ppa_fill_color_mode_t **fill_cm**

Color mode of the picture in a PPA fill operation. Supported color mode in `ppa_fill_color_mode_t`

ppa_color_range_t **yuv_range**

When the color mode is any YUV color space, this field is to describe its color range

ppa_color_conv_std_rgb_yuv_t **yuv_std**

When the color mode is any YUV color space, this field is to describe its YUV<->RGB conversion standard

struct **ppa_srm_oper_config_t**

A collection of configuration items to do a PPA SRM operation transaction.

Public Members

ppa_in_pic_blk_config_t **in**

Information of the input picture and the target block

ppa_out_pic_blk_config_t **out**

Information of the output picture and the target block

ppa_srm_rotation_angle_t **rotation_angle**

Rotation (counter-clockwise) to the target block, select from `ppa_srm_rotation_angle_t`

float **scale_x**

Scaling factor to the target block in the x direction

float **scale_y**

Scaling factor to the target block in the y direction

bool **mirror_x**

Whether to mirror the target block in the x direction

bool **mirror_y**

Whether to mirror the target block in the y direction

bool **rgb_swap**

Whether to swap the input data in RGB (e.g. ARGB becomes BGRA, RGB becomes BGR)

bool **byte_swap**

Whether to swap the input data in byte. Only available feature if input picture color mode is ARGB8888 or RGB565

ppa_alpha_update_mode_t **alpha_update_mode**

Select whether the alpha channel of the input picture needs update

uint32_t **alpha_fix_val**

Range: [0, 255] When PPA_ALPHA_FIX_VALUE mode is selected, alpha_fix_val is the new alpha value to replace the input alpha value (output_alpha = alpha_fix_val)

float **alpha_scale_ratio**

Range: (0, 1) When PPA_ALPHA_SCALE mode is selected, alpha_scale_ratio is the multiplier to the input alpha value (output_alpha = alpha_scale_ratio * input_alpha) Ratio resolution is 1/256

ppa_trans_mode_t **mode**

Determines whether to block inside the operation functions, see `ppa_trans_mode_t`

void ***user_data**

User registered data to be passed into `done_cb` callback function

struct **ppa_blend_oper_config_t**

A collection of configuration items to do a PPA blend operation transaction.

Public Members

ppa_in_pic_blk_config_t **in_bg**

Information of the input background picture and the target block

ppa_in_pic_blk_config_t **in_fg**

Information of the input foreground picture and the target block

ppa_out_pic_blk_config_t **out**

Information of the output picture and the target block

bool **bg_rgb_swap**

Whether to swap the background input data in RGB (e.g. ARGB becomes BGRA, RGB becomes BGR)

bool **bg_byte_swap**

Whether to swap the background input data in byte. Only available feature if input BG picture color mode is ARGB8888 or RGB565

ppa_alpha_update_mode_t **bg_alpha_update_mode**

Select whether the alpha channel of the input background picture needs update

uint32_t **bg_alpha_fix_val**

Range: [0, 255] When PPA_ALPHA_FIX_VALUE mode is selected, alpha_fix_val is the new alpha value to replace the input alpha value (output_alpha = alpha_fix_val)

float **bg_alpha_scale_ratio**

Range: (0, 1) When PPA_ALPHA_SCALE mode is selected, alpha_scale_ratio is the multiplier to the input alpha value (output_alpha = alpha_scale_ratio * input_alpha) Ratio resolution is 1/256

bool **fg_rgb_swap**

Whether to swap the foreground input data in RGB (e.g. ARGB becomes BGRA, RGB becomes BGR)

bool **fg_byte_swap**

Whether to swap the foreground input data in byte. Only available feature if input FG picture color mode is ARGB8888 or RGB565

ppa_alpha_update_mode_t **fg_alpha_update_mode**

Select whether the alpha channel of the input foreground picture needs update

uint32_t **fg_alpha_fix_val**

Range: [0, 255] When PPA_ALPHA_FIX_VALUE mode is selected, alpha_fix_val is the new alpha value to replace the input alpha value (output_alpha = alpha_fix_val)

float **fg_alpha_scale_ratio**

Range: (0, 1) When PPA_ALPHA_SCALE mode is selected, alpha_scale_ratio is the multiplier to the input alpha value (output_alpha = alpha_scale_ratio * input_alpha) Ratio resolution is 1/256

color_pixel_rgb888_data_t **fg_fix_rgb_val**

When in_fg.blend_cm is PPA_BLEND_COLOR_MODE_A8/4, this field can be used to set a fixed color for the foreground, in RGB888 format

bool **bg_ck_en**

Whether to enable color keying for background If not enabled, all background pixels are considered as out of the color-keying range

color_pixel_rgb888_data_t **bg_ck_rgb_low_thres**

The lower threshold of the color-keying range for the background, in RGB888 format

color_pixel_rgb888_data_t **bg_ck_rgb_high_thres**

The higher threshold of the color-keying range for the background, in RGB888 format

bool **fg_ck_en**

Whether to enable color keying for foreground If not enabled, all foreground pixels are considered as out of the color-keying range

color_pixel_rgb888_data_t **fg_ck_rgb_low_thres**

The lower threshold of the color-keying range for the foreground, in RGB888 format

color_pixel_rgb888_data_t **fg_ck_rgb_high_thres**

The higher threshold of the color-keying range for the foreground, in RGB888 format

color_pixel_rgb888_data_t **ck_rgb_default_val**

The color to overwrite when a pixel, where its background element and foreground element are both within their color-keying ranges, in RGB888 format

bool **ck_reverse_bg2fg**

If this bit is set, in color-keying, for the pixel, where its background element is in the color range, but its foreground element is not in the color range, it will output the foreground element instead of the background element

ppa_trans_mode_t **mode**

Determines whether to block inside the operation functions, see `ppa_trans_mode_t`

void ***user_data**

User registered data to be passed into `done_cb` callback function

struct **ppa_fill_oper_config_t**

A collection of configuration items to do a PPA fill operation transaction.

Public Members

ppa_out_pic_blk_config_t **out**

Information of the output picture and the target block

uint32_t **fill_block_w**

The width of the block to be filled (unit: pixel)

uint32_t **fill_block_h**

The height of the block to be filled (unit: pixel)

color_pixel_argb8888_data_t **fill_argb_color**

The color to be filled, in ARGB8888 format

ppa_trans_mode_t **mode**

Determines whether to block inside the operation functions, see `ppa_trans_mode_t`

void ***user_data**

User registered data to be passed into `done_cb` callback function

Type Definitions

typedef struct ppa_client_t ***ppa_client_handle_t**

Type of PPA client handle.

typedef bool (***ppa_event_callback_t**)(*ppa_client_handle_t* ppa_client, *ppa_event_data_t* *event_data, void *user_data)

Type of PPA event callback.

Param ppa_client **[in]** PPA client handle
Param event_data **[in]** PPA event data
Param user_data **[in]** User registered data from calling `ppa_do_xxx` to perform an operation
Return Whether a task switch is needed after the callback function returns, this is usually due to the callback wakes up some high priority task.

Enumerations

enum **ppa_operation_t**

Enumeration of all PPA available operations.

Values:

enumerator **PPA_OPERATION_SRM**

Do scale-rotate-mirror operation

enumerator **PPA_OPERATION_BLEND**

Do blend operation

enumerator **PPA_OPERATION_FILL**

Do fill operation, use one constant pixel to fill a target window

enumerator **PPA_OPERATION_INVALID**

Invalid PPA operations, indicates the quantity of available PPA operations

enum **ppa_trans_mode_t**

Modes to perform the PPA operations.

Values:

enumerator **PPA_TRANS_MODE_BLOCKING**

`ppa_do_xxx` function will block until the PPA operation is finished

enumerator **PPA_TRANS_MODE_NON_BLOCKING**

`ppa_do_xxx` function will return immediately after the PPA operation is pushed to the internal queue

Header File

- components/hal/include/hal/ppa_types.h
- This header file can be included with:

```
#include "hal/ppa_types.h"
```

Enumerations

enum **ppa_engine_type_t**

Enumeration of engines in PPA modules.

Values:

enumerator **PPA_ENGINE_TYPE_SRM**

PPA Scaling-Rotating-Mirroring (SRM) engine, used to perform scale, rotate, mirror

enumerator **PPA_ENGINE_TYPE_BLEND**

PPA Blending engine, used to perform blend or fill

enum **ppa_srm_rotation_angle_t**

Enumeration of PPA Scaling-Rotating-Mirroring available rotation angle (in the counterclockwise direction)

Values:

enumerator **PPA_SRM_ROTATION_ANGLE_0**

Picture does no rotation

enumerator **PPA_SRM_ROTATION_ANGLE_90**

Picture rotates 90 degrees CCW

enumerator **PPA_SRM_ROTATION_ANGLE_180**

Picture rotates 180 degrees CCW

enumerator **PPA_SRM_ROTATION_ANGLE_270**

Picture rotates 270 degrees CCW

enum **ppa_srm_color_mode_t**

Enumeration of PPA Scaling-Rotating-Mirroring available color mode.

Values:

enumerator **PPA_SRM_COLOR_MODE_ARGB8888**

PPA SRM color mode: ARGB8888

enumerator **PPA_SRM_COLOR_MODE_RGB888**

PPA SRM color mode: RGB888

enumerator **PPA_SRM_COLOR_MODE_RGB565**

PPA SRM color mode: RGB565

enumerator **PPA_SRM_COLOR_MODE_YUV420**

PPA SRM color mode: YUV420

enumerator **PPA_SRM_COLOR_MODE_YUV444**

PPA SRM color mode: YUV444 (limited range only)

enum **ppa_blend_color_mode_t**

Enumeration of PPA blend available color mode.

Values:

enumerator **PPA_BLEND_COLOR_MODE_ARGB8888**

PPA blend color mode: ARGB8888

enumerator **PPA_BLEND_COLOR_MODE_RGB888**

PPA blend color mode: RGB888

enumerator **PPA_BLEND_COLOR_MODE_RGB565**

PPA blend color mode: RGB565

enumerator **PPA_BLEND_COLOR_MODE_A8**

PPA blend color mode: A8, only available on blend foreground input

enumerator **PPA_BLEND_COLOR_MODE_A4**

PPA blend color mode: A4, only available on blend foreground input

enum **ppa_fill_color_mode_t**

Enumeration of PPA fill available color mode.

Values:

enumerator **PPA_FILL_COLOR_MODE_ARGB8888**

PPA fill color mode: ARGB8888

enumerator **PPA_FILL_COLOR_MODE_RGB888**

PPA fill color mode: RGB888

enumerator **PPA_FILL_COLOR_MODE_RGB565**

PPA fill color mode: RGB565

enum **ppa_alpha_update_mode_t**

Enumeration of PPA alpha compositing update mode.

Values:

enumerator **PPA_ALPHA_NO_CHANGE**

Do not replace alpha value (A' = A). If input format does not contain alpha info, alpha value 255 will be used.

enumerator **PPA_ALPHA_FIX_VALUE**

Replace the alpha value in received pixel with a new, fixed alpha value (A' = val)

enumerator **PPA_ALPHA_SCALE**

Scale the alpha value in received pixel to be a new alpha value (A' = (A * val) » 8). If input format does not contain alpha info, A' = (255 * val) » 8.

enumerator **PPA_ALPHA_INVERT**

Invert the alpha value in received pixel (A' = 255 - A). If input format does not contain alpha info, A' = 0, i.e. a layer with 0% opacity.

enum **ppa_color_conv_std_rgb_yuv_t**

Enumeration of PPA supported color conversion standard between RGB and YUV (determines the YUV<->RGB conversion equation)

Values:

enumerator **PPA_COLOR_CONV_STD_RGB_YUV_BT601**

YUV<->RGB conversion standard: BT.601

enumerator **PPA_COLOR_CONV_STD_RGB_YUV_BT709**

YUV<->RGB conversion standard: BT.709

enum **ppa_color_range_t**

Enumeration of PPA supported color range (determines the YUV<->RGB conversion equation)

Values:

enumerator **PPA_COLOR_RANGE_LIMIT**

Limited color range, 16 is the darkest black and 235 is the brightest white

enumerator **PPA_COLOR_RANGE_FULL**

Full color range, 0 is the darkest black and 255 is the brightest white

enum **ppa_data_burst_length_t**

Enumeration of PPA supported data burst length.

Values:

enumerator **PPA_DATA_BURST_LENGTH_8**

Data burst length: 8 bytes

enumerator **PPA_DATA_BURST_LENGTH_16**

Data burst length: 16 bytes

enumerator **PPA_DATA_BURST_LENGTH_32**

Data burst length: 32 bytes

enumerator **PPA_DATA_BURST_LENGTH_64**

Data burst length: 64 bytes

enumerator **PPA_DATA_BURST_LENGTH_128**

Data burst length: 128 bytes

2.5.23 红外遥控 (RMT)

简介

红外遥控 (RMT) 外设是一个红外发射和接收控制器。其数据格式灵活，可进一步扩展为多功能的通用收发器，发送或接收多种类型的信号。就网络分层而言，RMT 硬件包含物理层和数据链路层。物理层定义

通信介质和比特信号的表示方式，数据链路层定义 RMT 帧的格式。RMT 帧的最小数据单元称为 **RMT 符号**，在驱动程序中以 *rmt_symbol_word_t* 表示。

ESP32-P4 的 RMT 外设存在多个通道[1]，每个通道都可以独立配置为发射器或接收器。

RMT 外设通常支持以下场景：

- 发送或接收红外信号，支持所有红外线协议，如 NEC 协议
- 生成通用序列
- 有限或无限次地在硬件控制的循环中发送信号
- 多通道同时发送
- 将载波调制到输出信号或从输入信号解调载波

RMT 符号的内存布局 RMT 硬件定义了自己的数据模式，称为 **RMT 符号**。下图展示了一个 RMT 符号的位字段：每个符号由两对两个值组成，每对中的第一个值是一个 15 位的值，表示信号持续时间，以 RMT 滴答计。每对中的第二个值是一个 1 位的值，表示信号的逻辑电平，即高电平或低电平。

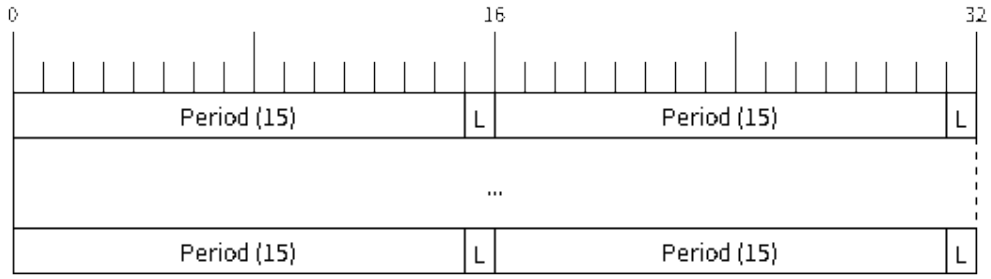


图 21: RMT 符号结构（L - 信号电平）

RMT 发射器概述 RMT 发送通道 (TX Channel) 的数据路径和控制路径如下图所示：

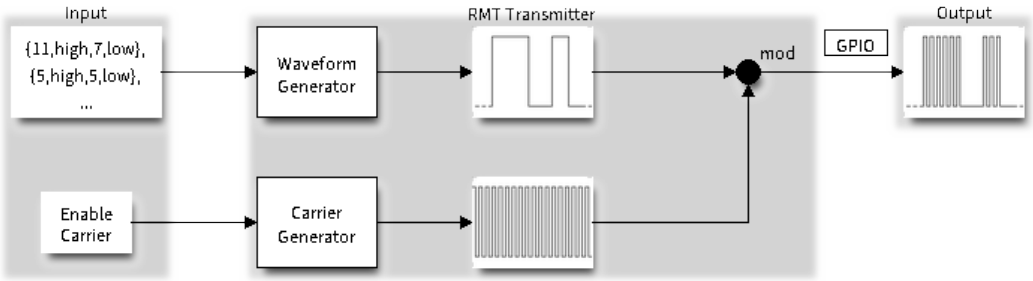


图 22: RMT 发射器概述

驱动程序将用户数据编码为 RMT 数据格式，随后由 RMT 发射器根据编码生成波形。在将波形发送到 GPIO 管脚前，还可以调制高频载波信号。

RMT 接收器概述 RMT 接收通道 (RX Channel) 的数据路径和控制路径如下图所示：

[1] 不同 ESP 芯片系列可能具有不同数量的 RMT 通道，详情请参阅 [TRM]。驱动程序对通道申请数量不做限制，但当硬件资源用尽时，驱动程序将返回错误。因此，每次进行资源分配 时，请注意检查返回值。

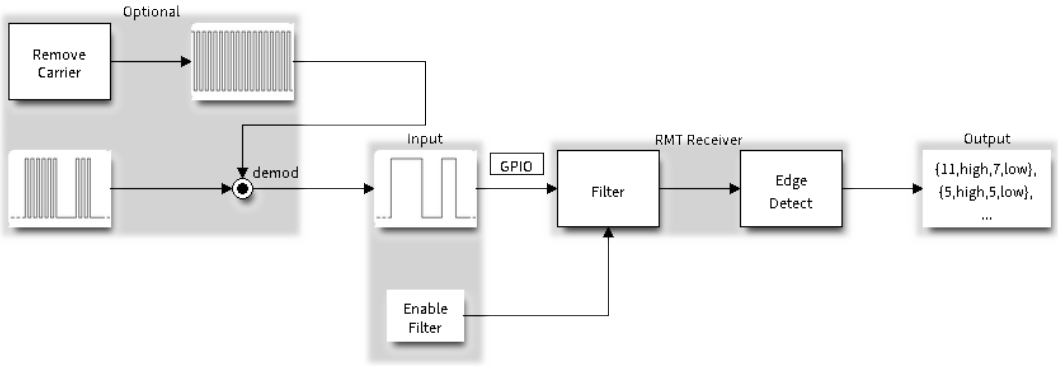


图 23: RMT 接收器概述

RMT 接收器可以对输入信号采样，将其转换为 RMT 数据格式，并将数据存储在内存中。还可以向接收器提供输入信号的基本特征，使其识别信号停止条件，并过滤掉信号干扰和噪声。RMT 外设还支持从基准信号中解调出高频载波信号。

功能概述

下文将分节概述 RMT 的功能：

- *资源分配* - 介绍如何分配和正确配置 RMT 通道，以及如何回收闲置通道及其他资源。
- *载波调制与解调* - 介绍如何调制和解调用于 TX 和 RX 通道的载波信号。
- *注册事件回调* - 介绍如何注册用户提供的事件回调函数以接收 RMT 通道事件。
- *启用及禁用通道* - 介绍如何启用和禁用 RMT 通道。
- *发起 TX 事务* - 介绍发起 TX 通道事务的步骤。
- *发起 RX 事务* - 介绍发起 RX 通道事务的步骤。
- *多通道同时发送* - 介绍如何将多个通道收集到一个同步组中，以便同时启动发送。
- *RMT 编码器* - 介绍如何通过组合驱动程序提供的多个基本编码器来编写自定义编码器。
- *电源管理* - 介绍不同时钟源对功耗的影响。
- *IRAM 安全* - 介绍禁用 cache 对 RMT 驱动程序的影响，并提供应对方案。
- *线程安全* - 介绍由驱动程序认证为线程安全的 API。
- *Kconfig 选项* - 介绍 RMT 驱动程序支持的各种 Kconfig 选项。

资源分配 驱动程序中，`rmt_channel_handle_t` 用于表示 RMT 的 TX 和 RX 通道。驱动程序在内部管理可用的通道，并在收到请求时提供空闲通道。

安装 RMT TX 通道 要安装 RMT TX 通道，应预先提供配置结构体`rmt_tx_channel_config_t`。以下列表介绍了配置结构体中的各个部分。

- `rmt_tx_channel_config_t::gpio_num` 设置发射器使用的 GPIO 编号。
- `rmt_tx_channel_config_t::clk_src` 选择 RMT 通道的时钟源。`rmt_clock_source_t` 中列出了可用的时钟源。注意，其他通道将使用同一所选时钟源，因此，应确保分配的任意 TX 或 RX 通道都享有相同的配置。有关不同时钟源对功耗的影响，请参阅电源管理。
- `rmt_tx_channel_config_t::resolution_hz` 设置内部滴答计数器的分辨率。基于此 **滴答**，可以计算 RMT 信号的定时参数。
- 在启用 DMA 后端和未启用 DMA 后端的情况下，`rmt_tx_channel_config_t::mem_block_symbols` 字段含义稍有不同。
  - 若通过`rmt_tx_channel_config_t::with_dma` 启用 DMA，则该字段可以控制内部 DMA 缓冲区大小。为实现更好的吞吐量、减少 CPU 开销，建议为字段设置一个较大的值，如 `1024`。
  - 如果未启用 DMA，则该字段控制通道专用内存块大小，至少为 48。
- `rmt_tx_channel_config_t::trans_queue_depth` 设置内部事务队列深度。队列越深，在待处理队列中可以准备的事务越多。

- *rmt_tx_channel_config_t::invert_out* 决定是否在将 RMT 信号发送到 GPIO 管脚前反转 RMT 信号。
- *rmt_tx_channel_config_t::with_dma* 为通道启用 DMA 后端。启用 DMA 后端可以释放 CPU 上的大部分通道工作负载，显著减轻 CPU 负担。但并非所有 ESP 芯片都支持 DMA 后端，在启用此选项前，请参阅 [TRM]。若所选芯片不支持 DMA 后端，可能会报告*ESP_ERR_NOT_SUPPORTED* 错误。
- *rmt_tx_channel_config_t::io_loop_back* 启用通道所分配的 GPIO 上的输入和输出功能，将发送通道和接收通道绑定到同一个 GPIO 上，从而实现回环功能。
- *rmt_tx_channel_config_t::io_od_mode* 配置通道分配的 GPIO 为开漏模式 (open-drain)。当与*rmt_tx_channel_config_t::io_loop_back* 结合使用时，可以实现双向总线，如 1-wire。
- *rmt_tx_channel_config_t::intr_priority* 设置中断的优先级。如果设置为 `0`，驱动将会使用一个中低优先级的中断（优先级可能为 1, 2 或 3），否则会使用*rmt_tx_channel_config_t::intr_priority* 指定的优先级。请使用优先级序号 (1, 2, 3)，而不是 bitmask 的形式 ((1«1), (1«2), (1«3))。请注意，中断优先级一旦设置，在*rmt_del_channel()* 被调用之前不可再次修改。

将必要参数填充到结构体*rmt_tx_channel_config_t* 后，可以调用*rmt_new_tx_channel()* 来分配和初始化 TX 通道。如果函数运行正确，会返回 RMT 通道句柄；如果 RMT 资源池内缺少空闲通道，会返回*ESP_ERR_NOT_FOUND* 错误；如果硬件不支持 DMA 后端等部分功能，则返回*ESP_ERR_NOT_SUPPORTED* 错误。

```
rmt_channel_handle_t tx_chan = NULL;
rmt_tx_channel_config_t tx_chan_config = {
    .clk_src = RMT_CLK_SRC_DEFAULT,   // 选择时钟源
    .gpio_num = 0,                    // GPIO 编号
    .mem_block_symbols = 64,          // 内存块大小，即 64 * 4 = 256 字节
    .resolution_hz = 1 * 1000 * 1000, // 1 MHz 滴答分辨率，即 1 滴答 = 1 µs
    .trans_queue_depth = 4,           // 设置后台等待处理的事务数量
    .flags.invert_out = false,        // 不反转输出信号
    .flags.with_dma = false,          // 不需要 DMA 后端
};
ESP_ERROR_CHECK(rmt_new_tx_channel(&tx_chan_config, &tx_chan));
```

安装 RMT RX 通道 要安装 RMT RX 通道，应预先提供配置结构体*rmt_rx_channel_config_t*。以下列表介绍了配置结构体中的各个部分。

- *rmt_rx_channel_config_t::gpio_num* 设置接收器使用的 GPIO 编号。
- *rmt_rx_channel_config_t::clk_src* 选择 RMT 通道的时钟源。*rmt_clock_source_t* 中列出了可用的时钟源。注意，其他通道将使用同一所选时钟源，因此，应确保分配的任意 TX 或 RX 通道都享有相同的配置。有关不同时钟源对功耗的影响，请参阅电源管理。
- *rmt_rx_channel_config_t::resolution_hz* 设置内部滴答计数器的分辨率。基于此 **滴答**，可以计算 RMT 信号的定时参数。
- 在启用 DMA 后端和未启用 DMA 后端的情况下，*rmt_rx_channel_config_t::mem_block_symbols* 字段含义稍有不同。
  - 若通过*rmt_rx_channel_config_t::with_dma* 启用 DMA，则该字段可以最大化控制内部 DMA 缓冲区大小。
  - 如果未启用 DMA，则该字段控制通道专用内存块大小，至少为 48。
- *rmt_rx_channel_config_t::invert_in* 在输入信号传递到 RMT 接收器前对其进行反转。该反转由 GPIO 交换矩阵完成，而非 RMT 外设。
- *rmt_rx_channel_config_t::with_dma* 为通道启用 DMA 后端。启用 DMA 后端可以释放 CPU 上的大部分通道工作负载，显著减轻 CPU 负担。但并非所有 ESP 芯片都支持 DMA 后端，在启用此选项前，请参阅 [TRM]。若所选芯片不支持 DMA 后端，可能会报告*ESP_ERR_NOT_SUPPORTED* 错误。
- *rmt_rx_channel_config_t::io_loop_back* 启用通道所分配的 GPIO 上的输入和输出功能，将发送通道和接收通道绑定到同一个 GPIO 上，从而实现回环功能。
- *rmt_rx_channel_config_t::intr_priority* 设置中断的优先级。如果设置为 `0`，驱动将会使用一个中低优先级的中断（优先级可能为 1, 2 或 3），否则会使用*rmt_rx_channel_config_t::intr_priority* 指定的优先级。请使用优先级序号 (1, 2, 3)，而不是 bitmask 的形式 ((1«1), (1«2), (1«3))。请注意，中断优先级一旦设置，在*rmt_del_channel()*

被调用之前不可再次修改。

将必要参数填充到结构体*rmt_rx_channel_config_t* 后，可以调用*rmt_new_rx_channel()* 来分配和初始化 RX 通道。如果函数运行正确，会返回 RMT 通道句柄；如果 RMT 资源池内缺少空闲通道，会返回*ESP_ERR_NOT_FOUND* 错误；如果硬件不支持 DMA 后端等部分功能，则返回*ESP_ERR_NOT_SUPPORTED* 错误。

```
rmt_channel_handle_t rx_chan = NULL;
rmt_rx_channel_config_t rx_chan_config = {
    .clk_src = RMT_CLK_SRC_DEFAULT,   // 选择时钟源
    .resolution_hz = 1 * 1000 * 1000, // 1 MHz 滴答分辨率，即 1 滴答 = 1 µs
    .mem_block_symbols = 64,          // 内存块大小，即 64 * 4 = 256 字节
    .gpio_num = 2,                    // GPIO 编号
    .flags.invert_in = false,         // 不反转输入信号
    .flags.with_dma = false,          // 不需要 DMA 后端
};
ESP_ERROR_CHECK(rmt_new_rx_channel(&rx_chan_config, &rx_chan));
```

备注： 由于 GPIO 驱动程序中的软件限制，当 TX 和 RX 通道都绑定到同一 GPIO 时，请确保在 TX 通道之前初始化 RX 通道。如果先设置 TX 通道，那么在 RX 通道设置期间，GPIO 控制信号将覆盖先前的 RMT TX 通道信号。

卸载 RMT 通道 如果不再需要之前安装的 RMT 通道，建议调用*rmt_del_channel()* 回收资源，使底层软件与硬件重新用于其他功能。

载波调制与解调 RMT 发射器可以生成载波信号，并将其调制到消息信号上。载波信号的频率远高于消息信号。此外，仅支持配置载波信号的频率和占空比。RMT 接收器可以从输入信号中解调出载波信号。注意，并非所有 ESP 芯片都支持载波调制和解调功能，在配置载波前，请参阅 [TRM]。若所选芯片不支持载波调制和解调功能，可能会报告*ESP_ERR_NOT_SUPPORTED* 错误。

载波相关配置位于*rmt_carrier_config_t* 中，该配置中的各部分详情如下：

- *rmt_carrier_config_t::frequency_hz* 设置载波频率，单位为 Hz。
- *rmt_carrier_config_t::duty_cycle* 设置载波占空比。
- *rmt_carrier_config_t::polarity_active_low* 设置载波极性，即应用载波的电平。
- *rmt_carrier_config_t::always_on* 设置是否在数据发送完成后仍输出载波，该配置仅适用于 TX 通道。

备注： RX 通道的载波频率不应设置为理论值，建议为载波频率留出一定的容差。例如，以下代码片段的载波频率设置为 25 KHz，而非 TX 侧配置的 38 KHz。因为信号在空气中传播时会发生反射和折射，导致接收端接收的频率失真。

```
rmt_carrier_config_t tx_carrier_cfg = {
    .duty_cycle = 0.33,                 // 载波占空比为 33%
    .frequency_hz = 38000,              // 38 KHz
    .flags.polarity_active_low = false, // 载波应调制到高电平
};
// 将载波调制到 TX 通道
ESP_ERROR_CHECK(rmt_apply_carrier(tx_chan, &tx_carrier_cfg));

rmt_carrier_config_t rx_carrier_cfg = {
    .duty_cycle = 0.33,                 // 载波占空比为 33%
    .frequency_hz = 25000,              // 载波频率为 25␣
↪KHz，应小于发射器的载波频率
    .flags.polarity_active_low = false, // 载波调制到高电平
};
```

(下页继续)

(续上页)

```
// 从 RX 通道解调载波
ESP_ERROR_CHECK(rmt_apply_carrier(rx_chan, &rx_carrier_cfg));
```

注册事件回调 当 RMT 通道生成发送或接收完成等事件时，会通过中断告知 CPU。如果需要在发生特定事件时调用函数，可以为 TX 和 RX 通道分别调用*rmt_tx_register_event_callbacks()*和*rmt_rx_register_event_callbacks()*，向 RMT 驱动程序的中断服务程序 (ISR) 注册事件回调。由于上述回调函数是在 ISR 中调用的，因此，这些函数不应涉及 block 操作。可以检查调用 API 的后缀，确保在函数中只调用了后缀为 ISR 的 FreeRTOS API。回调函数具有布尔返回值，指示回调是否解除了更高优先级任务的阻塞状态。

有关 TX 通道支持的事件回调，请参阅*rmt_tx_event_callbacks_t*：

- *rmt_tx_event_callbacks_t::on_trans_done* 为“发送完成”的事件设置回调函数，函数原型声明为*rmt_tx_done_callback_t*。

有关 RX 通道支持的事件回调，请参阅*rmt_rx_event_callbacks_t*：

- *rmt_rx_event_callbacks_t::on_recv_done* 为“接收完成”的事件设置回调函数，函数原型声明为*rmt_rx_done_callback_t*。

也可使用参数 user_data，在*rmt_tx_register_event_callbacks()*和*rmt_rx_register_event_callbacks()*中保存自定义上下文。用户数据将直接传递给每个回调函数。

备注： ”receive-done” 不等同于”receive-finished”. 这个回调函数也可以在”partial-receive-done” 时间发生的时候被调用。

在回调函数中可以获取驱动程序在 edata 中填充的特定事件数据。注意，edata 指针仅在回调的持续时间内有效。

有关 TX 完成事件数据的定义，请参阅*rmt_tx_done_event_data_t*：

- *rmt_tx_done_event_data_t::num_symbols* 表示已发送的 RMT 符号数量，也反映了编码数据大小。注意，该值还考虑了由驱动程序附加的 EOF 符号，该符号标志着一次事务的结束。

有关 RX 完成事件数据的定义，请参阅*rmt_rx_done_event_data_t*：

- *rmt_rx_done_event_data_t::received_symbols* 指向接收到的 RMT 符号，这些符号存储在*rmt_receive()* 函数的 buffer 参数中，在回调函数返回前不应释放此接收缓冲区。如果你还启用了部分接收的功能，则这个用户缓冲区会被用作“二级缓冲区”，其中的内容可以被随后传入的数据覆盖。在这种情况下，如果你想要保存或者稍后处理一些数据，则需要将接收到的数据复制到其他位置。
- *rmt_rx_done_event_data_t::num_symbols* 表示接收到的 RMT 符号数量，该值不会超过*rmt_receive()* 函数的 buffer_size 参数。如果 buffer_size 不足以容纳所有接收到的 RMT 符号，驱动程序将只保存缓冲区能够容纳的最大数量的符号，并丢弃或忽略多余的符号。
- *rmt_rx_done_event_data_t::is_last* 指示收到的数据包是否是当前的接收任务中的最后一个。这个标志在你使能*rmt_receive_config_t::extra_rmt_receive_flags::en_partial_rx* 部分接收功能时非常有用。

启用及禁用通道 在发送或接收 RMT 符号前，应预先调用*rmt_enable()*。启用 TX 通道会启用特定中断，并使硬件准备发送事务。启用 RX 通道也会启用中断，但由于传入信号的特性尚不明确，接收器不会在此时启动，而是在*rmt_receive()* 中启动。

相反，*rmt_disable()* 会禁用中断并清除队列中的中断，同时禁用发射器和接收器。

```
ESP_ERROR_CHECK(rmt_enable(tx_chan));
ESP_ERROR_CHECK(rmt_enable(rx_chan));
```

发起 TX 事务 RMT 是一种特殊的通信外设，无法像 SPI 和 I2C 那样发送原始字节流，只能以 *rmt_symbol_word_t* 格式发送数据。然而，硬件无法将用户数据转换为 RMT 符号，该转换只能通过 RMT 编码器在软件中完成。编码器将用户数据编码为 RMT 符号，随后写入 RMT 内存块或 DMA 缓冲区。有关创建 RMT 编码器的详细信息，请参阅 *RMT 编码器*。

获取编码器后，调用 *rmt_transmit()* 启动 TX 事务，该函数会接收少数位置参数，如通道句柄、编码器句柄和有效负载缓冲区。此外，还需要在 *rmt_transmit_config_t* 中提供专用于发送的配置，具体如下：

- *rmt_transmit_config_t::loop_count* 设置发送的循环次数。在发射器完成一轮发送后，如果该值未设置为零，则再次启动相同的发送程序。由于循环由硬件控制，RMT 通道可以在几乎不需要 CPU 干预的情况下，生成许多周期性序列。
  - 将 *rmt_transmit_config_t::loop_count* 设置为 `-1`，会启用无限循环发送机制，此时，除非手动调用 *rmt_disable()*，否则通道不会停止，也不会生成“完成发送”事件。
  - 将 *rmt_transmit_config_t::loop_count* 设置为正数，意味着迭代次数有限。此时，“完成发送”事件在指定的迭代次数完成后发生。

  备注： 注意，不是所有 ESP 芯片都支持 **循环发送**功能，在配置此选项前，请参阅 [TRM]。若所选芯片不支持配置此选项，可能会报告 *ESP_ERR_NOT_SUPPORTED* 错误。

- *rmt_transmit_config_t::eot_level* 设置发射器完成工作时的输出电平，该设置同时适用于调用 *rmt_disable()* 停止发射器工作时的输出电平。
- *rmt_transmit_config_t::queue_nonblocking* 设置当传输队列满的时候该函数是否需要等待。如果该值设置为 `true` 那么当遇到队列满的时候，该函数会立即返回错误代码 *ESP_ERR_INVALID_STATE*。否则，函数会阻塞当前线程，直到传输队列有空档。

备注： 如果将 *rmt_transmit_config_t::loop_count* 设置为非零值，即启用循环功能，则传输的大小将受到限制。编码的 RMT 符号不应超过 RMT 硬件内存块容量，否则会出现类似 `encoding artifacts can't exceed hw memory block for loop transmission` 的报错信息。如需通过循环启动大型事务，请尝试以下任一方法：

- 增加 *rmt_tx_channel_config_t::mem_block_symbols*。若此时启用了 DMA 后端，该方法将失效。
- 自定义编码器，并在编码函数中构造一个无限循环，详情请参阅 *RMT 编码器*。

rmt_transmit() 会在其内部构建一个事务描述符，并将其发送到作业队列中，该队列通常会在 ISR 上下文中被调度。因此，在 *rmt_transmit()* 返回时，该事务可能尚未启动。注意，你不能在事务结束前就去回收或者修改 payload 中的内容。通过 *rmt_tx_register_event_callbacks()* 来注册事件回调，可以在事务完成的时候被通知。为确保完成所有挂起的事务，你还可以调用 *rmt_tx_wait_all_done()*。

多通道同时发送 在一些实时控制应用程序中，启动多个 TX 通道（例如使两个器械臂同时移动）时，应避免出现任何时间漂移。为此，RMT 驱动程序可以创建 **同步管理器**帮助管理该过程。在驱动程序中，同步管理器为 *rmt_sync_manager_handle_t*。RMT 同步发送过程如下图所示：

安装 RMT 同步管理器 要创建同步管理器，应预先在 *rmt_sync_manager_config_t* 中指定要管理的通道：

- *rmt_sync_manager_config_t::tx_channel_array* 指向要管理的 TX 通道数组。
- *rmt_sync_manager_config_t::array_size* 设置要管理的通道数量。

成功调用 *rmt_new_sync_manager()* 函数将返回管理器句柄，该函数也可能因为无效参数等错误而无法调用。在已经安装了同步管理器，且缺少硬件资源来创建另一个管理器时，该函数将报告 *ESP_ERR_NOT_FOUND* 错误。此外，如果硬件不支持同步管理器，将报告 *ESP_ERR_NOT_SUPPORTED* 错误。在使用同步管理器功能之前，请参阅 [TRM]。

发起同时发送 在调用 *rmt_sync_manager_config_t::tx_channel_array* 中所有通道上的 *rmt_transmit()* 前，任何受管理的 TX 通道都不会启动发送机制，而是处于待命状态。由于各

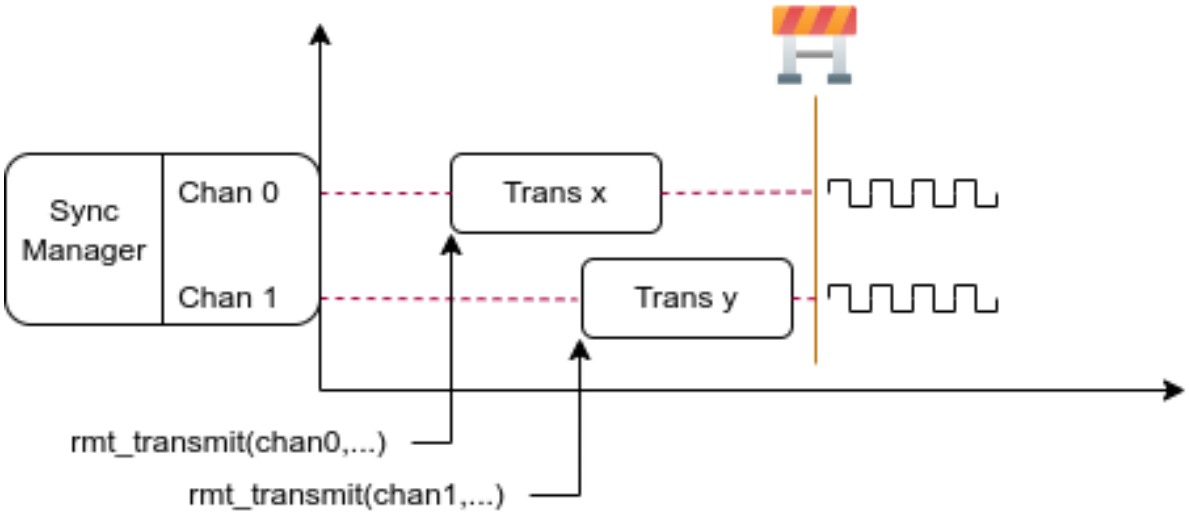


图 24: RMT TX 同步发送

通道事务不同，TX 通道通常会在不同的时间完成相应事务，这可能导致无法同步。因此，在重新启动同时发送程序之前，应调用 *rmt_sync_reset()* 函数重新同步所有通道。

调用 *rmt_del_sync_manager()* 函数可以回收同步管理器，并使通道可以在将来独立启动发送程序。

```
rmt_channel_handle_t tx_channels[2] = {NULL}; // 声明两个通道
int tx_gpio_number[2] = {0, 2};
// 依次安装通道
for (int i = 0; i < 2; i++) {
    rmt_tx_channel_config_t tx_chan_config = {
        .clk_src = RMT_CLK_SRC_DEFAULT,        // 选择时钟源
        .gpio_num = tx_gpio_number[i],     // GPIO 编号
        .mem_block_symbols = 64,           // 内存块大小，即 64 * 4 = 256 字节
        .resolution_hz = 1 * 1000 * 1000, // 1 MHz 分辨率
        .trans_queue_depth = 1,            // 设置可以在后台挂起的事务数量
    };
    ESP_ERROR_CHECK(rmt_new_tx_channel(&tx_chan_config, &tx_channels[i]));
}
// 安装同步管理器
rmt_sync_manager_handle_t synchro = NULL;
rmt_sync_manager_config_t synchro_config = {
    .tx_channel_array = tx_channels,
    .array_size = sizeof(tx_channels) / sizeof(tx_channels[0]),
};
ESP_ERROR_CHECK(rmt_new_sync_manager(&synchro_config, &synchro));

ESP_ERROR_CHECK(rmt_transmit(tx_channels[0], led_strip_encoders[0], led_data, led_
↪num * 3, &transmit_config));
// 只有在调用 tx_channels[1] 的 rmt_transmit() 函数返回后，tx_channels[0]␣
↪才会开始发送数据。
ESP_ERROR_CHECK(rmt_transmit(tx_channels[1], led_strip_encoders[1], led_data, led_
↪num * 3, &transmit_config));
```

发起 RX 事务 如*启用及禁用通道*一节所述，仅调用 *rmt_enable()* 时，RX 通道无法接收 RMT 符号。为此，应在 *rmt_receive_config_t* 中指明传入信号的基本特征：

- *rmt_receive_config_t::signal_range_min_ns* 指定高电平或低电平有效脉冲的最小持续时间。如果脉冲宽度小于指定值，硬件会将其视作干扰信号并忽略。
- *rmt_receive_config_t::signal_range_max_ns* 指定高电平或低电平有效脉冲的最大持续时间。如果脉冲宽度大于指定值，接收器会将其视作 **停止信号**，并立即生成接收完成事件。

- 如果传入的数据包很长，无法一次性保存在用户缓冲区中，可以通过将*rmt_receive_config_t::extra_rmt_receive_flags::en_partial_rx* 设置为 `true` 来开启部分接收功能。在这种情况下，当用户缓冲区快满的时候，驱动会多次调用*rmt_rx_event_callbacks_t::on_recv_done* 回调函数来通知用户去处理已经收到的数据。你可以检查:cpp:member::*rmt_rx_done_event_data_t::is_last* 的值来了解当前事务是否已经结束。请注意，并不是所有 ESP 系列芯片都支持这个功能，它依赖硬件提供的“ping-pong 接收”或者“DMA 接收”的能力。

根据以上配置调用*rmt_receive()* 后，RMT 接收器会启动 RX 机制。注意，以上配置均针对特定事务存在，也就是说，要开启新一轮的接收时，需要再次设置*rmt_receive_config_t* 选项。接收器会将传入信号以*rmt_symbol_word_t* 的格式保存在内部内存块或 DMA 缓冲区中。

由于内存块大小有限，RMT 接收器会交替提醒驱动程序将累积的符号复制到外部处理。

应在*rmt_receive()* 函数的 `buffer` 参数中提供复制目标。如果由于缓冲区大小不足而导致缓冲区溢出，接收器仍可继续工作，但会丢弃溢出的符号，并报告此错误信息：`user buffer too small, received symbols truncated`。请注意 `buffer` 参数的生命周期，确保在接收器完成或停止工作前不会回收缓冲区。

当接收器完成工作，即接收到持续时间大于*rmt_receive_config_t::signal_range_max_ns* 的信号时，驱动程序将停止接收器。如有需要，应再次调用*rmt_receive()* 重新启动接收器。在*rmt_rx_event_callbacks_t::on_recv_done* 的回调中可以获取接收到的数据。要获取更多有关详情，请参阅注册事件回调。

```
static bool example_rmt_rx_done_callback(rmt_channel_handle_t channel, const rmt_
↪rx_done_event_data_t *edata, void *user_data)
{
    BaseType_t high_task_wakeup = pdFALSE;
    QueueHandle_t receive_queue = (QueueHandle_t)user_data;
    // 将接收到的 RMT 符号发送到解析任务的消息队列中
    xQueueSendFromISR(receive_queue, edata, &high_task_wakeup);
    // 返回是否唤醒了任何任务
    return high_task_wakeup == pdTRUE;
}

QueueHandle_t receive_queue = xQueueCreate(1, sizeof(rmt_rx_done_event_data_t));
rmt_rx_event_callbacks_t cbs = {
    .on_recv_done = example_rmt_rx_done_callback,
};
ESP_ERROR_CHECK(rmt_rx_register_event_callbacks(rx_channel, &cbs, receive_queue));

// 以下时间要求均基于 NEC 协议
rmt_receive_config_t receive_config = {
    .signal_range_min_ns = 1250,     // NEC 信号的最短持续时间为 560 µs，由于 1250␣
↪ns < 560 µs，有效信号不会视为噪声
    .signal_range_max_ns = 12000000, // NEC 信号的最长持续时间为 9000 µs，由于␣
↪12000000 ns > 9000 µs，接收不会提前停止
};

rmt_symbol_word_t raw_symbols[64]; // 64 个符号应足够存储一个标准 NEC 帧的数据
// 准备开始接收
ESP_ERROR_CHECK(rmt_receive(rx_channel, raw_symbols, sizeof(raw_symbols), &receive_
↪config));
// 等待 RX 完成信号
rmt_rx_done_event_data_t rx_data;
xQueueReceive(receive_queue, &rx_data, portMAX_DELAY);
// 解析接收到的符号数据
example_parse_nec_frame(rx_data.received_symbols, rx_data.num_symbols);
```

RMT 编码器 RMT 编码器是 RMT TX 事务的一部分，用于在特定时间生成正确的 RMT 符号，并将其写入硬件内存或 DMA 缓冲区。对于编码函数，存在以下特殊限制条件：

- 由于目标 RMT 内存块无法一次性容纳所有数据，在单个事务中，须多次调用编码函数。为突破这一限制，可以采用 **交替**方式，将编码会话分成多个部分。为此，编码器需要 **记录其状态**，以便从上一部分编码结束之处继续编码。
- 编码函数在 ISR 上下文中运行。为加快编码会话，建议将编码函数放入 IRAM，这也有助于避免在编码过程中出现 cache 失效的情况。

为帮助用户更快速地上手 RMT 驱动程序，该程序默认提供了一些常用编码器，可以单独使用，也可以链式组合成新的编码器，有关原理请参阅 组合模式。驱动程序在*rmt_encoder_t* 中定义了编码器接口，包含以下函数：

- *rmt_encoder_t::encode* 是编码器的基本函数，编码会话即在此处进行。
  - 在单个事务中，可能会多次调用*rmt_encoder_t::encode* 函数，该函数会返回当前编码会话的状态。
  - 可能出现的编码状态已在*rmt_encode_state_t* 列出。如果返回结果中包含*RMT_ENCODING_COMPLETE*，表示当前编码器已完成编码。
  - 如果返回结果中包含*RMT_ENCODING_MEM_FULL*，表示保存编码数据的空间不足，需要从当前会话中退出。
- *rmt_encoder_t::reset* 会将编码器重置为初始状态（编码器有其特定状态）。
  - 如果在未重置 RMT 发射器对应编码器的情况下，手动停止 RMT 发射器，随后的编码会话将报错。
  - 该函数也会在*rmt_disable()* 中隐式调用。
- *rmt_encoder_t::del* 可以释放编码器分配的资源。

拷贝编码器 调用*rmt_new_copy_encoder()* 可以创建拷贝编码器，将 RMT 符号从用户空间复制到驱动程序层。拷贝编码器通常用于编码 `const` 数据，即初始化后在运行时不会发生更改的数据，如红外协议中的前导码。

调用*rmt_new_copy_encoder()* 前，应预先提供配置结构体*rmt_copy_encoder_config_t*。目前，该配置保留用作未来的扩展功能，暂无具体用途或设置项。

字节编码器 调用*rmt_new_bytes_encoder()* 可以创建字节编码器，将用户空间的字节流动态转化成 RMT 符号。字节编码区通常用于编码动态数据，如红外协议中的地址和命令字段。

调用*rmt_new_bytes_encoder()* 前，应预先提供配置结构体*rmt_bytes_encoder_config_t*，具体配置如下：

- *rmt_bytes_encoder_config_t::bit0* 和*rmt_bytes_encoder_config_t::bit1* 为必要项，用于告知编码器如何以*rmt_symbol_word_t* 格式表示零位和一位。
- *rmt_bytes_encoder_config_t::msb_first* 设置各字节的位编码。如果设置为真，编码器将首先编码 **最高有效位**，否则将首先编码 **最低有效位**。

除驱动程序提供的原始编码器外，也可以将现有编码器链式组合成自定义编码器。常见编码器链如下图所示：

自定义 NEC 协议的 RMT 编码器 本节将演示编写 NEC 编码器的流程。NEC 红外协议使用脉冲距离编码来发送消息位，每个脉冲突发的持续时间为 `562.5 µs`，逻辑位发送详见下文。注意，各字节的最低有效位会优先发送。

- 逻辑 `0`：`562.5 µs` 的脉冲突发后有 `562.5 µs` 的空闲时间，总发送时间为 `1.125 ms`
- 逻辑 `1`：`562.5 µs` 的脉冲突发后有 `1.6875 ms` 的空闲时间，总发送时间为 `2.25 ms`

在遥控器上按下某个按键时，将按以下顺序发送有关信号：

- `9 ms` 的引导脉冲发射，也称为 AGC 脉冲
- `4.5 ms` 的空闲时间
- 接收设备的 8 位地址
- 地址的 8 位逻辑反码
- 8 位命令
- 命令的 8 位逻辑反码
- 最后的 `562.5 µs` 脉冲突发，表示消息发送结束

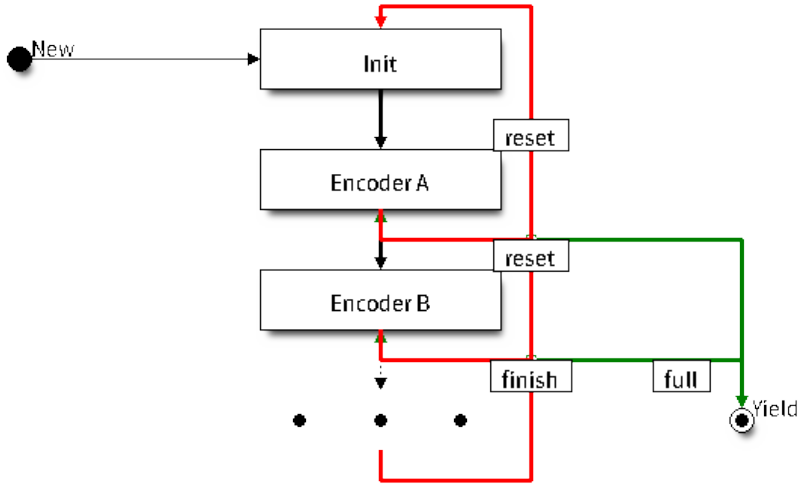


图 25: RMT 编码器链

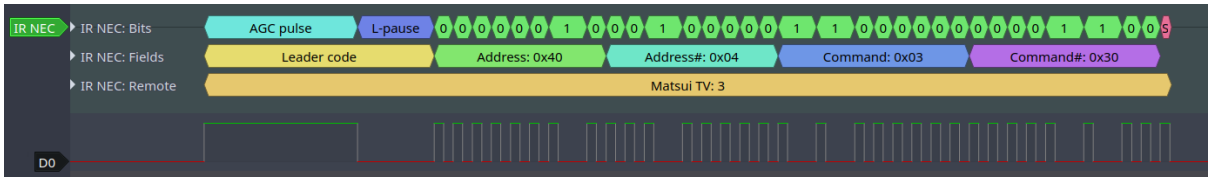


图 26: 红外 NEC 帧

随后可以按相同顺序构建 NEC *rmt_encoder_t::encode* 函数，例如

```
// 红外 NEC 扫码表示法
typedef struct {
    uint16_t address;
    uint16_t command;
} ir_nec_scan_code_t;

// 通过组合原始编码器构建编码器
typedef struct {
    rmt_encoder_t base;           // 基础类 "class" 声明了标准编码器接口
    rmt_encoder_t *copy_encoder;  // 使用拷贝编码器来编码前导码和结束码
    rmt_encoder_t *bytes_encoder; // 使用字节编码器来编码地址和命令数据
    rmt_symbol_word_t nec_leading_symbol; // 使用 RMT 表示的 NEC 前导码
    rmt_symbol_word_t nec_ending_symbol;  // 使用 RMT 表示的 NEC 结束码
    int state; // 记录当前编码状态，即所处编码阶段
} rmt_ir_nec_encoder_t;

static size_t rmt_encode_ir_nec(rmt_encoder_t *encoder, rmt_channel_handle_t␣
↪channel, const void *primary_data, size_t data_size, rmt_encode_state_t *ret_
↪state)
{
    rmt_ir_nec_encoder_t *nec_encoder = __containerof(encoder, rmt_ir_nec_encoder_
↪t, base);
    rmt_encode_state_t session_state = RMT_ENCODING_RESET;
    rmt_encode_state_t state = RMT_ENCODING_RESET;
    size_t encoded_symbols = 0;
    ir_nec_scan_code_t *scan_code = (ir_nec_scan_code_t *)primary_data;
    rmt_encoder_handle_t copy_encoder = nec_encoder->copy_encoder;
    rmt_encoder_handle_t bytes_encoder = nec_encoder->bytes_encoder;
    switch (nec_encoder->state) {
```

(下页继续)

(续上页)

```
    case 0: // 发送前导码
        encoded_symbols += copy_encoder->encode(copy_encoder, channel, &nec_
↪encoder->nec_leading_symbol,
                                                sizeof(rmt_symbol_word_t), &
↪session_state);
        if (session_state & RMT_ENCODING_COMPLETE) {
            nec_encoder->state = 1; //␣
↪只有在当前编码器完成工作时才能切换到下一个状态
        }
        if (session_state & RMT_ENCODING_MEM_FULL) {
            state |= RMT_ENCODING_MEM_FULL;
            goto out; //␣
↪如果没有足够的空间来存放其他编码相关的数据，程序会暂停当前操作，并跳转到指定位置继续执行。
        }
    // 继续执行
    case 1: // 发送地址
        encoded_symbols += bytes_encoder->encode(bytes_encoder, channel, &scan_
↪code->address, sizeof(uint16_t), &session_state);
        if (session_state & RMT_ENCODING_COMPLETE) {
            nec_encoder->state = 2; //␣
↪只有在当前编码器完成工作时才能切换到下一个状态
        }
        if (session_state & RMT_ENCODING_MEM_FULL) {
            state |= RMT_ENCODING_MEM_FULL;
            goto out; //␣
↪如果没有足够的空间来存放其他编码相关的数据，程序会暂停当前操作，并跳转到指定位置继续执行。
        }
    // 继续执行
    case 2: // 发送命令
        encoded_symbols += bytes_encoder->encode(bytes_encoder, channel, &scan_
↪code->command, sizeof(uint16_t), &session_state);
        if (session_state & RMT_ENCODING_COMPLETE) {
            nec_encoder->state = 3; //␣
↪只有在当前编码器完成工作时才能切换到下一个状态
        }
        if (session_state & RMT_ENCODING_MEM_FULL) {
            state |= RMT_ENCODING_MEM_FULL;
            goto out; //␣
↪如果没有足够的空间来存放其他编码相关的数据，程序会暂停当前操作，并跳转到指定位置继续执行。
        }
    // 继续执行
    case 3: // 发送结束码
        encoded_symbols += copy_encoder->encode(copy_encoder, channel, &nec_
↪encoder->nec_ending_symbol,
                                                sizeof(rmt_symbol_word_t), &
↪session_state);
        if (session_state & RMT_ENCODING_COMPLETE) {
            nec_encoder->state = RMT_ENCODING_RESET; // 返回初始编码会话
            state |= RMT_ENCODING_COMPLETE; // 告知调用者 NEC 编码已完成
        }
        if (session_state & RMT_ENCODING_MEM_FULL) {
            state |= RMT_ENCODING_MEM_FULL;
            goto out; //␣
↪如果没有足够的空间来存放其他编码相关的数据，程序会暂停当前操作，并跳转到指定位置继续执行。
        }
    }
out:
    *ret_state = state;
    return encoded_symbols;
}
```

完整示例代码存放在 peripherals/rmt/ir_nec_transceiver 目录下。以上代码片段使用了 `switch-case` 和一

些 goto 语句实现了一个 有限状态机，借助此模式可构建更复杂的红外协议。

电源管理 通过*CONFIG_PM_ENABLE* 选项启用电源管理时，系统会在进入 Light-sleep 模式前调整 APB 频率。该操作可能改变 RMT 内部计数器的分辨率。

然而，驱动程序可以通过获取*ESP_PM_APB_FREQ_MAX* 类型的电源管理锁，防止系统改变 APB 频率。每当驱动创建以 RMT_CLK_SRC_APB 作为时钟源的 RMT 通道时，都会在通过*rmt_enable()* 启用通道后获取电源管理锁。反之，调用*rmt_disable()* 时，驱动程序释放锁。这也意味着*rmt_enable()* 和*rmt_disable()* 应成对出现。

如果将通道时钟源设置为其他选项，如*RMT_CLK_SRC_XTAL*，则驱动程序不会为其安装电源管理锁。对于低功耗应用程序来说，只要时钟源仍然可以提供足够的分辨率，不安装电源管理锁更为合适。

IRAM 安全 默认情况下，禁用 cache 时，写入/擦除主 flash 等原因将导致 RMT 中断延迟，事件回调函数也将延迟执行。在实时应用程序中，应避免此类情况。此外，当 RMT 事务依赖 **交替**中断连续编码或复制 RMT 符号时，上述中断延迟将导致不可预测的结果。

因此，可以启用 Kconfig 选项*CONFIG_RMT_ISR_IRAM_SAFE*，该选项：

1. 支持在禁用 cache 时启用所需中断
2. 支持将 ISR 使用的所有函数存放在 IRAM 中[2]
3. 支持将驱动程序实例存放在 DRAM 中，以防其意外映射到 PSRAM 中

启用该选项可以保证 cache 禁用时的中断运行，但会相应增加 IRAM 占用。

另外一个 Kconfig 选项*CONFIG_RMT_RECV_FUNC_IN_IRAM* 可以将*rmt_receive()* 函数放进内部的 IRAM 中，从而当 flash cache 被关闭的时候，这个函数也能够被使用。

线程安全 RMT 驱动程序会确保工厂函数*rmt_new_tx_channel()*、*rmt_new_rx_channel()* 和*rmt_new_sync_manager()* 的线程安全。使用时，可以直接从不同的 RTOS 任务中调用此类函数，无需额外锁保护。其他以*rmt_channel_handle_t* 和*rmt_sync_manager_handle_t* 作为第一个位置参数的函数均非线程安全，在没有设置互斥锁保护的任务中，应避免从多个任务中调用这类函数。

以下函数允许在 ISR 上下文中使用：

- *rmt_receive()*

Kconfig 选项

- *CONFIG_RMT_ISR_IRAM_SAFE* 控制默认 ISR 处理程序能否在禁用 cache 的情况下工作。详情请参阅*IRAM 安全*。
- *CONFIG_RMT_ENABLE_DEBUG_LOG* 用于启用调试日志输出，启用此选项将增加固件的二进制文件大小。
- *CONFIG_RMT_RECV_FUNC_IN_IRAM* 用于控制 RMT 接收函数被链接到系统存储的哪个位置（IRAM 还是 Flash）。详情请参阅*IRAM 安全*。

应用示例

- 基于 RMT 的 RGB LED 灯带自定义编码器：peripherals/rmt/led_strip
- RMT 红外 NEC 协议的编码与解码：peripherals/rmt/ir_nec_transceiver
- 队列中的 RMT 事务：peripherals/rmt/musical_buzzer
- 基于 RMT 的步进电机与 S 曲线算法：：peripherals/rmt/stepper_motor
- 用于驱动 DShot ESC 的 RMT 无限循环：peripherals/rmt/dshot_esc
- 模拟 1-wire 协议的 RMT 实现（以 DS18B20 为例）：peripherals/rmt/onewire

[2] 注意，回调函数（如*rmt_tx_event_callbacks_t::on_trans_done*）及回调函数所调用的函数也应位于 IRAM 中。

FAQ

- RMT 为什么会发送比预期更多的数据?
  RMT 的传输层编码是在 ISR 上下文中完成的，如果 RMT 编码耗时较长（例如，增加了过多的调试追踪信息），或者由于中断延迟和抢占导致编码工作被推迟执行，导致传输器在编码器更新内存数据之前就读取了老数据，致使传输器再次发送先前的数据。我们无法告诉硬件传输器自动等待新数据的更新，但是可以通过以下方法来缓解此问题：
  - 增加*rmt_tx_channel_config_t::mem_block_symbols* 的值，步长为 48。
  - 将编码函数放置在 IRAM 中。
  - 如果所用芯片支持*rmt_tx_channel_config_t::with_dma*，请启用该选项。
  - 将 RMT 驱动安装在另外一个 CPU 核上，避免和其他高中断频率的外设竞争同一个 CPU 资源。

API 参考

Header File

- components/esp_driver_rmt/include/driver/rmt_tx.h
- This header file can be included with:

```
#include "driver/rmt_tx.h"
```

- This header file is a part of the API provided by the `esp_driver_rmt` component. To declare that your component depends on `esp_driver_rmt`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_rmt
```

or

```
PRIV_REQUIRES esp_driver_rmt
```

Functions

esp_err_t **rmt_new_tx_channel**(const *rmt_tx_channel_config_t* *config, *rmt_channel_handle_t* *ret_chan)

Create a RMT TX channel.

参数

- **config** -- **[in]** TX channel configurations
- **ret_chan** -- **[out]** Returned generic RMT channel handle

返回

- ESP_OK: Create RMT TX channel successfully
- ESP_ERR_INVALID_ARG: Create RMT TX channel failed because of invalid argument
- ESP_ERR_NO_MEM: Create RMT TX channel failed because out of memory
- ESP_ERR_NOT_FOUND: Create RMT TX channel failed because all RMT channels are used up and no more free one
- ESP_ERR_NOT_SUPPORTED: Create RMT TX channel failed because some feature is not supported by hardware, e.g. DMA feature is not supported by hardware
- ESP_FAIL: Create RMT TX channel failed because of other error

esp_err_t **rmt_transmit**(*rmt_channel_handle_t* tx_channel, *rmt_encoder_handle_t* encoder, const void *payload, size_t payload_bytes, const *rmt_transmit_config_t* *config)

Transmit data by RMT TX channel.

备 注: This function constructs a transaction descriptor then pushes to a queue. The transaction will not start immediately if there's another one under processing. Based on the setting of *rmt_transmit_config_t::queue_nonblocking*, if there're too many transactions pending in the queue, this function can block until it has free slot, otherwise just return quickly.

备注： The payload data to be transmitted will be encoded into RMT symbols by the specific `encoder`.

备注： You CAN'T modify the `payload` during the transmission, it should be kept valid until the transmission is finished.

参数

- **`tx_channel`** -- **[in]** RMT TX channel that created by `rmt_new_tx_channel()`
- **`encoder`** -- **[in]** RMT encoder that created by various factory APIs like `rmt_new_bytes_encoder()`
- **`payload`** -- **[in]** The raw data to be encoded into RMT symbols
- **`payload_bytes`** -- **[in]** Size of the `payload` in bytes
- **`config`** -- **[in]** Transmission specific configuration

返回

- ESP_OK: Transmit data successfully
- ESP_ERR_INVALID_ARG: Transmit data failed because of invalid argument
- ESP_ERR_INVALID_STATE: Transmit data failed because channel is not enabled
- ESP_ERR_NOT_SUPPORTED: Transmit data failed because some feature is not supported by hardware, e.g. unsupported loop count
- ESP_FAIL: Transmit data failed because of other error

esp_err_t **`rmt_tx_wait_all_done`**(*rmt_channel_handle_t* tx_channel, int timeout_ms)

Wait for all pending TX transactions done.

备注： This function will block forever if the pending transaction can't be finished within a limited time (e.g. an infinite loop transaction). See also `rmt_disable()` for how to terminate a working channel.

参数

- **`tx_channel`** -- **[in]** RMT TX channel that created by `rmt_new_tx_channel()`
- **`timeout_ms`** -- **[in]** Wait timeout, in ms. Specially, -1 means to wait forever.

返回

- ESP_OK: Flush transactions successfully
- ESP_ERR_INVALID_ARG: Flush transactions failed because of invalid argument
- ESP_ERR_TIMEOUT: Flush transactions failed because of timeout
- ESP_FAIL: Flush transactions failed because of other error

esp_err_t **`rmt_tx_register_event_callbacks`**(*rmt_channel_handle_t* tx_channel, const *rmt_tx_event_callbacks_t* *cbs, void *user_data)

Set event callbacks for RMT TX channel.

备注： User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

备注： When CONFIG_RMT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. The `user_data` should also reside in SRAM.

参数

- **`tx_channel`** -- **[in]** RMT generic channel that created by `rmt_new_tx_channel()`
- **`cbs`** -- **[in]** Group of callback functions

- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

esp_err_t **rmt_new_sync_manager**(const *rmt_sync_manager_config_t* *config, *rmt_sync_manager_handle_t* *ret_synchro)

Create a synchronization manager for multiple TX channels, so that the managed channel can start transmitting at the same time.

备注: All the channels to be managed should be enabled by `rmt_enable()` before put them into sync manager.

参数

- **config** -- **[in]** Synchronization manager configuration
- **ret_synchro** -- **[out]** Returned synchronization manager handle

返回

- ESP_OK: Create sync manager successfully
- ESP_ERR_INVALID_ARG: Create sync manager failed because of invalid argument
- ESP_ERR_NOT_SUPPORTED: Create sync manager failed because it is not supported by hardware
- ESP_ERR_INVALID_STATE: Create sync manager failed because not all channels are enabled
- ESP_ERR_NO_MEM: Create sync manager failed because out of memory
- ESP_ERR_NOT_FOUND: Create sync manager failed because all sync controllers are used up and no more free one
- ESP_FAIL: Create sync manager failed because of other error

esp_err_t **rmt_del_sync_manager**(*rmt_sync_manager_handle_t* synchro)

Delete synchronization manager.

参数 **synchro** -- **[in]** Synchronization manager handle returned from `rmt_new_sync_manager()`

返回

- ESP_OK: Delete the synchronization manager successfully
- ESP_ERR_INVALID_ARG: Delete the synchronization manager failed because of invalid argument
- ESP_FAIL: Delete the synchronization manager failed because of other error

esp_err_t **rmt_sync_reset**(*rmt_sync_manager_handle_t* synchro)

Reset synchronization manager.

参数 **synchro** -- **[in]** Synchronization manager handle returned from `rmt_new_sync_manager()`

返回

- ESP_OK: Reset the synchronization manager successfully
- ESP_ERR_INVALID_ARG: Reset the synchronization manager failed because of invalid argument
- ESP_FAIL: Reset the synchronization manager failed because of other error

Structures

struct **rmt_tx_event_callbacks_t**

Group of RMT TX callbacks.

备注： The callbacks are all running under ISR environment

备注： When CONFIG_RMT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

Public Members

rmt_tx_done_callback_t **on_trans_done**

Event callback, invoked when transmission is finished

struct **rmt_tx_channel_config_t**

RMT TX channel specific configuration.

Public Members

gpio_num_t **gpio_num**

GPIO number used by RMT TX channel. Set to -1 if unused

rmt_clock_source_t **clk_src**

Clock source of RMT TX channel, channels in the same group must use the same clock source

uint32_t **resolution_hz**

Channel clock resolution, in Hz

size_t **mem_block_symbols**

Size of memory block, in number of *rmt_symbol_word_t*, must be an even. In the DMA mode, this field controls the DMA buffer size, it can be set to a large value; In the normal mode, this field controls the number of RMT memory block that will be used by the channel.

size_t **trans_queue_depth**

Depth of internal transfer queue, increase this value can support more transfers pending in the background

int **intr_priority**

RMT interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **invert_out**

Whether to invert the RMT channel signal before output to GPIO pad

uint32_t **with_dma**

If set, the driver will allocate an RMT channel with DMA capability

uint32_t **io_loop_back**

The signal output from the GPIO will be fed to the input path as well

uint32_t **io_od_mode**

Configure the GPIO as open-drain mode

struct *rmt_tx_channel_config_t*::[anonymous] **flags**

TX channel config flags

struct **rmt_transmit_config_t**

RMT transmit specific configuration.

Public Members

int **loop_count**

Specify the times of transmission in a loop, -1 means transmitting in an infinite loop

uint32_t **eot_level**

Set the output level for the ”End Of Transmission”

uint32_t **queue_nonblocking**

If set, when the transaction queue is full, driver will not block the thread but return directly

struct *rmt_transmit_config_t*::[anonymous] **flags**

Transmit specific config flags

struct **rmt_sync_manager_config_t**

Synchronous manager configuration.

Public Members

const *rmt_channel_handle_t* ***tx_channel_array**

Array of TX channels that are about to be managed by a synchronous controller

size_t **array_size**

Size of the `tx_channel_array`

Header File

- components/esp_driver_rmt/include/driver/rmt_rx.h
- This header file can be included with:

```
#include "driver/rmt_rx.h"
```

- This header file is a part of the API provided by the `esp_driver_rmt` component. To declare that your component depends on `esp_driver_rmt`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_rmt
```

or

```
PRIV_REQUIRES esp_driver_rmt
```

Functions

esp_err_t **rmt_new_rx_channel**(const *rmt_rx_channel_config_t* *config, *rmt_channel_handle_t* *ret_chan)

Create a RMT RX channel.

参数

- **config** -- **[in]** RX channel configurations
- **ret_chan** -- **[out]** Returned generic RMT channel handle

返回

- ESP_OK: Create RMT RX channel successfully
- ESP_ERR_INVALID_ARG: Create RMT RX channel failed because of invalid argument
- ESP_ERR_NO_MEM: Create RMT RX channel failed because out of memory
- ESP_ERR_NOT_FOUND: Create RMT RX channel failed because all RMT channels are used up and no more free one
- ESP_ERR_NOT_SUPPORTED: Create RMT RX channel failed because some feature is not supported by hardware, e.g. DMA feature is not supported by hardware
- ESP_FAIL: Create RMT RX channel failed because of other error

esp_err_t **rmt_receive**(*rmt_channel_handle_t* rx_channel, void *buffer, size_t buffer_size, const *rmt_receive_config_t* *config)

Initiate a receive job for RMT RX channel.

备注: This function is non-blocking, it initiates a new receive job and then returns. User should check the received data from the `on_recv_done` callback that registered by `rmt_rx_register_event_callbacks()`.

备注: This function can also be called in ISR context.

备注: If you want this function to work even when the flash cache is disabled, please enable the `CONFIG_RMT_RECV_FUNC_IN_IRAM` option.

参数

- **rx_channel** -- **[in]** RMT RX channel that created by `rmt_new_rx_channel()`
- **buffer** -- **[in]** The buffer to store the received RMT symbols
- **buffer_size** -- **[in]** size of the `buffer`, in bytes
- **config** -- **[in]** Receive specific configurations

返回

- ESP_OK: Initiate receive job successfully
- ESP_ERR_INVALID_ARG: Initiate receive job failed because of invalid argument
- ESP_ERR_INVALID_STATE: Initiate receive job failed because channel is not enabled
- ESP_FAIL: Initiate receive job failed because of other error

esp_err_t **rmt_rx_register_event_callbacks**(*rmt_channel_handle_t* rx_channel, const *rmt_rx_event_callbacks_t* *cbs, void *user_data)

Set callbacks for RMT RX channel.

备注: User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

备注: When CONFIG_RMT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. The `user_data` should also reside in SRAM.

参数

- **rx_channel** -- **[in]** RMT generic channel that created by `rmt_new_rx_channel()`
- **cbs** -- **[in]** Group of callback functions
- **user_data** -- **[in]** User data, which will be passed to callback functions directly

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

Structures

struct **rmt_rx_event_callbacks_t**

Group of RMT RX callbacks.

备注： The callbacks are all running under ISR environment

备注： When CONFIG_RMT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

Public Members

rmt_rx_done_callback_t **on_recv_done**

Event callback, invoked when the RMT channel reception is finished or partial data is received

struct **rmt_rx_channel_config_t**

RMT RX channel specific configuration.

Public Members

gpio_num_t **gpio_num**

GPIO number used by RMT RX channel. Set to -1 if unused

rmt_clock_source_t **clk_src**

Clock source of RMT RX channel, channels in the same group must use the same clock source

uint32_t **resolution_hz**

Channel clock resolution, in Hz

size_t **mem_block_symbols**

Size of memory block, in number of *rmt_symbol_word_t*, must be an even. In the DMA mode, this field controls the DMA buffer size, it can be set to a large value (e.g. 1024); In the normal mode, this field controls the number of RMT memory block that will be used by the channel.

int **intr_priority**

RMT interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **invert_in**

Whether to invert the incoming RMT channel signal

uint32_t **with_dma**

If set, the driver will allocate an RMT channel with DMA capability

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

struct *rmt_rx_channel_config_t*::[anonymous] **flags**

RX channel config flags

struct **rmt_receive_config_t**

RMT receive specific configuration.

Public Members

uint32_t **signal_range_min_ns**

A pulse whose width is smaller than this threshold will be treated as glitch and ignored

uint32_t **signal_range_max_ns**

RMT will stop receiving if one symbol level has kept more than `signal_range_max_ns`

struct *rmt_receive_config_t*::*extra_rmt_receive_flags* **flags**

Receive specific config flags

struct **extra_rmt_receive_flags**

Receive specific flags.

Public Members

uint32_t **en_partial_rx**

Set this flag if the incoming data is very long, and the driver can only receive the data piece by piece, because the user buffer is not sufficient to save all the data.

Header File

- components/esp_driver_rmt/include/driver/rmt_common.h
- This header file can be included with:

```
#include "driver/rmt_common.h"
```

- This header file is a part of the API provided by the `esp_driver_rmt` component. To declare that your component depends on `esp_driver_rmt`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_rmt
```

or

```
PRIV_REQUIRES esp_driver_rmt
```

Functions

esp_err_t **rmt_del_channel**(*rmt_channel_handle_t* channel)

Delete an RMT channel.

参数 **channel** -- **[in]** RMT generic channel that created by `rmt_new_tx_channel()` or `rmt_new_rx_channel()`

返回

- ESP_OK: Delete RMT channel successfully
- ESP_ERR_INVALID_ARG: Delete RMT channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete RMT channel failed because it is still in working
- ESP_FAIL: Delete RMT channel failed because of other error

esp_err_t **rmt_apply_carrier**(*rmt_channel_handle_t* channel, const *rmt_carrier_config_t* *config)

Apply modulation feature for TX channel or demodulation feature for RX channel.

参数

- **channel** -- **[in]** RMT generic channel that created by `rmt_new_tx_channel()` or `rmt_new_rx_channel()`
- **config** -- **[in]** Carrier configuration. Specially, a NULL config means to disable the carrier modulation or demodulation feature

返回

- ESP_OK: Apply carrier configuration successfully
- ESP_ERR_INVALID_ARG: Apply carrier configuration failed because of invalid argument
- ESP_FAIL: Apply carrier configuration failed because of other error

esp_err_t **rmt_enable**(*rmt_channel_handle_t* channel)

Enable the RMT channel.

备注: This function will acquire a PM lock that might be installed during channel allocation

参数 **channel** -- **[in]** RMT generic channel that created by `rmt_new_tx_channel()` or `rmt_new_rx_channel()`

返回

- ESP_OK: Enable RMT channel successfully
- ESP_ERR_INVALID_ARG: Enable RMT channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable RMT channel failed because it's enabled already
- ESP_FAIL: Enable RMT channel failed because of other error

esp_err_t **rmt_disable**(*rmt_channel_handle_t* channel)

Disable the RMT channel.

备注: This function will release a PM lock that might be installed during channel allocation

参数 **channel** -- **[in]** RMT generic channel that created by `rmt_new_tx_channel()` or `rmt_new_rx_channel()`

返回

- ESP_OK: Disable RMT channel successfully
- ESP_ERR_INVALID_ARG: Disable RMT channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable RMT channel failed because it's not enabled yet
- ESP_FAIL: Disable RMT channel failed because of other error

Structures

struct **rmt_carrier_config_t**

RMT carrier wave configuration (for either modulation or demodulation)

Public Members

uint32_t **frequency_hz**

Carrier wave frequency, in Hz, 0 means disabling the carrier

float **duty_cycle**

Carrier wave duty cycle (0~100%)

uint32_t **polarity_active_low**

Specify the polarity of carrier, by default it's modulated to base signal's high level

uint32_t **always_on**

If set, the carrier can always exist even there's not transfer undergoing

struct *rmt_carrier_config_t*::[anonymous] **flags**

Carrier config flags

Header File

- components/esp_driver_rmt/include/driver/rmt_encoder.h
- This header file can be included with:

```
#include "driver/rmt_encoder.h"
```

- This header file is a part of the API provided by the `esp_driver_rmt` component. To declare that your component depends on `esp_driver_rmt`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_rmt
```

or

```
PRIV_REQUIRES esp_driver_rmt
```

Functions

esp_err_t **rmt_new_bytes_encoder**(const *rmt_bytes_encoder_config_t* *config, *rmt_encoder_handle_t* *ret_encoder)

Create RMT bytes encoder, which can encode byte stream into RMT symbols.

参数

- **config** -- **[in]** Bytes encoder configuration
- **ret_encoder** -- **[out]** Returned encoder handle

返回

- ESP_OK: Create RMT bytes encoder successfully
- ESP_ERR_INVALID_ARG: Create RMT bytes encoder failed because of invalid argument
- ESP_ERR_NO_MEM: Create RMT bytes encoder failed because out of memory
- ESP_FAIL: Create RMT bytes encoder failed because of other error

esp_err_t **rmt_bytes_encoder_update_config**(*rmt_encoder_handle_t* bytes_encoder, const *rmt_bytes_encoder_config_t* *config)

Update the configuration of the bytes encoder.

备注: The configurations of the bytes encoder is also set up by `rmt_new_bytes_encoder()`. This function is used to update the configuration of the bytes encoder at runtime.

参数

- **bytes_encoder** -- **[in]** Bytes encoder handle, created by e.g `rmt_new_bytes_encoder()`
- **config** -- **[in]** Bytes encoder configuration

返回

- ESP_OK: Update RMT bytes encoder successfully
- ESP_ERR_INVALID_ARG: Update RMT bytes encoder failed because of invalid argument
- ESP_FAIL: Update RMT bytes encoder failed because of other error

esp_err_t **rmt_new_copy_encoder**(const *rmt_copy_encoder_config_t* *config, *rmt_encoder_handle_t* *ret_encoder)

Create RMT copy encoder, which copies the given RMT symbols into RMT memory.

参数

- **config** -- **[in]** Copy encoder configuration
- **ret_encoder** -- **[out]** Returned encoder handle

返回

- ESP_OK: Create RMT copy encoder successfully
- ESP_ERR_INVALID_ARG: Create RMT copy encoder failed because of invalid argument
- ESP_ERR_NO_MEM: Create RMT copy encoder failed because out of memory
- ESP_FAIL: Create RMT copy encoder failed because of other error

esp_err_t **rmt_new_simple_encoder**(const *rmt_simple_encoder_config_t* *config, *rmt_encoder_handle_t* *ret_encoder)

Create RMT simple callback encoder, which uses a callback to convert incoming data into RMT symbols.

参数

- **config** -- **[in]** Simple callback encoder configuration
- **ret_encoder** -- **[out]** Returned encoder handle

返回

- ESP_OK: Create RMT simple callback encoder successfully
- ESP_ERR_INVALID_ARG: Create RMT simple callback encoder failed because of invalid argument
- ESP_ERR_NO_MEM: Create RMT simple callback encoder failed because out of memory
- ESP_FAIL: Create RMT simple callback encoder failed because of other error

esp_err_t **rmt_del_encoder**(*rmt_encoder_handle_t* encoder)

Delete RMT encoder.

参数 **encoder** -- **[in]** RMT encoder handle, created by e.g `rmt_new_bytes_encoder()`

返回

- ESP_OK: Delete RMT encoder successfully
- ESP_ERR_INVALID_ARG: Delete RMT encoder failed because of invalid argument
- ESP_FAIL: Delete RMT encoder failed because of other error

esp_err_t **rmt_encoder_reset**(*rmt_encoder_handle_t* encoder)

Reset RMT encoder.

参数 **encoder** -- **[in]** RMT encoder handle, created by e.g `rmt_new_bytes_encoder()`

返回

- ESP_OK: Reset RMT encoder successfully
- ESP_ERR_INVALID_ARG: Reset RMT encoder failed because of invalid argument
- ESP_FAIL: Reset RMT encoder failed because of other error

void ***rmt_alloc_encoder_mem**(size_t size)

A helper function to allocate a proper memory for RMT encoder.

参数 **size** -- Size of memory to be allocated

返回 Pointer to the allocated memory if the allocation is successful, NULL otherwise

Structures

struct **rmt_encoder_t**

Interface of RMT encoder.

Public Members

size_t (***encode**)(*rmt_encoder_t* *encoder, *rmt_channel_handle_t* tx_channel, const void *primary_data, size_t data_size, *rmt_encode_state_t* *ret_state)

Encode the user data into RMT symbols and write into RMT memory.

备注: The encoding function will also be called from an ISR context, thus the function must not call any blocking API.

备注: It's recommended to put this function implementation in the IRAM, to achieve a high performance and less interrupt latency.

Param encoder **[in]** Encoder handle
Param tx_channel **[in]** RMT TX channel handle, returned from `rmt_new_tx_channel()`
Param primary_data **[in]** App data to be encoded into RMT symbols
Param data_size **[in]** Size of primary_data, in bytes
Param ret_state **[out]** Returned current encoder's state
Return Number of RMT symbols that the primary data has been encoded into

esp_err_t (***reset**)(*rmt_encoder_t* *encoder)

Reset encoding state.

Param encoder **[in]** Encoder handle
Return
- ESP_OK: reset encoder successfully
- ESP_FAIL: reset encoder failed

esp_err_t (***del**)(*rmt_encoder_t* *encoder)

Delete encoder object.

Param encoder **[in]** Encoder handle
Return
- ESP_OK: delete encoder successfully
- ESP_FAIL: delete encoder failed

struct **rmt_bytes_encoder_config_t**

Bytes encoder configuration.

Public Members

rmt_symbol_word_t **bit0**

How to represent BIT0 in RMT symbol

rmt_symbol_word_t **bit1**

How to represent BIT1 in RMT symbol

uint32_t **msb_first**

Whether to encode MSB bit first

struct *rmt_bytes_encoder_config_t*::[anonymous] **flags**

Encoder config flag

struct **rmt_copy_encoder_config_t**

Copy encoder configuration.

struct **rmt_simple_encoder_config_t**

Simple callback encoder configuration.

Public Members

rmt_encode_simple_cb_t **callback**

Callback to call for encoding data into RMT items

void ***arg**

Opaque user-supplied argument for callback

size_t **min_chunk_size**

Minimum amount of free space, in RMT symbols, the encoder needs in order to guarantee it always returns non-zero. Defaults to 64 if zero / not given.

Type Definitions

typedef size_t (***rmt_encode_simple_cb_t**)(const void *data, size_t data_size, size_t symbols_written, size_t symbols_free, *rmt_symbol_word_t* *symbols, bool *done, void *arg)

Callback for simple callback encoder.

This will get called to encode the data stream of given length (as passed to rmt_transmit by the user) into symbols to be sent by the hardware.

The callback will be initially called with symbol_pos=0. If the callback encodes N symbols and finishes, the next callback will always be with symbols_written=N. If the callback then encodes M symbols, the next callback will always be with symbol_pos=N+M, etc. The only exception is when the encoder is reset (e.g. to begin a new transaction) in which case symbol_pos will always restart at 0.

If the amount of free space in the symbol buffer (as indicated by symbols_free) is too low, the function can return 0 as result and the RMT will call the function again once there is more space available. Note that the callback should eventually return non-0 if called with free space of *rmt_simple_encoder_config_t::min_chunk_size* or more. It is acceptable to return 0 for a given free space N, then on the next call (possibly with a larger free buffer space) return less or more than N symbols.

When the transaction is done (all data_size data is encoded), the callback can indicate this by setting *done to true. This can either happen on the last callback call that returns an amount of symbols encoded, or on a callback that returns zero. In either case, the callback will not be called again for this transaction.

Param data **[in]** Data pointer, as passed to rmt_transmit()
Param data_size **[in]** Size of the data, as passed to rmt_transmit()
Param symbols_written **[in]** Current position in encoded stream, in symbols

Param symbols_free [in] The maximum amount of symbols that can be written into the `symbols` buffer
Param symbols [out] Symbols to be sent to the RMT hardware
Param done [out] Setting this to true marks this transaction as finished
Param arg Opaque argument
Return Amount of symbols encoded in this callback round. 0 if more space is needed.

Enumerations

enum **rmt_encode_state_t**

RMT encoding state.

Values:

enumerator **RMT_ENCODING_RESET**

The encoding session is in reset state

enumerator **RMT_ENCODING_COMPLETE**

The encoding session is finished, the caller can continue with subsequent encoding

enumerator **RMT_ENCODING_MEM_FULL**

The encoding artifact memory is full, the caller should return from current encoding session

Header File

- components/esp_driver_rmt/include/driver/rmt_types.h
- This header file can be included with:

```
#include "driver/rmt_types.h"
```

- This header file is a part of the API provided by the `esp_driver_rmt` component. To declare that your component depends on `esp_driver_rmt`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_rmt
```

or

```
PRIV_REQUIRES esp_driver_rmt
```

Structures

struct **rmt_tx_done_event_data_t**

Type of RMT TX done event data.

Public Members

size_t **num_symbols**

The number of transmitted RMT symbols, including one EOF symbol, which is appended by the driver to mark the end of a transmission. For a loop transmission, this value only counts for one round.

struct **rmt_rx_done_event_data_t**

Type of RMT RX done event data.

Public Members

rmt_symbol_word_t ***received_symbols**

Point to the received RMT symbols

size_t **num_symbols**

The number of received RMT symbols

uint32_t **is_last**

Indicating if the current received data are the last part of the transaction

struct *rmt_rx_done_event_data_t*::[anonymous] **flags**

Extra flags

Type Definitions

typedef struct rmt_channel_t ***rmt_channel_handle_t**

Type of RMT channel handle.

typedef struct rmt_sync_manager_t ***rmt_sync_manager_handle_t**

Type of RMT synchronization manager handle.

typedef struct *rmt_encoder_t* ***rmt_encoder_handle_t**

Type of RMT encoder handle.

typedef bool (***rmt_tx_done_callback_t**)(*rmt_channel_handle_t* tx_chan, const *rmt_tx_done_event_data_t* *edata, void *user_ctx)

Prototype of RMT event callback.

Param tx_chan **[in]** RMT channel handle, created from `rmt_new_tx_channel()`
Param edata **[in]** Point to RMT event data. The lifecycle of this pointer memory is inside this function, user should copy it into static memory if used outside this function.
Param user_ctx **[in]** User registered context, passed from `rmt_tx_register_event_callbacks()`
Return Whether a high priority task has been waken up by this callback function

typedef bool (***rmt_rx_done_callback_t**)(*rmt_channel_handle_t* rx_chan, const *rmt_rx_done_event_data_t* *edata, void *user_ctx)

Prototype of RMT event callback.

Param rx_chan **[in]** RMT channel handle, created from `rmt_new_rx_channel()`
Param edata **[in]** Point to RMT event data. The lifecycle of this pointer memory is inside this function, user should copy it into static memory if used outside this function.
Param user_ctx **[in]** User registered context, passed from `rmt_rx_register_event_callbacks()`
Return Whether a high priority task has been waken up by this function

Header File

- components/hal/include/hal/rmt_types.h
- This header file can be included with:

```
#include "hal/rmt_types.h"
```

Unions

union **rmt_symbol_word_t**

#include <rmt_types.h> The layout of RMT symbol stored in memory, which is decided by the hardware design.

Public Members

uint16_t **duration0**

Duration of level0

uint16_t **level0**

Level of the first part

uint16_t **duration1**

Duration of level1

uint16_t **level1**

Level of the second part

struct *rmt_symbol_word_t*::[anonymous] **[anonymous]**

uint32_t **val**

Equivalent unsigned value for the RMT symbol

Type Definitions

typedef *soc_periph_rmt_clk_src_t* **rmt_clock_source_t**

RMT group clock source.

备注： User should select the clock source based on the power and resolution requirement

警告： This document is not updated for ESP32P4 yet, so some of the content may not be correct.

This warning was automatically inserted due to the source file being in the *add_warnings_pages* list.

2.5.24 SD Pull-up Requirements

Espressif hardware products are designed for multiple use cases which may require different pull states on pins. For this reason, the pull state of particular pins on certain products needs to be adjusted to provide the pull-ups required in the SD bus.

SD pull-up requirements apply to cases where ESP32-P4 uses the SPI or SDMMC controller to communicate with SD cards. When an SD card is operating in SPI mode or 1-bit SD mode, the CMD and DATA (DAT0 - DAT3) lines of the SD bus must be pulled up by 10 kOhm resistors. SD cards and SDIO devices should also have pull-ups on all above-mentioned lines (regardless of whether these lines are connected to the host) in order to prevent them from entering a wrong state.

This document has the following structure:

- *Overview of compatibility* between the default pull states on pins of Espressif's products and the states required by the SD bus
- *Solutions* - ideas on how to resolve compatibility issues

- *Related information* - other relevant information

Overview of Compatibility

This section provides an overview of compatibility issues that might occur when using SDIO (secure digital input output). Since the SD bus needs to be connected to pull-ups, these issues should be resolved regardless of whether they are related to master (host) or slave (device). Each issue has links to its respective solution. A solution for a host and device may differ.

Systems on a Chip (SoCs) ESP32-P4 SDMMC host controller allows using any of GPIOs for any of SD interface signals. However, it is recommended to avoid using strapping GPIOs, GPIOs with internal weak pull-downs and GPIOs commonly used for other purposes to prevent conflicts:

Systems in Packages (SIP)

Modules

Development Boards

Solutions

No Pull-ups If you use a development board without pull-ups, you can do the following:

- If your host and slave device are on separate boards, replace one of them with a board that has pull-ups. For the list of Espressif's development boards with pull-ups, go to *Development Boards*.
- Attach external pull-ups by connecting each pin which requires a pull-up to VDD via a 10 kOhm resistor.

Related Information

2.5.25 SDMMC 主机驱动

概述

ESP32-P4 的 SDMMC 主机外设共有两个卡槽，用于插入 SD 卡、连接 SDIO 设备或连接 eMMC 芯片，每个卡槽均可单独使用。

卡槽 `SDMMC_HOST_SLOT_0` 和 `SDMMC_HOST_SLOT_1` 都支持 1、4、8 线的 SD 接口，这些卡槽通过 GPIO 交换矩阵连接到 ESP32-P4 的 GPIO，即每个 SD 卡信号都可以使用任意 GPIO 连接。

- 卡槽 `SDMMC_HOST_SLOT_1` 通过 GPIO 矩阵连接。这意味着任何 GPIO 都可以用于每个 SD 卡信号。它适用于非 UHS-I 用途。
- 卡槽 `SDMMC_HOST_SLOT_0` 专用于 UHS-I 模式，驱动程序中尚不支持该模式。

目前 SDMMC 主机需要为 IO 电平提供外部电压参考。如果您自行购买 ESP32P4 芯片并计划使用 SDMMC 外设，请参阅:ref:'wr-ctrl' 。

支持的速率模式

SDMMC 主机驱动支持以下速率模式：

- 默认速率 (20 MHz)：对于 SD 卡，支持 1 线或 4 线传输；对于 3.3 V eMMC，支持 1 线、4 线或 8 线传输。
- 高速模式 (40 MHz)：对于 SD 卡，支持 1 线或 4 线传输；对于 3.3 V eMMC，支持 1 线、4 线或 8 线传输。
- 高速 DDR 模式 (40 MHz)：对于 3.3 V eMMC，支持 4 线传输。

当前尚不支持的速率模式：

- 高速 DDR 模式：不支持 8 线 eMMC 传输
- UHS-I 1.8 V 模式：不支持 4 线 SD 卡传输

使用 SDMMC 主机驱动

在大多数应用程序中，只有下列函数会被直接调用：

- *sdmmc_host_init()*
- *sdmmc_host_init_slot()*
- *sdmmc_host_deinit()*

其他函数将通过 sdmmc_host_t 结构体中的函数指针由 SD/MMC 协议层调用，例如：

- *sdmmc_host_set_bus_width()*
- *sdmmc_host_set_card_clk()*
- *sdmmc_host_do_transaction()*

配置总线宽度和频率

使用 sdmmc_host_t 和*sdmmc_slot_config_t* 的默认初始化配置，即 SDMMC_HOST_DEFAULT 和 SDMMC_SLOT_CONFIG_DEFAULT 时，SDMMC 主机驱动会尝试以当前卡所支持的最大总线宽度进行通信（SD 卡为 4 线，eMMC 为 8 线），并使用 20 MHz 的通信频率。

在支持 40 MHz 频率通信的设计中，可以调整 sdmmc_host_t 结构体中的 max_freq_khz 字段，提升总线频率：

```
sdmmc_host_t host = SDMMC_HOST_DEFAULT();
host.max_freq_khz = SDMMC_FREQ_HIGHSPEED;
```

如需选择标准速率以外的特定频率，请根据所使用的 SD 接口（SDMMC 或 SDSPI）确定适当频率范围，并选择其中的任意值。然而，实际的时钟频率会由底层驱动程序计算，可能与你所需的值不同。

使用 SDMMC 接口时，max_freq_khz 即频率上限，因此最终的频率值应始终低于该上限。而使用 SDSPI 接口时，驱动程序会提供最接近的适配频率，因此该值可以大于、等于或小于 max_freq_khz。

请配置*sdmmc_slot_config_t* 的 width 字段，配置总线宽度。例如，配置 1 线模式的代码如下：

```
sdmmc_slot_config_t slot = SDMMC_SLOT_CONFIG_DEFAULT();
slot.width = 1;
```

配置 GPIO

通过配置结构体*sdmmc_slot_config_t*，ESP32-P4 的 SDMMC 主机可以根据需要，为每个信号配置任意的 GPIO 管脚。

例如，使用以下代码，可以将 GPIO 1-6 分别用于 CLK、CMD、D0-D3 信号：

```
sdmmc_slot_config_t slot = SDMMC_SLOT_CONFIG_DEFAULT();
slot.clk = GPIO_NUM_1;
slot.cmd = GPIO_NUM_2;
slot.d0 = GPIO_NUM_3;
slot.d1 = GPIO_NUM_4;
slot.d2 = GPIO_NUM_5;
slot.d3 = GPIO_NUM_6;
```

也可以配置 CD 和 WP 管脚。与配置其他信号的方法类似，你只需配置相同结构体的 `cd` 和 `wp` 参数：

```
slot.cd = GPIO_NUM_7;
slot.wp = GPIO_NUM_8;
```

`SDMMC_SLOT_CONFIG_DEFAULT` 将 CD 和 WP 管脚都配置为 `GPIO_NUM_NC`，表明默认情况下不会使用这两个管脚。

通过上述方式初始化*sdmmc_slot_config_t* 结构体后，即可在调用*sdmmc_host_init_slot()* 或其他任意高层函数（如*esp_vfs_fat_sdmmc_mount()*）时使用该结构体。

配置供电和参考电压

ESP32-P4 SDMMC 主机需要为 IO 电平提供外部电压参考，以支持高速设备，驱动器将动态配置电压参考。您可以使用片上可编程 LDO 作为从机电源和电压参考，也可以提供正确的外部电源。

- 要使用片上 LDO，请确保 VDDPST_5(sd_vref) 引脚连接到所选的片上 LD 通道，并调用:cpp:func:'sd_pwr_ctrl_new_on_chip_ldo' 分配所选的 LDO 通道，然后将'pwr_ctr_handle' 传递给:cpp:class:'sdmmc_host_t::pwr_ctl_handle'。
- 要使用外部电源，请确保 VDDPST_5(sd_vref) 引脚已连接，然后按照:cpp:class:'sd_pwr_ctrl_drv_t' 构造外部电源控制结构体，并将其传递给:cpp:class:'sdmmc_host_t::pwr_ctr_handle'。

eMMC 芯片的 DDR 模式

默认情况下，如果满足以下条件，将使用 DDR 模式：

- 在 `sdmmc_host_t` 结构体中将 SDMMC 主机频率配置为 `SDMMC_FREQ_HIGHSPEED`，且
- eMMC 芯片在其 CSD 寄存器中报告支持 DDR 模式

DDR 模式对信号完整性要求更高。如果要在保持 `SDMMC_FREQ_HIGHSPEED` 频率的同时禁用 DDR 模式，请在 `sdmmc_host_t` 结构体的 `sdmmc_host_t::flags` 字段中清除 `SDMMC_HOST_FLAG_DDR` 位：

```
sdmmc_host_t host = SDMMC_HOST_DEFAULT();
host.max_freq_khz = SDMMC_FREQ_HIGHSPEED;
host.flags &= ~SDMMC_HOST_FLAG_DDR;
```

相关文档

- *SD/SDIO/MMC 驱动程序*：介绍了实现协议层的高层驱动程序。
- *SD SPI 主机驱动程序*：介绍了一种类似驱动，该驱动使用 SPI 控制器且受限于 SD 协议的 SPI 模式。
- *SD Pull-up Requirements* 介绍了模组和开发套件上的上拉支持和兼容信息。

API 参考

Header File

- components/esp_driver_sdmmc/include/driver/sdmmc_host.h
- This header file can be included with:

```
#include "driver/sdmmc_host.h"
```

- This header file is a part of the API provided by the `esp_driver_sdmmc` component. To declare that your component depends on `esp_driver_sdmmc`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_sdmmc
```

or

```
PRIV_REQUIRES esp_driver_sdmmc
```

Functions

esp_err_t **sdmmc_host_init**(void)

Initialize SDMMC host peripheral.

备注： This function is not thread safe

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdmmc_host_init was already called
- ESP_ERR_NO_MEM if memory can not be allocated

esp_err_t **sdmmc_host_init_slot**(int slot, const *sdmmc_slot_config_t* *slot_config)

Initialize given slot of SDMMC peripheral.

On the ESP32, SDMMC peripheral has two slots:

- Slot 0: 8-bit wide, maps to HS1_* signals in PIN MUX
- Slot 1: 4-bit wide, maps to HS2_* signals in PIN MUX

Card detect and write protect signals can be routed to arbitrary GPIOs using GPIO matrix.

备注： This function is not thread safe

参数

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **slot_config** -- additional configuration for the slot

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if host has not been initialized using sdmmc_host_init

esp_err_t **sdmmc_host_set_bus_width**(int slot, size_t width)

Select bus width to be used for data transfer.

SD/MMC card must be initialized prior to this command, and a command to set bus width has to be sent to the card (e.g. SD_APP_SET_BUS_WIDTH)

备注： This function is not thread safe

参数

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **width** -- bus width (1, 4, or 8 for slot 0; 1 or 4 for slot 1)

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if slot number or width is not valid

size_t **sdmmc_host_get_slot_width**(int slot)

Get bus width configured in `sdmmc_host_init_slot` to be used for data transfer.

参数 **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
返回 configured bus width of the specified slot.

esp_err_t **sdmmc_host_set_card_clk**(int slot, uint32_t freq_khz)

Set card clock frequency.

Currently only integer fractions of 40MHz clock can be used. For High Speed cards, 40MHz can be used. For Default Speed cards, 20MHz can be used.

备注: This function is not thread safe

参数

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **freq_khz** -- card clock frequency, in kHz

返回

- ESP_OK on success
- other error codes may be returned in the future

esp_err_t **sdmmc_host_set_bus_ddr_mode**(int slot, bool ddr_enabled)

Enable or disable DDR mode of SD interface.

参数

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **ddr_enabled** -- enable or disable DDR mode

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if DDR mode is not supported on this slot

esp_err_t **sdmmc_host_set_cclk_always_on**(int slot, bool cclk_always_on)

Enable or disable always-on card clock When cclk_always_on is false, the host controller is allowed to shut down the card clock between the commands. When cclk_always_on is true, the clock is generated even if no command is in progress.

参数

- **slot** -- slot number
- **cclk_always_on** -- enable or disable always-on clock

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the slot number is invalid

esp_err_t **sdmmc_host_do_transaction**(int slot, sdmmc_command_t *cmdinfo)

Send command to the card and get response.

This function returns when command is sent and response is received, or data is transferred, or timeout occurs.

Attention Data buffer passed in cmdinfo->data must be in DMA capable memory and aligned to 4 byte boundary. If it's behind the cache, both cmdinfo->data and cmdinfo->buflen need to be aligned to cache line boundary.

备注: This function is not thread safe w.r.t. init/deinit functions, and bus width/clock speed configuration functions. Multiple tasks can call sdmmc_host_do_transaction as long as other sdmmc_host_* functions are not called.

参数

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **cmdinfo** -- pointer to structure describing command and data to transfer

返回

- ESP_OK on success
- ESP_ERR_TIMEOUT if response or data transfer has timed out
- ESP_ERR_INVALID_CRC if response or data transfer CRC check has failed
- ESP_ERR_INVALID_RESPONSE if the card has sent an invalid response
- ESP_ERR_INVALID_SIZE if the size of data transfer is not valid in SD protocol
- ESP_ERR_INVALID_ARG if the data buffer is not in DMA capable memory

esp_err_t **sdmmc_host_io_int_enable**(int slot)

Enable IO interrupts.

This function configures the host to accept SDIO interrupts.

参数 **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
返回 returns ESP_OK, other errors possible in the future

esp_err_t **sdmmc_host_io_int_wait**(int slot, TickType_t timeout_ticks)

Block until an SDIO interrupt is received, or timeout occurs.

参数

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **timeout_ticks** -- number of RTOS ticks to wait for the interrupt

返回

- ESP_OK on success (interrupt received)
- ESP_ERR_TIMEOUT if the interrupt did not occur within timeout_ticks

esp_err_t **sdmmc_host_deinit**(void)

Disable SDMMC host and release allocated resources.

备注: This function is not thread safe

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdmmc_host_init function has not been called

esp_err_t **sdmmc_host_get_real_freq**(int slot, int *real_freq_khz)

Provides a real frequency used for an SD card installed on specific slot of SD/MMC host controller.

This function calculates real working frequency given by current SD/MMC host controller setup for required slot: it reads associated host and card dividers from corresponding SDMMC registers, calculates respective frequency and stores the value into the 'real_freq_khz' parameter

参数

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **real_freq_khz** -- **[out]** output parameter for the result frequency (in kHz)

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG on real_freq_khz == NULL or invalid slot number used

esp_err_t **sdmmc_host_set_input_delay**(int slot, sdmmc_delay_phase_t delay_phase)

set input delay

- This API sets delay when the SDMMC Host samples the signal from the SD Slave.
- This API will check if the given `delay_phase` is valid or not.
- This API will print out the delay time, in picosecond (ps)

备注： ESP32 doesn't support this feature, you will get an `ESP_ERR_NOT_SUPPORTED`

参数

- **`slot`** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **`delay_phase`** -- delay phase, this API will convert the phase into picoseconds and print it out

返回

- ESP_OK: ON success.
- ESP_ERR_INVALID_ARG: Invalid argument.
- ESP_ERR_NOT_SUPPORTED: ESP32 doesn't support this feature.

esp_err_t **`sdmmc_host_get_dma_info`**(int slot, esp_dma_mem_info_t *dma_mem_info)

Get the DMA memory information for the host driver.

参数

- **`slot`** -- **[in]** slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **`dma_mem_info`** -- **[out]** DMA memory information structure

返回

- ESP_OK: ON success.
- ESP_ERR_INVALID_ARG: Invalid argument.

Structures

struct **`sdmmc_slot_config_t`**

Extra configuration for SDMMC peripheral slot

Public Members

gpio_num_t **`clk`**

GPIO number of CLK signal.

gpio_num_t **`cmd`**

GPIO number of CMD signal.

gpio_num_t **`d0`**

GPIO number of D0 signal.

gpio_num_t **`d1`**

GPIO number of D1 signal.

gpio_num_t **`d2`**

GPIO number of D2 signal.

gpio_num_t **`d3`**

GPIO number of D3 signal.

gpio_num_t **`d4`**

GPIO number of D4 signal. Ignored in 1- or 4- line mode.

gpio_num_t **d5**

GPIO number of D5 signal. Ignored in 1- or 4- line mode.

gpio_num_t **d6**

GPIO number of D6 signal. Ignored in 1- or 4- line mode.

gpio_num_t **d7**

GPIO number of D7 signal. Ignored in 1- or 4- line mode.

gpio_num_t **gpio_cd**

GPIO number of card detect signal.

gpio_num_t **cd**

GPIO number of card detect signal; shorter name.

gpio_num_t **gpio_wp**

GPIO number of write protect signal.

gpio_num_t **wp**

GPIO number of write protect signal; shorter name.

uint8_t **width**

Bus width used by the slot (might be less than the max width supported)

uint32_t **flags**

Features used by this slot.

Macros

SDMMC_SLOT_FLAG_INTERNAL_PULLUP

Enable internal pullups on enabled pins. The internal pullups are insufficient however, please make sure external pullups are connected on the bus. This is for debug / example purpose only.

SDMMC_SLOT_FLAG_WP_ACTIVE_HIGH

GPIO write protect polarity. 0 means "active low", i.e. card is protected when the GPIO is low; 1 means "active high", i.e. card is protected when GPIO is high.

2.5.26 SD SPI 主机驱动程序

概述

SD SPI 主机驱动程序支持使用 SPI 主控驱动程序与一或多张 SD 卡通信，SPI 主控驱动程序则利用 SPI 主机实现功能。每张 SD 卡都通过一个 SD SPI 设备访问，相应设备以 SD SPI 设备句柄 *sdspi_dev_handle_t* 表示。调用 *sdspi_host_init_device()* 将设备连接到 SPI 总线上时会返回所需 SPI 设备句柄。注意，在使用 SPI 总线前，需要先通过 *spi_bus_initialize()* 初始化总线。

SD SPI 主机驱动程序基于 *SPI 主机驱动程序* 实现。借助 SPI 主控驱动程序，SD 卡及其他 SPI 设备可以共享同一 SPI 总线。SPI 主机驱动程序将处理来自不同任务的独占访问。

SD SPI 驱动程序使用受软件控制的 CS 信号。

使用方法

首先，使用宏*SDSPI_DEVICE_CONFIG_DEFAULT* 初始化结构体*sdspi_device_config_t*，该结构体用于初始化 SD SPI 设备。该宏还会填充默认的管脚映射，与 SDMMC 主机驱动的管脚映射相同。随后根据需要，修改结构体中的主机和管脚配置。然后调用 sdspi_host_init_device 初始化 SD SPI 设备，并将其连接到所属的总线上。

接着，使用宏*SDSPI_HOST_DEFAULT* 初始化结构体 sdmmc_host_t，该结构体用于存储上层（SD/SDIO/MMC 驱动）的状态和配置信息。将结构体中的 slot 参数设置为从 sdspi_host_init_device 返回的 SD SPI 设备的 SD SPI 句柄。使用 sdmmc_host_t 调用 sdmmc_card_init，检测并初始化 SD 卡。

初始化完成后，即可使用 SD/SDIO/MMC 驱动函数访问 SD 卡。

其他细节

通常，大多数应用程序仅使用驱动程序的以下 API 函数：

- *sdspi_host_init()*
- *sdspi_host_init_device()*
- *sdspi_host_remove_device()*
- *sdspi_host_deinit()*

对于其他函数，大多由协议层的 SD/SDIO/MMC 驱动程序通过 sdmmc_host_t 结构体中的函数指针使用。更多详情，请参阅*SD/SDIO/MMC 驱动程序*。

备注： 由于 SPI 驱动程序的限制，SD 卡在通过 SPI 总线与主设备进行数据传输和通信时，速度不超过 SDMMC_FREQ_DEFAULT。

警告： 在 SD 卡之间以及其他 SPI 设备间共享 SPI 总线时，存在部分限制，详情请参阅*SD 卡与其他 SPI 设备共享 SPI 总线*。

相关文档

SD 卡与其他 SPI 设备共享 SPI 总线

SD 卡支持 SPI 模式，使其能够作为 SPI 设备通信，但使用时需注意其限制。

其他设备的管脚负载 向同一总线添加设备会增加管脚的整体负载，包括交流负载（管脚电容）和直流负载（上拉电阻）。

交流负载 在通信速率不超过 50 MHz 的情况下，专为高速通信设计的 SD 卡采用小型管脚电容（交流负载小）。但在同一 SPI 总线上连接其他设备后，会增加管脚的交流负载。

管脚交流负载过高时，可能无法实现电平快速切换。通过使用示波器，你可以观察到管脚状态变化的边缘变得更加平滑，即边缘变化率更低。当 SD 卡连接到交流负载较高的总线时，可能无法满足 SD 卡建立时间 (setup) 的时序要求。高交流负载甚至可能导致 SD 卡和其他 SPI 设备无法正确解析主机发出的时钟信号，影响通信稳定性。

如果连接的其他设备的工作频率与 SD 卡工作频率不同，上述问题可能会更加明显。这是因为其他设备可能具有更大的管脚电容。管脚容量越大，响应时间越长，SD 总线可工作的最高频率越低。

你可以尝试以下测试，判断管脚交流负载是否过重：

术语

- **启动边沿 (launch edge)**：数据开始切换的时钟边沿；

- **锁存边沿 (latch edge)**：数据接收侧应进行数据采样的时钟边沿，对 SD 卡来说是上升沿。

1. 使用示波器观察时钟，并比较数据线与时钟。
   - 如果时钟不够快，例如上升/下降沿长于时钟周期的 1/4，表明时钟偏斜过大。
   - 如果在时钟锁存边沿之前数据线不稳定，表明数据线负载过大。

   借助逻辑分析仪，也可以观察到数据与时钟启动沿相比，延迟较大。但比起用示波器，用逻辑分析仪观察到的现象可能不太明显。
2. 尝试使用较低的时钟频率。
   如果设备可以在较低的通信频率下正常工作，而在较高的通信频率下出现问题，说明管脚交流负载过大。

如果管脚的交流负载过大，你可以选择使用其他管脚负载更低但通信速度更快的设备，或降低时钟速度。

直流负载 SD 卡所需的上拉电阻通常在 10 kΩ 至 50 kΩ 之间，注意对某些 SPI 设备来说，这可能是过强的上拉电阻。

请查阅设备的规格说明书，了解其直流输出电流。此直流输出电流应大于 700 µA，否则可能无法正确读取设备输出。

初始化顺序

备注： 如果在执行以下步骤时遇到任何问题，请首先确保时序正确。如前文所述，你可以尝试降低时钟速度，例如将 SD 卡的 `SDMMC_FREQ_PROBING` 设置为 400 kHz，排除管脚交流负载的影响。

在同一 SPI 总线上与其他 SPI 设备一起使用 SD 卡时，由于 SD 卡启动流程的限制，应遵循以下初始化顺序。有关详情，请参阅 storage/sd_card。

1. 通过 *`spi_bus_initialize()`* 正确初始化总线。
2. 将除 SD 卡外所有设备的 CS 线置为空闲状态（默认为高电平），避免在后续步骤中与 SD 卡发生冲突。
   这可以通过以下任一方式实现：
   1. 调用 *`spi_bus_add_device()`* 将设备连接到 SPI 总线，该函数将初始化用作 CS 的 GPIO 管脚到空闲电平：默认为高电平。
   2. 在添加新设备前，初始化需要拉高的 CS 管脚 GPIO。
   3. 在 ESP 的 GPIO 未初始化前，依靠内部/外部上拉电阻拉高所有 CS 管脚（**不推荐**）。请确保上拉电阻拥有足够强度，且没有其他下拉电阻影响拉高。例如，应启用内部下拉电阻。
3. 调用 *`esp_vfs_fat_sdspi_mount()`* 将卡挂载到文件系统。
   此步骤将使 SD 卡进入 SPI 模式，**应该**在同一总线上的所有其他 SPI 通信前完成。否则，SD 卡会保持在 SD 模式，即使在未选中其 CS 线的情况下，SD 卡也可能随机响应总线上的任何 SPI 通信。
   如果你想测试这种行为，请注意，一旦 SD 卡置于 SPI 模式，在下次上电前，也就是断电后重新上电之前，SD 卡将不会返回 SD 模式
4. 此时，即可与其他 SPI 设备自由通信。

API 参考

Header File

- components/esp_driver_sdspi/include/driver/sdspi_host.h
- This header file can be included with:

```
#include "driver/sdspi_host.h"
```

- This header file is a part of the API provided by the `esp_driver_sdspi` component. To declare that your component depends on `esp_driver_sdspi`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_sdspi
```

or

```
PRIV_REQUIRES esp_driver_sdspi
```

Functions

esp_err_t **sdspi_host_init**(void)

Initialize SD SPI driver.

备注： This function is not thread safe

返回

- ESP_OK on success
- other error codes may be returned in future versions

esp_err_t **sdspi_host_init_device**(const *sdspi_device_config_t* *dev_config, *sdspi_dev_handle_t* *out_handle)

Attach and initialize an SD SPI device on the specific SPI bus.

备注： This function is not thread safe

备注： Initialize the SPI bus by `spi_bus_initialize()` before calling this function.

备注： The SDIO over sdspi needs an extra interrupt line. Call `gpio_install_isr_service()` before this function.

参数

- **dev_config** -- pointer to device configuration structure
- **out_handle** -- Output of the handle to the sdspi device.

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if sdspi_host_init_device has invalid arguments
- ESP_ERR_NO_MEM if memory can not be allocated
- other errors from the underlying spi_master and gpio drivers

esp_err_t **sdspi_host_remove_device**(*sdspi_dev_handle_t* handle)

Remove an SD SPI device.

参数 **handle** -- Handle of the SD SPI device

返回 Always ESP_OK

esp_err_t **sdspi_host_do_transaction**(*sdspi_dev_handle_t* handle, sdmmc_command_t *cmdinfo)

Send command to the card and get response.

This function returns when command is sent and response is received, or data is transferred, or timeout occurs.

备注： This function is not thread safe w.r.t. init/deinit functions, and bus width/clock speed configuration functions. Multiple tasks can call sdspi_host_do_transaction as long as other sdspi_host_* functions are not called.

参数

- **handle** -- Handle of the sdspi device
- **cmdinfo** -- pointer to structure describing command and data to transfer

返回

- ESP_OK on success
- ESP_ERR_TIMEOUT if response or data transfer has timed out
- ESP_ERR_INVALID_CRC if response or data transfer CRC check has failed

- ESP_ERR_INVALID_RESPONSE if the card has sent an invalid response

esp_err_t **sdspi_host_set_card_clk**(*sdspi_dev_handle_t* host, uint32_t freq_khz)

Set card clock frequency.

Currently only integer fractions of 40MHz clock can be used. For High Speed cards, 40MHz can be used. For Default Speed cards, 20MHz can be used.

备注: This function is not thread safe

参数

- **host** -- Handle of the sdspi device
- **freq_khz** -- card clock frequency, in kHz

返回

- ESP_OK on success
- other error codes may be returned in the future

esp_err_t **sdspi_host_get_real_freq**(*sdspi_dev_handle_t* handle, int *real_freq_khz)

Calculate working frequency for specific device.

参数

- **handle** -- SDSPI device handle
- **real_freq_khz** -- **[out]** output parameter to hold the calculated frequency (in kHz)

返回

- ESP_ERR_INVALID_ARG : `handle` is NULL or invalid or `real_freq_khz` parameter is NULL
- ESP_OK : Success

esp_err_t **sdspi_host_deinit**(void)

Release resources allocated using sdspi_host_init.

备注: This function is not thread safe

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdspi_host_init function has not been called

esp_err_t **sdspi_host_io_int_enable**(*sdspi_dev_handle_t* handle)

Enable SDIO interrupt.

参数 **handle** -- Handle of the sdspi device

返回

- ESP_OK on success

esp_err_t **sdspi_host_io_int_wait**(*sdspi_dev_handle_t* handle, TickType_t timeout_ticks)

Wait for SDIO interrupt until timeout.

参数

- **handle** -- Handle of the sdspi device
- **timeout_ticks** -- Ticks to wait before timeout.

返回

- ESP_OK on success

esp_err_t **sdspi_host_get_dma_info**(int slot, esp_dma_mem_info_t *dma_mem_info)

Get the DMA memory information for the host driver.

参数

- **slot** -- **[in]** Not used

- **dma_mem_info** -- **[out]** DMA memory information structure

返回

- ESP_OK: ON success.
- ESP_ERR_INVALID_ARG: Invalid argument.

Structures

struct **sdspi_device_config_t**

Extra configuration for SD SPI device.

Public Members

spi_host_device_t **host_id**

SPI host to use, SPIx_HOST (see spi_types.h).

gpio_num_t **gpio_cs**

GPIO number of CS signal.

gpio_num_t **gpio_cd**

GPIO number of card detect signal.

gpio_num_t **gpio_wp**

GPIO number of write protect signal.

gpio_num_t **gpio_int**

GPIO number of interrupt line (input) for SDIO card.

bool **gpio_wp_polarity**

GPIO write protect polarity 0 means ”active low”, i.e. card is protected when the GPIO is low; 1 means ”active high”, i.e. card is protected when GPIO is high.

Macros

SDSPI_DEFAULT_HOST

SDSPI_DEFAULT_DMA

SDSPI_HOST_DEFAULT()

Default sdmmc_host_t structure initializer for SD over SPI driver.

Uses SPI mode and max frequency set to 20MHz

’slot’ should be set to an sdspi device initialized by `sdspi_host_init_device()`.

SDSPI_SLOT_NO_CS

indicates that card select line is not used

SDSPI_SLOT_NO_CD

indicates that card detect line is not used

SDSPI_SLOT_NO_WP

indicates that write protect line is not used

SDSPI_SLOT_NO_INT

indicates that interrupt line is not used

SDSPI_IO_ACTIVE_LOW

SDSPI_DEVICE_CONFIG_DEFAULT()

Macro defining default configuration of SD SPI device.

Type Definitions

typedef int **sdspi_dev_handle_t**

Handle representing an SD SPI device.

2.5.27 Sigma-Delta 调制器 (SDM)

简介

ESP32-P4 具备二阶 Sigma-Delta 调制器，可以为多个通道生成独立的脉冲密度调制 (PDM) 脉冲。请参阅技术参考手册，查看可用的硬件通道数量。[1]

Sigma-Delta 调制器可以将模拟电压信号转换为脉冲频率或脉冲密度，该过程称为脉冲密度调制 (PDM)（请参阅 维基百科上有关 Sigma-Delta 调制的介绍）。

与 I2S 外设中的 PDM 模式和和数模转换器 (DAC) 相比，SDM 中的 PDM 主要有以下特点：

1. SDM 没有时钟信号，类似于 PDM 的 DAC 模式；
2. SDM 没有 DMA 支持，无法持续改变其输出密度。如果需要改变 SDM 的输出密度，可以在定时器的回调函数中进行操作；
3. 基于以上两点，不同于 DAC，要还原模拟波形，还必须使用外部的有源或无源低通滤波器，详情请参阅转换为模拟信号（可选）。

Sigma-Delta 调制通道通常应用于以下场景：

- LED 调光
- 使用有源 RC 低通滤波器，实现简单的数模转换（8 位分辨率）
- 结合半桥或全桥回路，以及 LC 低通滤波器，实现 D 级功率放大

功能概述

下文将分节概述安装和操作 SDM 通道的一般步骤：

- 资源分配 - 介绍如何初始化和配置 SDM 通道，以及在通道完成任务后如何回收相关资源。
- 启用和禁用通道 - 介绍如何启用和禁用 SDM 通道。
- 设置脉冲密度 - 介绍如何设置 PDM 脉冲的等效占空比。
- 电源管理 - 介绍不同时钟源对功耗的影响。
- *IRAM* 安全 - 介绍禁用 cache 时仍可使用的功能。
- 线程安全 - 介绍由驱动程序认证为线程安全的 API。
- *Kconfig* 选项 - 介绍 SDM 驱动程序支持的各种 Kconfig 选项，这些选项可以给驱动程序的行为造成不同影响。

[1] 不同的 ESP 芯片系列可能具有不同数量的 SDM 通道，请参阅 ESP32-P4 技术参考手册中的 GPIO 和 IOMUX 章节，了解更多详情。驱动程序对通道申请数量不做限制，但当硬件资源用尽时，驱动程序将返回错误。因此，每次进行通道分配（如调用 *sdm_new_channel()*）时，请注意检查返回值。

资源分配 在 ESP-IDF 中，SDM 通道的信息和属性通过特定的数据结构进行管理和访问，该数据结构表示为*sdm_channel_handle_t*。每个通道都可以输出由硬件生成的二进制信号，且这些信号都经过 Sigma-Delta 调制。所有可用通道均存放在资源池中，由驱动程序管理，无需手动将固定通道分配给 GPIO。

要安装 SDM 通道，请调用*sdm_new_channel()* 获取通道句柄。通道的具体配置信息由结构体*sdm_config_t* 传递。

- *sdm_config_t::gpio_num* 设置 PDM 脉冲输出的 GPIO 管脚号。
- *sdm_config_t::clk_src* 选择 SDM 模块的时钟源。注意，所有通道选择的时钟源应保持一致。
- *sdm_config_t::sample_rate_hz* 设置 SDM 模块的采样率。
- *sdm_config_t::invert_out* 设置是否反转输出信号。
- *sdm_config_t::io_loop_back* 通过 GPIO 矩阵外设，启用 GPIO 的输入和输出功能。注意，该字段仅供调试使用。

函数*sdm_new_channel()* 可能因为各种原因失败，如内存不足、参数无效等。当缺少空闲通道（即所有的硬件 SDM 通道均在使用中）时，将返回*ESP_ERR_NOT_FOUND*。

SDM 通道完成任务后，请调用*sdm_del_channel()* 回收相应资源，以便底层硬件通道用于其他目的。在删除 SDM 通道句柄前，请通过*sdm_channel_disable()* 禁用要删除的通道，或确保该通道尚未由*sdm_channel_enable()* 启用，再继续删除操作。

创建采样率为 1 MHz 的 SDM 通道

```
sdm_channel_handle_t chan = NULL;
sdm_config_t config = {
    .clk_src = SDM_CLK_SRC_DEFAULT,
    .sample_rate_hz = 1 * 1000 * 1000,
    .gpio_num = 0,
};
ESP_ERROR_CHECK(sdm_new_channel(&config, &chan));
```

启用和禁用通道 在对 SDM 通道进行进一步的 IO 控制之前，需要先调用*sdm_channel_enable()* 启用通道。在内部，该函数实现了以下操作：

- 将通道状态从 **init** 切换到 **enable**
- 如果选择了特定时钟源（如 APB 锁），则会获取合适的电源管理锁。要了解更多有关信息，请参阅电源管理。

调用*sdm_channel_disable()* 则执行相反操作，即将通道恢复到 **init** 状态，并释放电源管理锁。

设置脉冲密度 在 PDM 中，脉冲密度决定了低通滤波器转换后的输出模拟电压，该模拟电压可以通过公式 `Vout = VDD_IO / 256 * duty + VDD_IO / 2` 计算。使用函数*sdm_channel_set_pulse_density()* 时，需要传入一个名为 `density` 的参数。这个参数是一个整数值，范围在 -128 到 127 之间，表示一个 8 位有符号整数。根据 `density` 参数的不同取值，输出信号的占空比也会相应改变。例如，如果将 `density` 参数设置为零，输出信号的占空比约为 50%。

电源管理 启用电源管理（即启用*CONFIG_PM_ENABLE*）时，在进入 Light-sleep 模式前，系统会调整 APB 频率，这可能会改变 Sigma-Delta 调制器的采样率。

但是，通过获取类型为*ESP_PM_APB_FREQ_MAX* 的电源管理锁，驱动程序可以防止系统改变 APB 频率。每当驱动程序创建 SDM 通道，且该通道选择 `SDM_CLK_SRC_APB` 作为其时钟源时，在通过*sdm_channel_enable()* 启用通道的过程中，驱动程序会确保获取类型为*ESP_PM_APB_FREQ_MAX* 的电源管理锁。反之，调用*sdm_channel_disable()* 禁用通道时，驱动程序释放该锁。

IRAM 安全 Kconfig 选项*CONFIG_SDM_CTRL_FUNC_IN_IRAM* 支持将常用的 IO 控制函数存放在 IRAM 中，以保证在禁用 cache 时可以正常使用函数。IO 控制函数如下所示：

- *sdm_channel_set_pulse_density()*

线程安全 驱动程序会确保工厂函数*sdm_new_channel()* 的线程安全，使用时，可以直接从不同的 RTOS 任务中调用此类函数，无需额外锁保护。

驱动程序设置了临界区，以防函数同时在任务和 ISR 中调用。因此，以下函数支持在 ISR 上下文运行：

- *sdm_channel_set_pulse_density()*

其他以*sdm_channel_handle_t* 作为第一个位置参数的函数均非线程安全，因此应避免从多个任务中调用这类函数。

Kconfig 选项

- *CONFIG_SDM_CTRL_FUNC_IN_IRAM* 控制 SDM 通道控制函数的存放位置（IRAM 或 flash）。更多信息请参阅*IRAM 安全*。
- *CONFIG_SDM_ENABLE_DEBUG_LOG* 用于启用调试日志输出。启用此选项将增加固件的二进制文件大小。

转换为模拟信号（可选）

一般而言，Sigma-Delta 信号连接到 LED 用来调节明暗时，无需在信号和 LED 之间添加滤波器，因为人眼本身对光强变化有低通滤波作用。但是，如果你想测量实际电压，或观察模拟波形，就需要设计一个模拟低通滤波器。此外，建议使用有源滤波器，相较于无源滤波器，有源滤波器在处理信号时具有更强的抗干扰性，且损失的电压较少。

请参阅如下示例 Sallen-Key 拓扑低通滤波器，了解滤波器的相关知识。

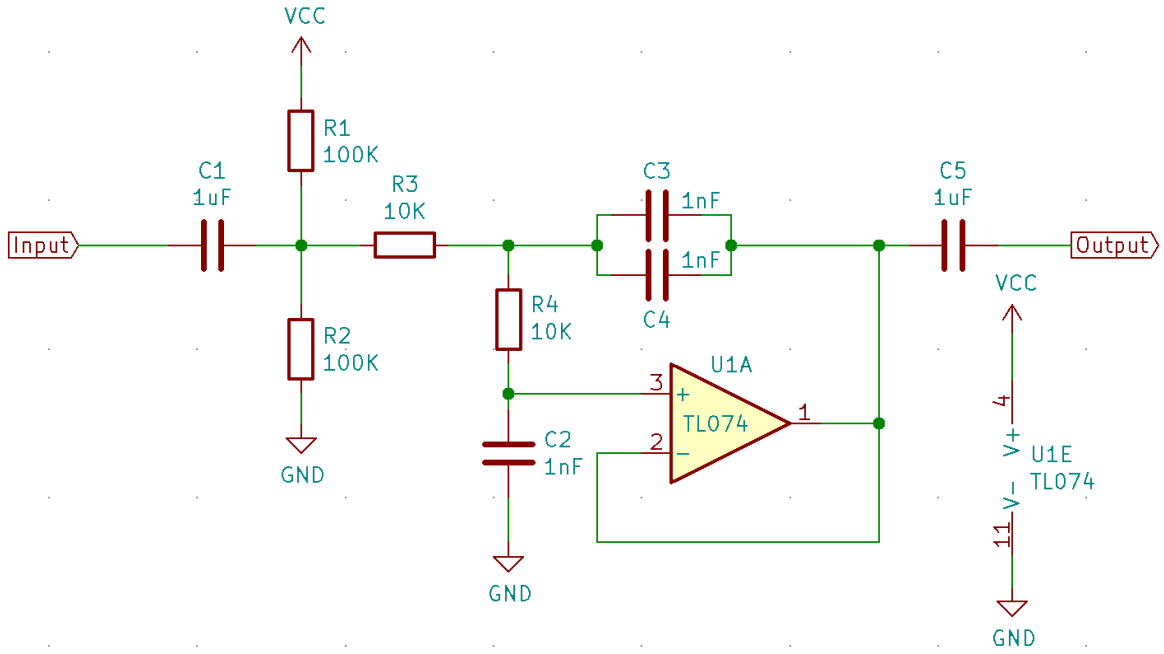


图 27: Sallen-Key 拓扑低通滤波器

应用示例

- 使用 Sigma-Delta 调制的 100 Hz 正弦波：peripherals/sigma_delta/sdm_dac。
- 使用 Sigma-Delta 调制、并由 GPIO 驱动的 LED：peripherals/sigma_delta/sdm_led。

API 参考

Header File

- components/esp_driver_sdm/include/driver/sdm.h
- This header file can be included with:

```
#include "driver/sdm.h"
```

- This header file is a part of the API provided by the `esp_driver_sdm` component. To declare that your component depends on `esp_driver_sdm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_sdm
```

or

```
PRIV_REQUIRES esp_driver_sdm
```

Functions

esp_err_t **sdm_new_channel**(const *sdm_config_t* *config, *sdm_channel_handle_t* *ret_chan)

Create a new Sigma Delta channel.

参数

- **config** -- **[in]** SDM configuration
- **ret_chan** -- **[out]** Returned SDM channel handle

返回

- ESP_OK: Create SDM channel successfully
- ESP_ERR_INVALID_ARG: Create SDM channel failed because of invalid argument
- ESP_ERR_NO_MEM: Create SDM channel failed because out of memory
- ESP_ERR_NOT_FOUND: Create SDM channel failed because all channels are used up and no more free one
- ESP_FAIL: Create SDM channel failed because of other error

esp_err_t **sdm_del_channel**(*sdm_channel_handle_t* chan)

Delete the Sigma Delta channel.

参数 **chan** -- **[in]** SDM channel created by `sdm_new_channel`

返回

- ESP_OK: Delete the SDM channel successfully
- ESP_ERR_INVALID_ARG: Delete the SDM channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete the SDM channel failed because the channel is not in init state
- ESP_FAIL: Delete the SDM channel failed because of other error

esp_err_t **sdm_channel_enable**(*sdm_channel_handle_t* chan)

Enable the Sigma Delta channel.

备注: This function will transit the channel state from init to enable.

备注: This function will acquire a PM lock, if a specific source clock (e.g. APB) is selected in the *sdm_config_t*, while `CONFIG_PM_ENABLE` is enabled.

参数 **chan** -- **[in]** SDM channel created by `sdm_new_channel`

返回

- ESP_OK: Enable SDM channel successfully
- ESP_ERR_INVALID_ARG: Enable SDM channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable SDM channel failed because the channel is already enabled

- ESP_FAIL: Enable SDM channel failed because of other error

esp_err_t **sdm_channel_disable**(*sdm_channel_handle_t* chan)

Disable the Sigma Delta channel.

备注: This function will do the opposite work to the `sdm_channel_enable()`

参数 **chan** -- **[in]** SDM channel created by `sdm_new_channel`

返回

- ESP_OK: Disable SDM channel successfully
- ESP_ERR_INVALID_ARG: Disable SDM channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable SDM channel failed because the channel is not enabled yet
- ESP_FAIL: Disable SDM channel failed because of other error

esp_err_t **sdm_channel_set_pulse_density**(*sdm_channel_handle_t* chan, int8_t density)

Set the pulse density of the PDM output signal.

备注: The raw output signal requires a low-pass filter to restore it into analog voltage, the restored analog output voltage could be Vout = VDD_IO / 256 * density + VDD_IO / 2

备注: This function is allowed to run within ISR context

备注: This function will be placed into IRAM if `CONFIG_SDM_CTRL_FUNC_IN_IRAM` is on, so that it's allowed to be executed when Cache is disabled

参数

- **chan** -- **[in]** SDM channel created by `sdm_new_channel`
- **density** -- **[in]** Quantized pulse density of the PDM output signal, ranges from -128 to 127. But the range of [-90, 90] can provide a better randomness.

返回

- ESP_OK: Set pulse density successfully
- ESP_ERR_INVALID_ARG: Set pulse density failed because of invalid argument
- ESP_FAIL: Set pulse density failed because of other error

esp_err_t **sdm_channel_set_duty**(*sdm_channel_handle_t* chan, int8_t duty)

The alias function of `sdm_channel_set_pulse_density`, it decides the pulse density of the output signal.

备注: `sdm_channel_set_pulse_density` has a more appropriate name compare this alias function, suggest to turn to `sdm_channel_set_pulse_density` instead

参数

- **chan** -- **[in]** SDM channel created by `sdm_new_channel`
- **duty** -- **[in]** Actually it's the quantized pulse density of the PDM output signal

返回

- ESP_OK: Set duty cycle successfully
- ESP_ERR_INVALID_ARG: Set duty cycle failed because of invalid argument
- ESP_FAIL: Set duty cycle failed because of other error

Structures

struct **sdm_config_t**

Sigma Delta channel configuration.

Public Members

int **gpio_num**

GPIO number

sdm_clock_source_t **clk_src**

Clock source

uint32_t **sample_rate_hz**

Over sample rate in Hz, it determines the frequency of the carrier pulses

uint32_t **invert_out**

Whether to invert the output signal

uint32_t **io_loop_back**

For debug/test, the signal output from the GPIO will be fed to the input path as well

struct *sdm_config_t*::[anonymous] **flags**

Extra flags

Type Definitions

typedef struct sdm_channel_t ***sdm_channel_handle_t**

Type of Sigma Delta channel handle.

Header File

- components/hal/include/hal/sdm_types.h
- This header file can be included with:

```
#include "hal/sdm_types.h"
```

Type Definitions

typedef *soc_periph_sdm_clk_src_t* **sdm_clock_source_t**

2.5.28 SPI flash API

概述

spi_flash 组件提供外部 flash 数据读取、写入、擦除和内存映射相关的 API 函数。

关于更多高层次的用于访问分区（分区表定义于分区表）的 API 函数，参见分区 *API* 。

备注： 访问主 flash 芯片时，建议使用上述 `esp_partition_*` API 函数，而非低层级的 `esp_flash_*` API 函数。分区表 API 函数根据存储在分区表中的数据，进行边界检查并计算在 flash 中的正确偏移量。不过，仍支持使用 `esp_flash_*` 函数直接访问外部（额外）的 SPI flash 芯片。

与 ESP-IDF v4.0 之前的 API 不同，这一版 `esp_flash_*` API 功能并不局限于主 SPI flash 芯片（即运行程序的 SPI flash 芯片）。通过使用不同的芯片指针，可以访问连接到 SPI0/1 或 SPI2 总线的外部 flash 芯片。

备注： 大多数 `esp_flash_*` API 使用 SPI1，SPI2 等外设而非通过 SPI0 上的 cache。这使得它们不仅能访问主 flash，也能访问外部 flash。

而由于 cache 的限制，所有经过 cache 的操作都只能对主 flash 进行。这些操作的地址同样受到 cache 能力的限制。Cache 无法访问外部 flash 或者高于它能力的地址段。这些 cache 操作包括：mmap、加密读写、执行代码或者访问在 flash 中的变量。

备注： ESP-IDF v4.0 之后的 flash API 不再是 **原子**的。因此，如果读操作执行过程中发生写操作，且读操作和写操作的 flash 地址出现重叠，读操作返回的数据可能会包含旧数据和新数据（新数据为写操作更新产生的数据）。

备注： 仅有主 flash 芯片支持加密操作，外接（经 SPI1 使用其他不同片选访问，或经其它 SPI 总线访问）的 flash 芯片则不支持加密操作。硬件的限制也决定了仅有主 flash 支持从 cache 当中读取。

---

flash 功能支持情况

支持的 flash 列表 不同厂家的 flash 特性有不同的操作方式，因此需要特殊的驱动支持。当前驱动支持大多数厂家 flash 24 位地址范围内的快速/慢速读，以及二线模式 (DIO/DOUT)，因为他们不需要任何厂家的自定义命令。

当前驱动支持以下厂家/型号的 flash 的四线模式 (QIO/QOUT)：

1. ISSI
2. GD
3. MXIC
4. FM
5. Winbond
6. XMC
7. BOYA

备注： 只有 ESP32-P4 支持上述某个 flash 时，芯片的驱动才默认支持这款 flash。可使用 menuconfig 中的 `Component config` > `SPI flash driver` > `Auto-detect flash chips` 选项来使能/禁用某个 flash。

---

flash 可选的功能

Optional Features for Flash Some features are not supported on all ESP chips and Flash chips. You can check the list below for more information.

- *Auto Suspend & Resume*
- *Flash unique ID*
- *High Performance Mode of QSPI Flash Chips*
- *32-bit Address Support of QSPI Flash Chips*
- *OPI Flash Support*

备注： When Flash optional features listed in this page are used, aside from the capability of ESP chips, and ESP-IDF version you are using, you will also need to make sure these features are supported by flash chips used.

- If you are using an official Espressif modules/SiP. Some of the modules/SiPs always support the feature, in this case you can see these features listed in the datasheet. Otherwise please contact Espressif's business team to know if we can supply such products for you.
- If you are making your own modules with your own bought flash chips, and you need features listed above. Please contact your vendor if they support the those features, and make sure that the chips can be supplied continuously.

注意： This document only shows that ESP-IDF code has supported the features of those flash chips. It is not a list of stable flash chips certified by Espressif. If you build your own hardware from flash chips with your own brought flash chips (even with flash listed in this page), you need to validate the reliability of flash chips yourself.

Auto Suspend & Resume This feature is only supported on ESP32-S3, ESP32-C2, ESP32-C3, ESP32-C6, ESP32-H2 for now.

The support for ESP32-P4 may be added in the future.

List of Flash chips that support this feature:

1. XM25QxxC series.
2. GD25QxxE series.
3. FM25Q32

注意： There are multiple limitations about the auto-suspend feature, please do read *flash 自动暂停功能* for more information before you enable this feature.

Flash Unique ID This feature is supported on all Espressif chips.

Unique ID is not flash id, which means flash has 64-Bit unique ID for each device. The instruction to read the unique ID (4Bh) accesses a factory-set read-only 64-bit number that is unique to each flash device. This ID number helps you to recognize each single device. Not all flash vendors support this feature. If you try to read the unique ID on a chip which does not have this feature, the behavior is not determined. The support list is as follows.

List of Flash chips that support this feature:

1. ISSI
2. GD
3. TH
4. FM
5. Winbond
6. XMC
7. BOYA

High Performance Mode of QSPI Flash Chips This featuer is only supported on ESP32-S3 for now.

The support for ESP32-S2, ESP32-C3, ESP32-C6, ESP32-H2, ESP32-P4 may be added in the future.

备注： This section is provided for QSPI flash chips. Octal flash used on ESP-chips support High performance mode by default so far, please refer to the *OPI Flash Support* for the list of supported octal flash chips.

32-bit Address Support of QSPI Flash Chips This feature is supported on all Espressif chips (see restrictions to application below).

备注： This section is provided for QSPI flash chips. The 32-bit address support of Octal Flash chips are considered as part of the Octal flash support. Please refer to the *OPI Flash Support* for the list of supported octal flash chips.

Most NOR flash chips used by Espressif chips use 24-bits address, which can cover 16 MBytes memory. However, for larger memory (usually equal to or larger than 32 MBytes), flash uses a 32-bits address to address memory region higher than 16 MBytes. Unfortunately, 32-bits address chips have vendor-specific commands, so we need to support the chips one by one.

List of Flash chips that support this feature:

1. W25Q256
2. GD25Q256

Restrictions By default, space over 16 MBytes on flash mentioned above can be used for `data saving`, like file system.

Furthermore, to map data/instructions to 32-bit physical address space (so as to be accessed by the CPU), please enable the config `IDF_EXPERIMENTAL_FEATURES` and `BOOTLOADER_CACHE_32BIT_ADDR_QUAD_FLASH`.

Please note that, this option is experimental, which means it can not be used on all flash chips stably. For more information, please contact Espressif Business Support.

OPI Flash Support This feature is only supported on ESP32-S3 for now.

OPI flash means that the flash chip supports octal peripheral interface, which has octal I/O pins. Different octal flash has different configurations and different commands. Hence, it is necessary to carefully check the support list.

有一些功能可能不是所有的 flash 芯片都支持，或不是所有的 ESP 芯片都支持。这些功能包括：

- 32 比特地址的 flash 支持 - 通常意味着拥有大于 16 MB 内存空间的大容量 flash 需要更长的地址去访问。
- flash 的私有 ID (unique ID) - 表示 flash 支持它自己的 64-bit 独有 ID。
- 暂停与恢复 - 表示 flash 可以在读/写的过程中接受暂停/恢复的命令。ESP32-P4 可以在 flash 正在写/擦除的过程中保持 cache 开启，并能随机读取 flash 中的内容。

如果想使用这些功能，则需保证 ESP32-P4 支持这些功能，且产品里所使用的 flash 芯片也要支持这些功能。请参阅*Optional Features for Flash*，查看更多信息。

也可以自定义 flash 芯片驱动。请参阅*Overriding Default Chip Drivers*，查看详细信息。

警告： Customizing SPI Flash Chip Drivers is considered an ”expert” feature. Users should only do so at their own risk. (See the notes below)

Overriding Default Chip Drivers During the SPI Flash driver's initialization (i.e., *esp_flash_init()*), there is a chip detection step during which the driver iterates through a Default Chip Driver List and determine which chip driver can properly support the currently connected flash chip. The Default Chip Drivers are provided by the ESP-IDF, thus are updated in together with each ESP-IDF version. However ESP-IDF also allows users to customize their own chip drivers.

Users should note the following when customizing chip drivers:

1. You may need to rely on some non-public ESP-IDF functions, which have slight possibility to change between ESP-IDF versions. On the one hand, these changes may be useful bug fixes for your driver, on the other hand, they may also be breaking changes (i.e., breaks your code).
2. Some ESP-IDF bug fixes to other chip drivers are not automatically applied to your own custom chip drivers.
3. If the protection of flash is not handled properly, there may be some random reliability issues.
4. If you update to a newer ESP-IDF version that has support for more chips, you will have to manually add those new chip drivers into your custom chip driver list. Otherwise the driver will only search for the drivers in custom list you provided.

Steps For Creating Custom Chip Drivers and Overriding the ESP-IDF Default Driver List

1. Enable the *CONFIG_SPI_FLASH_OVERRIDE_CHIP_DRIVER_LIST* config option. This prevents compilation and linking of the Default Chip Driver List (`default_registered_chips`) provided by ESP-IDF. Instead, the linker searches for the structure of the same name (`default_registered_chips`) that must be provided by the user.
2. Add a new component in your project, e.g., `custom_chip_driver`.
3. Copy the necessary chip driver files from the `spi_flash` component in ESP-IDF. This may include:
   - `spi_flash_chip_drivers.c` (to provide the `default_registered_chips` structure)
   - Any of the `spi_flash_chip_*.c` files that matches your own flash model best
   - `CMakeLists.txt` and `linker.lf` files

   Modify the files above properly. Including:
   - Change the `default_registered_chips` variable to non-static and remove the #ifdef logic around it.
   - Update `linker.lf` file to rename the fragment header and the library name to match the new component.
   - If reusing other drivers, some header names need prefixing with `spi_flash/` when included from outside spi_flash component.

备注:

- When writing your own flash chip driver, you can set your flash chip capabilities through `spi_flash_chip_***(vendor)_get_caps` and points the function pointer `get_chip_caps` for protection to the `spi_flash_chip_***_get_caps` function. The steps are as follows.
  1. Please check whether your flash chip have the capabilities listed in `spi_flash_caps_t` by checking the flash datasheet.
  2. Write a function named `spi_flash_chip_***(vendor)_get_caps`. Take the example below as a reference. (if the flash support `suspend` and `read unique id`).
  3. Points the pointer `get_chip_caps` (in `spi_flash_chip_t`) to the function mentioned above.

```
spi_flash_caps_t spi_flash_chip_***(vendor)_get_caps(esp_flash_t *chip)
{
    spi_flash_caps_t caps_flags = 0;
    // 32-bit-address flash is not supported
    flash-suspend is supported
    caps_flags |= SPI_FLAHS_CHIP_CAP_SUSPEND;
    // flash read unique id.
    caps_flags |= SPI_FLASH_CHIP_CAP_UNIQUE_ID;
    return caps_flags;
}
```

```
const spi_flash_chip_t esp_flash_chip_eon = {
    // Other function pointers
    .get_chip_caps = spi_flash_chip_eon_get_caps,
};
```

- You also can see how to implement this in the example storage/custom_flash_driver.

4. Write a new `CMakeLists.txt` file for the `custom_chip_driver` component, including an additional line to add a linker dependency from `spi_flash` to `custom_chip_driver`:

```
idf_component_register(SRCS "spi_flash_chip_drivers.c"
                       "spi_flash_chip_mychip.c"  # modify as needed
                       REQUIRES hal
                       PRIV_REQUIRES spi_flash
                       LDFRAGMENTS linker.lf)
idf_component_add_link_dependency(FROM spi_flash)
```

   - An example of this component CMakeLists.txt can be found in storage/custom_flash_driver/components/custom_chip_driver/CMakeLists.txt
5. The `linker.lf` is used to put every chip driver that you are going to use whilst cache is disabled into internal RAM. See 链接器脚本生成机制 for more details. Make sure this file covers all the source files that you add.
6. Build your project, and you will see the new flash driver is used.

Example See also storage/custom_flash_driver.

初始化 flash 设备

在使用 `esp_flash_*` API 之前，需要在 SPI 总线上初始化芯片，步骤如下：

1. 调用*`spi_bus_initialize()`* 初始化 SPI 总线。此函数将初始化总线上设备间共享的资源，如 I/O、DMA、中断等。
2. 调用*`spi_bus_add_flash_device()`* 将 flash 设备连接到总线上。然后分配内存，填充 `esp_flash_t` 结构体，同时初始化 CS I/O。
3. 调用*`esp_flash_init()`* 与芯片进行通信。后续操作会依据芯片类型不同而有差异。

备注： 当前，已支持多个 flash 芯片连接到同一总线。

SPI flash 访问 API

如下所示为处理 flash 中数据的函数集：

- *`esp_flash_read()`*：将数据从 flash 读取到 RAM；
- *`esp_flash_write()`*：将数据从 RAM 写入到 flash；
- *`esp_flash_erase_region()`*：擦除 flash 中指定区域的数据；
- *`esp_flash_erase_chip()`*：擦除整个 flash；
- `esp_flash_get_chip_size()`：返回 menuconfig 中设置的 flash 芯片容量（以字节为单位）。

一般来说，请尽量避免对主 SPI flash 芯片直接使用原始 SPI flash 函数。如需对主 SPI flash 芯片进行操作，请使用分区专用函数。

SPI flash 容量

SPI flash 容量由引导加载程序镜像头部（烧录偏移量为 0x1000）的一个字段进行配置。

默认情况下，引导程序被写入 flash 时，`esptool.py` 会自动检测 SPI flash 容量，同时使用正确容量更新引导程序的头部。也可以在工程配置中设置*CONFIG_ESPTOOLPY_FLASHSIZE*，生成固定的 flash 容量。

如需在运行时覆盖已配置的 flash 容量，请配置 `g_rom_flashchip` 结构中的 `chip_size`。`esp_flash_*` 函数使用此容量（于软件和 ROM 中）进行边界检查。

SPI1 flash 并发约束

SPI1 flash 并发约束

指令/数据 cache（用以执行固件）与 SPI1 外设（由像 SPI flash 驱动一样的驱动程序控制）共享 SPI0/1 总线。因此，SPI1 外设上的操作会对整个系统造成显著的影响。这类操作包括调用 SPI flash API 或者 SPI1 总线上的其他驱动、任何 flash 操作（如读取、写入、擦除）或是由其他用户定义的 SPI 操作（对主 flash 或是其他 SPI 从机）。

在 ESP32-P4 上，配置选项*CONFIG_SPI_FLASH_AUTO_SUSPEND* 允许 flash/PSRAM 的 cache 访问和 SPI1 的操作并发执行。该选项是可选的，依赖于特定的 SPI Flash 型号，因此默认是关闭的。请参阅*flash 自动暂停功能*，查看详细信息。

禁用该选项时，在读取/写入/擦除 flash 期间，必须禁用 cache。使用驱动访问 SPI1 的相关约束参见*禁用 cache 时*。这些约束会带来更多的 IRAM/DRAM 消耗。

在 ESP32-P4 上，启用配置选项*CONFIG_SPIRAM_XIP_FROM_PSRAM*（默认禁用）后将允许 flash/PSRAM 的 cache 访问和 SPI1 的操作并发执行。请参阅*在 PSRAM 中执行代码*，查看详细信息。

禁用该选项时，在读取/写入/擦除 flash 期间，必须禁用 cache。使用驱动访问 SPI1 的相关约束参见*禁用 cache 时*。这些约束会带来更多的 IRAM/DRAM 消耗。

禁用 cache 时 此时，在 flash 擦写操作中，所有的 CPU 都只能执行 IRAM 中的代码，而且必须从 DRAM 中读取数据。如果使用本文档中的 API 函数，上述限制将自动生效且透明（无需额外关注），但这些限制可能会影响系统中的其他任务的性能。

备注： 启用*CONFIG_SPIRAM_XIP_FROM_PSRAM* 选项后，不会禁用 cache。

为避免意外读取 flash cache，一个 CPU 在启动 flash 写入或擦除操作时，另一个 CPU 将阻塞。在 flash 操作完成前，会禁用所有在 CPU 上非 IRAM 安全的中断。

另请参阅*OS 函数* 和*SPI 总线锁*。

除 SPI0/1 以外，SPI 总线上的其他 flash 芯片则不受这种限制。

请参阅*应用程序内存分布*，查看内部 RAM（如 IRAM、DRAM）和 flash cache 的区别。

IRAM 安全中断处理程序 如果需要在 flash 操作期间运行中断处理程序（比如低延迟操作），请在*注册中断处理程序* 时设置 `ESP_INTR_FLAG_IRAM`。

请确保中断处理程序访问的所有数据和函数（包括其调用的数据和函数）都存储在 IRAM 或 DRAM 中。参见*如何将代码放入 IRAM*。

在未将函数或符号正确放入 IRAM/DRAM 的情况下，在 flash 操作期间，中断处理程序从 flash cache 中读取数据时，会导致程序崩溃。这可能是因为代码未正确放入 IRAM，产生了非法指令异常，也可能是因为常数未正确放入 DRAM，读取到了垃圾数据。

备注： 在 ISRs 中处理字符串时，不建议使用 `printf` 和其他输出函数。为了方便调试，在从 ISRs 中获取数据时，请使用 `ESP_DRAM_LOGE()` 和类似的宏。请确保 `TAG` 和格式字符串都放置于 `DRAM` 中。

非 IRAM 安全中断处理程序 如果在注册时没有设置 `ESP_INTR_FLAG_IRAM` 标志，当禁用 cache 时，将不会执行中断处理程序。一旦 cache 恢复，非 IRAM 安全的中断将重新启用，中断处理程序随即再次正常运行。这意味着，只要禁用了 cache，就不会发生相应的硬件事件。

当 DMA 也可以访问 Flash 中的数据时 当 DMA 正在从 Flash 中读取数据时，来自 SPI1 的擦/写操作优先级会更高，如果 Flash 的 auto-suspend 功能没有开启，将会导致 DMA 读到错误的数据。建议在擦写 Flash 之前先停止 DMA 对 Flash 的访问。如果 DMA 不可以停止，比如 LCD 需要持续刷新保存在 Flash 中的图像数据，建议将此类数据拷贝到 PSRAM 或者内部的 SRAM 中。

flash 自动暂停功能

重要:

1. 应使用支持暂停/恢复功能的 flash。
2. 应使用支持自动暂停功能的 MSPI 硬件，即硬件支持自动发送暂停命令。

如果在不支持自动暂停的 flash 上使用该功能，可能会造成严重的程序崩溃。因此，强烈建议提前阅读规格书，确保 flash 至少满足如下条件：

1. 基于当前软件的实现限制，flash 状态寄存器中的 SUS 位应当位于 SR2 bit7 或 SR bit15。
2. 基于当前软件的实现限制，flash 暂停命令应为 75H，恢复命令为 7AH。
3. flash 成功暂停后，除已擦除的段或块外，flash 地址中的任何内容都支持读取。也可以在此种状态下立即下达恢复命令。
4. flash 从暂停模式恢复后，支持立即下达另一个暂停命令。

启用*CONFIG_SPI_FLASH_AUTO_SUSPEND* 后，缓存将保持启用状态，禁用该选项即可禁用缓存。SPI0 和 SPI1 之间的仲裁由硬件决定。当 SPI1 进行读取等耗时较短的操作时，CPU 和缓存将等待至 SPI1 操作完成。然而，在擦除、页面写入或状态寄存器写入等过程中，如 `SE`、`PP` 和 `WRSR`，自动暂停功能将中断正在进行的 flash 操作，使 CPU 得以在有限的时间内读取缓存及 flash 中的数据。

基于此功能，部分的代码及变量现可存放在 flash/PSRAM 中，同时仍能保证在 flash 擦除期间的正常执行，减少了 IRAM/DRAM 的消耗。

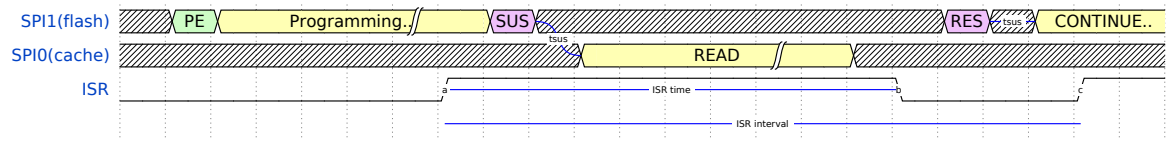
需注意，使用此功能暂停/恢复 flash 时，会导致额外开销。在 flash 擦除期间，频繁触发中断将显著延长擦除时间。为避免这种情况，建议调整 FreeRTOS 任务优先级，仅将实时关键的任务排在擦除操作前，从而确保在合理时间内完成 flash 擦除操作。

代码可以分为以下三类：

1. 关键代码：存放在 IRAM/DRAM 中。这类代码通常有较高的性能要求，与 cache/flash/PSRAM 相关，或者被频繁调用。
2. 缓存访问的代码：存放在 flash/PSRAM 中。这类代码的性能要求较低，或者较少被调用。flash 擦除时将执行这类代码，导致一定的开销。
3. 低优先级代码：存放在 flash/PSRAM 中，且在 flash 擦除期间禁止运行。这类代码的任务优先级应低于擦除操作，避免影响 flash 擦除的速度。

关于 flash 自动暂停功能的用法和对应的响应时间延迟，请参考 system/flash_suspend 示例。

备注: 由于 flash 自动暂停功能极度依赖时序，其多用作为一种优化方案，而非万能解法。例如 LCD 刷新、蓝牙、Wi-Fi 等场景对实时系统要求较高、或需要频繁触发中断，所以此方案并不适用。建议参照以下步骤，选择搭配 flash 自动暂停功能使用的 ISR 类型：



需注意图中的两个关键值：

1. ISR 时间 (ISR time)：ISR 时间不能过长，需小于 `IWDT` 中设置的值。ISR 时间会显著增加擦除/写入操作的完成时间。
2. ISR 间隔时间 (ISR interval)：由于不能频繁触发 ISR，需格外注意 **ISR 间隔时间减去 ISR 时间后的剩余时间**（图中 b 点至 c 点的距离）。在此期间，SPI1 会发送恢复命令重新启动操作，所需准备时间 `tsus` 的典型值约为 **40 us**。如果在 SPI1 完成恢复操作前接收到了新的暂停命令，可能导致 CPU 饥饿，触发 `TWDT`。

对于第 2 点中所提到的 tsus 时间可以通过翻阅 flash datasheets 查找，通常在 AC CHARACTERISTICS 章节中。用户需要保证从 datasheets 获得的 tsus 值不大于*CONFIG_SPI_FLASH_SUSPEND_TSUS_VAL_US* 值。

此外，flash 暂停可能延迟。CPU 和缓存通过 SPI0 频繁访问 flash，且 SPI1 频繁发送暂停命令时，会导致 MSPI 数据传输效率下降。可以通过在内部使用 **锁**来避免此种情况。当 SPI1 发送暂停命令时，SPI0 将接管内存 SPI 总线并启用锁。SPI0 完成数据传输后，在锁延迟时间结束前，都将保有对内存 SPI 总线的控制权。在此锁延迟期间，如果接收到其他 SPI0 事务，则该 SPI0 事务将正常进行，并开启新一轮锁延迟周期。如无其他 SPI0 事务，则 SPI0 释放内存总线并启动 SPI0/1 仲裁。

在 PSRAM 中执行代码 启用*CONFIG_SPIRAM_XIP_FROM_PSRAM* 选项后，flash 中 .text 部分的数据（用于指令）和 flash 中 .rodata 部分的数据（用于只读数据）将被放入 PSRAM。

相应的虚拟内存地址将被映射到 PSRAM。

如果同时启用以上两个选项，则在 SPI1 flash 操作期间 cache 不会被禁用，无需确保 ISR、ISR 回调及相关数据放置在内部 RAM 中。

注意： 指令/数据 cache（用以执行固件）与 SPI1 外设（由像 SPI flash 驱动一样的驱动程序控制）共享 SPI0/1 总线。因此，在 SPI1 总线上调用 SPI flash API（包括访问主 flash）会对整个系统造成显著的影响。请参阅*SPI1 flash 并发约束*，查看详细信息。

SPI flash 加密

SPI flash 内容支持加密，并在硬件层进行透明解密。

请参阅*flash 加密*，查看详细信息。

内存映射 API

ESP32-P4 的内存硬件可以将 flash 部分区域映射到指令地址空间和数据地址空间。此映射仅用于读操作，不能通过写入 flash 映射的存储区域来改变 flash 中的内容。

flash 在 64 KB 页进行映射。内存映射硬件既可将 flash 映射到数据地址空间，也能映射到指令地址空间。请查看技术参考手册，了解内存映射硬件的详细信息及有关限制。

请注意，有些页被用于将应用程序映射到内存中，因此实际可用的页会少于硬件提供的总数。

启用*flash 加密* 时，使用内存映射区域从 flash 读取数据是解密 flash 的唯一方法，解密需在硬件层进行。

内存映射 API 在 spi_flash_mmap.h 和 esp_partition.h 中声明：

- *spi_flash_mmap()*：将 flash 物理地址区域映射到 CPU 指令空间或数据空间；
- *spi_flash_munmap()*：取消上述区域的映射；
- *esp_partition_mmap()*：将分区的一部分映射至 CPU 指令空间或数据空间；

spi_flash_mmap() 和*esp_partition_mmap()* 的区别如下：

- *spi_flash_mmap()*：需要给定一个 64 KB 对齐的物理地址；
- *esp_partition_mmap()*：给定分区内任意偏移量即可，此函数根据需要将返回的指针调整至指向映射内存。

内存映射以页为单位，即使传递给 esp_partition_mmap 的是一个分区，分区外的数据也是也是可以被读取到的，不会受到分区边界的影响。

备注： 由于 mmap 是由 cache 支持的，因此，mmap 也仅能用在主 flash 上。

SPI flash 实现

`esp_flash_t` 结构体包含芯片数据和该 API 的三个重要部分：

1. 主机驱动，为访问芯片提供硬件支持；
2. 芯片驱动，为不同芯片提供兼容性服务；
3. OS 函数，在不同阶段（一级或二级 Boot 或者应用程序阶段）为部分 OS 函数（如锁、延迟）提供支持。

主机驱动 主机驱动依赖 `hal/include/hal` 文件夹下 `spi_flash_types.h` 定义的 `spi_flash_host_driver_t` 接口。该接口提供了一些常用的函数，用于与芯片通信。

在 SPI HAL 文件中，有些函数是基于现有的 ESP32-P4 memory-spi 来实现的。但是，由于 ESP32-P4 的速度限制，HAL 层无法提供某些读命令的高速实现（所以这些命令根本没有在 HAL 的文件中被实现）。`memspi_host_driver.h` 和 `.c` 文件使用 HAL 提供的 `common_command` 函数实现上述读命令的高速版本，并将所有它实现的以及 HAL 函数封装为 `spi_flash_host_driver_t` 供更上层调用。

仅通过 GPIO，也可实现自己的主机驱动。只要实现了 `spi_flash_host_driver_t` 中所有函数，不管底层硬件是什么，esp_flash API 都可以访问 flash。

芯片驱动 芯片驱动在 `spi_flash_chip_driver.h` 中进行定义，并将主机驱动提供的基本函数进行封装以供 API 层使用。

有些操作需在执行前先发送命令，或在执行后读取状态，因此有些芯片需要不同的命令或值以及通信方式。

`generic chip` 芯片代表了常见的 flash 芯片，其他芯片驱动可以在这种通用芯片的基础上进行开发。

芯片驱动依赖主机驱动。

OS 函数

SPI 特性

SPI 主机

SPI 总线锁 为了多路复用来自 SPI 主机、SPI flash 等不同驱动的设备，每个 SPI 总线上都配有 SPI 总线锁。驱动程序可以通过对锁实施仲裁，将设备连接到总线上。

每个总线锁都已初始化并注册了后台服务 (BG)。设备应在 BG 禁用后，再在总线上进行传输。

- SPI1 总线的后台服务为高速缓存。在设备操作开始前，总线锁可以禁用高速缓存，并在设备释放锁后将其再次启用。高速缓存处于禁用状态时，让出当前任务的执行权毫无意义，因此，该情况下 SPI1 总线上的任何设备都无法使用 ISR。
  SPI 主机驱动程序暂不支持 SPI1 总线。只有 SPI flash 驱动程序可以连接到该总线。
- 对于其他总线，驱动程序可以将 ISR 注册为后台服务。若设备任务要求独占总线，则总线锁将阻塞该任务，同时禁用 ISR，随即解除对该任务的阻塞。任务释放锁后，如果 ISR 中还有待处理的事务，则锁将尝试重新启用 ISR。

OS 函数层目前支持访问锁和延迟的方法。

锁（见*SPI 总线锁*）用于解决同一 SPI 总线上的设备访问和 SPI flash 芯片访问之间的冲突。例如：

1. 经 SPI1 总线访问 flash 芯片时，应当禁用 cache（平时用于获取代码和 PSRAM 数据）。
2. 经其他总线访问 flash 芯片时，应当禁用 flash 上 SPI 主驱动器注册的 ISR 以避免冲突。
3. SPI 主驱动器上某些没有 CS 线或者 CS 线受软件（如 SDSPI）控制的设备需要在一段时间内独占总线。

延时则用于某些长时操作，需要主机处于等待状态或执行轮询。

顶层 API 将芯片驱动和 OS 函数封装成一个完整的组件，并提供参数检查。

使用 OS 函数还可以在一定程度上避免在擦除大块 flash 区域时出现看门狗超时的情况。在这段时间内，CPU 将被 flash 擦除任务占用，从而阻止其他任务的执行，包括为看门狗定时器 (WDT) 供电的空闲任务。若已选中配置选项*CONFIG_ESP_TASK_WDT_PANIC*，并且 flash 操作时间长于看门狗的超时时间，系统将重新启动。

不过，由于不同的 flash 芯片擦除时间不同，flash 驱动几乎无法兼容，很难完全规避超时的风险，这一点需要格外注意。请遵照以下指南：

1. 建议启用*CONFIG_SPI_FLASH_YIELD_DURING_ERASE* 选项，允许调度器在擦除 flash 时进行重新调度。此外，还可以使用下列参数。

- 在 menuconfig 中增加*CONFIG_SPI_FLASH_ERASE_YIELD_TICKS* 或减少*CONFIG_SPI_FLASH_ERASE_YIELD_DURATION_MS* 的时间。
- 在 menuconfig 中增加*CONFIG_ESP_TASK_WDT_TIMEOUT_S* 的时间，以设置更长的看门狗超时周期。然而，看门狗超时周期拉长后，可能无法再检测到以前可检测到的超时。

1. 请注意，在进行长时间的 SPI flash 操作时，启用*CONFIG_ESP_TASK_WDT_PANIC* 选项将会在超时时触发紧急处理程序。不过，启用该选项也可以帮助处理应用程序中的意外异常，请根据实际情况决定是否需要启用这个选项。
2. 在开发过程中，请根据项目对擦除 flash 的具体要求和时间限制，谨慎进行 flash 操作。在配置 flash 擦除超时周期时，请在实际产品要求的基础上留出合理的冗余时间，从而提高产品的可靠性。

实现细节

必须确保操作期间，两个 CPU 均未从 flash 运行代码，实现细节如下：

- 单核模式下，SDK 在执行 flash 操作前将禁用中断或调度算法。
- 双核模式下，SDK 需确保两个 CPU 均未运行 flash 代码。

如果有 SPI flash API 在 CPU A（PRO 或 APP）上调用，它使用 `esp_ipc_call` API 在 CPU B 上运行 `spi_flash_op_block_func` 函数。`esp_ipc_call` API 会在 CPU B 上唤醒一个高优先级任务，即运行 `spi_flash_op_block_func` 函数。运行该函数将禁用 CPU B 上的 cache，并使用 `s_flash_op_can_start` 旗帜来标志 cache 已禁用。然后，CPU A 上的任务也会禁用 cache 并继续执行 flash 操作。

执行 flash 操作时，CPU A 和 CPU B 仍然可以执行中断操作。默认中断代码均存储于 RAM 中，如果新添加了中断分配 API，则应添加一个标志位以请求在 flash 操作期间禁用该新分配的中断。

flash 操作完成后，CPU A 上的函数将设置另一标志位，即 `s_flash_op_complete`，用以通知 CPU B 上的任务可以重新启用 cache 并释放 CPU。接着，CPU A 上的函数也重新启用 cache，并将控制权返还给调用者。

另外，所有 API 函数均受互斥量 `s_flash_op_mutex` 保护。

在单核环境中（启用*CONFIG_FREERTOS_UNICORE*），需要禁用上述两个 cache，以防发生 CPU 间通信。

相关文档

- *Optional Features for Flash*
- *SPI1 flash 并发约束*

SPI Flash API ESP-IDF Version vs Chip-ROM Version There is a set of SPI Flash drivers in Chip-ROM which you can use by enabling *CONFIG_SPI_FLASH_ROM_IMPL*. Most of the ESP-IDF SPI Flash driver code are in internal RAM, therefore enabling this option frees some internal RAM usage. Note if you enable this option, this means some SPI Flash driver features and bugfixes that are done in ESP-IDF might not be included in the Chip-ROM version.

Feature Supported by ESP-IDF but Not in Chip-ROM

- Octal Flash chip support. See *OPI Flash Support* for details.
- 32-bit-address support for GD25Q256. Note this feature is an optional feature, please do read *32-bit Address Support of QSPI Flash Chips* for details.
- TH Flash chip support.
- Kconfig option *CONFIG_SPI_FLASH_CHECK_ERASE_TIMEOUT_DISABLED*.
- *CONFIG_SPI_FLASH_VERIFY_WRITE*, enabling this option helps you detect bad writing.
- *CONFIG_SPI_FLASH_LOG_FAILED_WRITE*, enabling this option prints the bad writing.
- *CONFIG_SPI_FLASH_WARN_SETTING_ZERO_TO_ONE*, enabling this option checks if you are writing zero to one.
- *CONFIG_SPI_FLASH_DANGEROUS_WRITE*, enabling this option checks for flash programming to certain protected regions like bootloader, partition table or application itself.
- *CONFIG_SPI_FLASH_ENABLE_COUNTERS*, enabling this option to collect performance data for ESP-IDF SPI Flash driver APIs.
- *CONFIG_SPI_FLASH_AUTO_SUSPEND*, enabling this option to automatically suspend / resume a long Flash operation when short Flash operation happens. Note this feature is an optional feature, please do read *Auto Suspend & Resume* for more limitations.

Bugfixes Introduced in ESP-IDF but Not in Chip-ROM

- Detected Flash physical size correctly, for larger than 256MBit Flash chips. (Commit ID: b4964279d44f73cce7cfd5cf684567fbdfd6fd9e)

SPI flash API 参考

Header File

- components/spi_flash/include/esp_flash_spi_init.h
- This header file can be included with:

```
#include "esp_flash_spi_init.h"
```

- This header file is a part of the API provided by the `spi_flash` component. To declare that your component depends on `spi_flash`, add the following to your CMakeLists.txt:

```
REQUIRES spi_flash
```

or

```
PRIV_REQUIRES spi_flash
```

Functions

esp_err_t **spi_bus_add_flash_device**(*esp_flash_t* **out_chip, const *esp_flash_spi_device_config_t* *config)

Add a SPI Flash device onto the SPI bus.

The bus should be already initialized by `spi_bus_initialization`.

参数

- **out_chip** -- Pointer to hold the initialized chip.
- **config** -- Configuration of the chips to initialize.

返回

- ESP_ERR_INVALID_ARG: out_chip is NULL, or some field in the config is invalid.
- ESP_ERR_NO_MEM: failed to allocate memory for the chip structures.
- ESP_OK: success.

esp_err_t **spi_bus_remove_flash_device**(*esp_flash_t* *chip)

Remove a SPI Flash device from the SPI bus.

参数 **chip** -- The flash device to remove.

返回

- ESP_ERR_INVALID_ARG: The chip is invalid.
- ESP_OK: success.

Structures

struct **esp_flash_spi_device_config_t**

Configurations for the SPI Flash to init.

Public Members

spi_host_device_t **host_id**

Bus to use.

int **cs_io_num**

GPIO pin to output the CS signal.

esp_flash_io_mode_t **io_mode**

IO mode to read from the Flash.

enum *esp_flash_speed_s* **speed**

Speed of the Flash clock. Replaced by freq_mhz.

int **input_delay_ns**

Input delay of the data pins, in ns. Set to 0 if unknown.

int **cs_id**

CS line ID, ignored when not `host_id` is not SPI1_HOST, or `CONFIG_SPI_FLASH_SHARE_SPI1_BUS` is enabled. In this case, the CS line used is automatically assigned by the SPI bus lock.

int **freq_mhz**

The frequency of flash chip(MHZ)

Header File

- components/spi_flash/include/esp_flash.h
- This header file can be included with:

```
#include "esp_flash.h"
```

- This header file is a part of the API provided by the `spi_flash` component. To declare that your component depends on `spi_flash`, add the following to your CMakeLists.txt:

```
REQUIRES spi_flash
```

or

```
PRIV_REQUIRES spi_flash
```

Functions

esp_err_t **esp_flash_init**(*esp_flash_t* *chip)

Initialise SPI flash chip interface.

This function must be called before any other API functions are called for this chip.

备注： Only the `host` and `read_mode` fields of the chip structure must be initialised before this function is called. Other fields may be auto-detected if left set to zero or NULL.

备注： If the chip->drv pointer is NULL, chip chip_drv will be auto-detected based on its manufacturer & product IDs. See `esp_flash_registered_flash_drivers` pointer for details of this process.

参数 **chip** -- Pointer to SPI flash chip to use. If NULL, esp_flash_default_chip is substituted.
返回 ESP_OK on success, or a flash error code if initialisation fails.

bool **esp_flash_chip_driver_initialized**(const *esp_flash_t* *chip)

Check if appropriate chip driver is set.

参数 **chip** -- Pointer to SPI flash chip to use. If NULL, esp_flash_default_chip is substituted.
返回 true if set, otherwise false.

esp_err_t **esp_flash_read_id**(*esp_flash_t* *chip, uint32_t *out_id)

Read flash ID via the common "RDID" SPI flash command.

ID is a 24-bit value. Lower 16 bits of 'id' are the chip ID, upper 8 bits are the manufacturer ID.

参数

- **chip** -- Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- **out_id** -- **[out]** Pointer to receive ID value.

返回 ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_get_size**(*esp_flash_t* *chip, uint32_t *out_size)

Detect flash size based on flash ID.

备注： 1. Most flash chips use a common format for flash ID, where the lower 4 bits specify the size as a power of 2. If the manufacturer doesn't follow this convention, the size may be incorrectly detected.

a. The out_size returned only stands for The out_size stands for the size in the binary image header. If you want to get the real size of the chip, please call `esp_flash_get_physical_size` instead.

参数

- **chip** -- Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- **out_size** -- **[out]** Detected size in bytes, standing for the size in the binary image header.

返回 ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_get_physical_size**(*esp_flash_t* *chip, uint32_t *flash_size)

Detect flash size based on flash ID.

备注： Most flash chips use a common format for flash ID, where the lower 4 bits specify the size as a power of 2. If the manufacturer doesn't follow this convention, the size may be incorrectly detected.

参数

- **chip** -- Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- **flash_size** -- **[out]** Detected size in bytes.

返回 ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_read_unique_chip_id**(*esp_flash_t* *chip, uint64_t *out_id)

Read flash unique ID via the common "RDUID" SPI flash command.

ID is a 64-bit value.

备注: This is an optional feature, which is not supported on all flash chips. READ PROGRAMMING GUIDE FIRST!

参数

- **chip** -- Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init().
- **out_id** -- **[out]** Pointer to receive unique ID value.

返回

- ESP_OK on success, or a flash error code if operation failed.
- ESP_ERR_NOT_SUPPORTED if the chip doesn't support read id.

esp_err_t **esp_flash_erase_chip**(*esp_flash_t* *chip)

Erase flash chip contents.

参数 **chip** -- Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()

返回

- ESP_OK on success,
- ESP_ERR_NOT_SUPPORTED if the chip is not able to perform the operation. This is indicated by WREN = 1 after the command is sent.
- ESP_ERR_NOT_ALLOWED if a read-only partition is present.
- Other flash error code if operation failed.

esp_err_t **esp_flash_erase_region**(*esp_flash_t* *chip, uint32_t start, uint32_t len)

Erase a region of the flash chip.

Sector size is specifyed in chip->drv->sector_size field (typically 4096 bytes.) ESP_ERR_INVALID_ARG will be returned if the start & length are not a multiple of this size.

Erase is performed using block (multi-sector) erases where possible (block size is specified in chip->drv->block_erase_size field, typically 65536 bytes). Remaining sectors are erased using individual sector erase commands.

参数

- **chip** -- Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialised via esp_flash_init()
- **start** -- Address to start erasing flash. Must be sector aligned.
- **len** -- Length of region to erase. Must also be sector aligned.

返回

- ESP_OK on success,
- ESP_ERR_NOT_SUPPORTED if the chip is not able to perform the operation. This is indicated by WREN = 1 after the command is sent.
- ESP_ERR_NOT_ALLOWED if the address range (start –start + len) overlaps with a read-only partition address space
- Other flash error code if operation failed.

esp_err_t **esp_flash_get_chip_write_protect**(*esp_flash_t* *chip, bool *write_protected)

Read if the entire chip is write protected.

备注： A correct result for this flag depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

参数

- **chip** -- Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialised via esp_flash_init()
- **write_protected** -- **[out]** Pointer to boolean, set to the value of the write protect flag.

返回 ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_set_chip_write_protect**(*esp_flash_t* *chip, bool write_protect)

Set write protection for the SPI flash chip.

Some SPI flash chips may require a power cycle before write protect status can be cleared. Otherwise, write protection can be removed via a follow-up call to this function.

备注： Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

参数

- **chip** -- Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialised via esp_flash_init()
- **write_protect** -- Boolean value for the write protect flag

返回 ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_get_protectable_regions**(const *esp_flash_t* *chip, const *esp_flash_region_t* **out_regions, uint32_t *out_num_regions)

Read the list of individually protectable regions of this SPI flash chip.

备注： Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

参数

- **chip** -- Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- **out_regions** -- **[out]** Pointer to receive a pointer to the array of protectable regions of the chip.
- **out_num_regions** -- **[out]** Pointer to an integer receiving the count of protectable regions in the array returned in 'regions'.

返回 ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_get_protected_region**(*esp_flash_t* *chip, const *esp_flash_region_t* *region, bool *out_protected)

Detect if a region of the SPI flash chip is protected.

备注： It is possible for this result to be false and write operations to still fail, if protection is enabled for the entire chip.

备注： Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

参数

- **chip** -- Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- **region** -- Pointer to a struct describing a protected region. This must match one of the regions returned from esp_flash_get_protectable_regions(...).
- **out_protected** -- **[out]** Pointer to a flag which is set based on the protected status for this region.

返回 ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_set_protected_region**(*esp_flash_t* *chip, const *esp_flash_region_t* *region, bool protect)

Update the protected status for a region of the SPI flash chip.

备注： It is possible for the region protection flag to be cleared and write operations to still fail, if protection is enabled for the entire chip.

备注： Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

参数

- **chip** -- Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- **region** -- Pointer to a struct describing a protected region. This must match one of the regions returned from esp_flash_get_protectable_regions(...).
- **protect** -- Write protection flag to set.

返回 ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_read**(*esp_flash_t* *chip, void *buffer, uint32_t address, uint32_t length)

Read data from the SPI flash chip.

There are no alignment constraints on buffer, address or length.

备注： If on-chip flash encryption is used, this function returns raw (ie encrypted) data. Use the flash cache to transparently decrypt data.

参数

- **chip** -- Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialised via esp_flash_init()
- **buffer** -- Pointer to a buffer where the data will be read. To get better performance, this should be in the DRAM and word aligned.
- **address** -- Address on flash to read from. Must be less than chip->size field.
- **length** -- Length (in bytes) of data to read.

返回

- ESP_OK: success
- ESP_ERR_NO_MEM: Buffer is in external PSRAM which cannot be concurrently accessed, and a temporary internal buffer could not be allocated.
- or a flash error code if operation failed.

esp_err_t **esp_flash_write**(*esp_flash_t* *chip, const void *buffer, uint32_t address, uint32_t length)

Write data to the SPI flash chip.

There are no alignment constraints on buffer, address or length.

参数

- **chip** -- Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialised via esp_flash_init()
- **address** -- Address on flash to write to. Must be previously erased (SPI NOR flash can only write bits 1->0).
- **buffer** -- Pointer to a buffer with the data to write. To get better performance, this should be in the DRAM and word aligned.
- **length** -- Length (in bytes) of data to write.

返回

- ESP_OK on success
- ESP_FAIL, bad write, this will be detected only when CONFIG_SPI_FLASH_VERIFY_WRITE is enabled
- ESP_ERR_NOT_SUPPORTED if the chip is not able to perform the operation. This is indicated by WREN = 1 after the command is sent.
- ESP_ERR_NOT_ALLOWED if the address range (address –address + length) overlaps with a read-only partition address space
- Other flash error code if operation failed.

esp_err_t **esp_flash_write_encrypted**(*esp_flash_t* *chip, uint32_t address, const void *buffer, uint32_t length)

Encrypted and write data to the SPI flash chip using on-chip hardware flash encryption.

备注: Both address & length must be 16 byte aligned, as this is the encryption block size

参数

- **chip** -- Pointer to identify flash chip. Must be NULL (the main flash chip). For other chips, encrypted write is not supported.
- **address** -- Address on flash to write to. 16 byte aligned. Must be previously erased (SPI NOR flash can only write bits 1->0).
- **buffer** -- Pointer to a buffer with the data to write.
- **length** -- Length (in bytes) of data to write. 16 byte aligned.

返回

- ESP_OK: on success
- ESP_FAIL: bad write, this will be detected only when CONFIG_SPI_FLASH_VERIFY_WRITE is enabled
- ESP_ERR_NOT_SUPPORTED: encrypted write not supported for this chip.
- ESP_ERR_INVALID_ARG: Either the address, buffer or length is invalid.
- ESP_ERR_NOT_ALLOWED if the address range (address –address + length) overlaps with a read-only partition address space

esp_err_t **esp_flash_read_encrypted**(*esp_flash_t* *chip, uint32_t address, void *out_buffer, uint32_t length)

Read and decrypt data from the SPI flash chip using on-chip hardware flash encryption.

参数

- **chip** -- Pointer to identify flash chip. Must be NULL (the main flash chip). For other chips, encrypted read is not supported.
- **address** -- Address on flash to read from.
- **out_buffer** -- Pointer to a buffer for the data to read to.
- **length** -- Length (in bytes) of data to read.

返回

- ESP_OK: on success
- ESP_ERR_NOT_SUPPORTED: encrypted read not supported for this chip.

static inline bool **esp_flash_is_quad_mode**(const *esp_flash_t* *chip)

Returns true if chip is configured for Quad I/O or Quad Fast Read.

参数 **chip** -- Pointer to SPI flash chip to use. If NULL, esp_flash_default_chip is substituted.
返回 true if flash works in quad mode, otherwise false

Structures

struct **esp_flash_region_t**

Structure for describing a region of flash.

Public Members

uint32_t **offset**

Start address of this region.

uint32_t **size**

Size of the region.

struct **esp_flash_os_functions_t**

OS-level integration hooks for accessing flash chips inside a running OS.

It's in the public header because some instances should be allocated statically in the startup code. May be updated according to hardware version and new flash chip feature requirements, shouldn't be treated as public API.

For advanced developers, you may replace some of them with your implementations at your own risk.

Public Members

esp_err_t (***start**)(void *arg)

Called before commencing any flash operation. Does not need to be recursive (ie is called at most once for each call to 'end').

esp_err_t (***end**)(void *arg)

Called after completing any flash operation.

esp_err_t (***region_protected**)(void *arg, size_t start_addr, size_t size)

Called before any erase/write operations to check whether the region is limited by the OS

esp_err_t (***delay_us**)(void *arg, uint32_t us)

Delay for at least 'us' microseconds. Called in between 'start' and 'end'.

void *(***get_temp_buffer**)(void *arg, size_t reqest_size, size_t *out_size)

Called for get temp buffer when buffer from application cannot be directly read into/write from.

void (***release_temp_buffer**)(void *arg, void *temp_buf)

Called for release temp buffer.

esp_err_t (***check_yield**)(void *arg, uint32_t chip_status, uint32_t *out_request)

Yield to other tasks. Called during erase operations.

Return ESP_OK means yield needs to be called (got an event to handle), while ESP_ERR_TIMEOUT means skip yield.

esp_err_t (***yield**)(void *arg, uint32_t *out_status)

Yield to other tasks. Called during erase operations.

int64_t (***get_system_time**)(void *arg)

Called for get system time.

void (***set_flash_op_status**)(uint32_t op_status)

Call to set flash operation status

struct **esp_flash_t**

Structure to describe a SPI flash chip connected to the system.

Structure must be initialized before use (passed to esp_flash_init()). It's in the public header because some instances should be allocated statically in the startup code. May be updated according to hardware version and new flash chip feature requirements, shouldn't be treated as public API.

For advanced developers, you may replace some of them with your implementations at your own risk.

Public Members

spi_flash_host_inst_t ***host**

Pointer to hardware-specific "host_driver" structure. Must be initialized before used.

const *spi_flash_chip_t* ***chip_drv**

Pointer to chip-model-specific "adapter" structure. If NULL, will be detected during initialisation.

const *esp_flash_os_functions_t* ***os_func**

Pointer to os-specific hook structure. Call `esp_flash_init_os_functions()` to setup this field, after the host is properly initialized.

void ***os_func_data**

Pointer to argument for os-specific hooks. Left NULL and will be initialized with `os_func`.

esp_flash_io_mode_t **read_mode**

Configured SPI flash read mode. Set before `esp_flash_init` is called.

uint32_t **size**

Size of SPI flash in bytes. If 0, size will be detected during initialisation. Note: this stands for the size in the binary image header. If you want to get the flash physical size, please call `esp_flash_get_physical_size`.

uint32_t **chip_id**

Detected chip id.

uint32_t **busy**

This flag is used to verify chip's status.

uint32_t **hpm_dummy_ena**

This flag is used to verify whether flash works under HPM status.

uint32_t **reserved_flags**

reserved.

Macros

SPI_FLASH_YIELD_REQ_YIELD

SPI_FLASH_YIELD_REQ_SUSPEND

SPI_FLASH_YIELD_STA_RESUME

SPI_FLASH_OS_IS_ERASING_STATUS_FLAG

Type Definitions

typedef struct *spi_flash_chip_t* **spi_flash_chip_t**

Header File

- components/spi_flash/include/spi_flash_mmap.h
- This header file can be included with:

```
#include "spi_flash_mmap.h"
```

- This header file is a part of the API provided by the `spi_flash` component. To declare that your component depends on `spi_flash`, add the following to your CMakeLists.txt:

```
REQUIRES spi_flash
```

or

```
PRIV_REQUIRES spi_flash
```

Functions

esp_err_t **spi_flash_mmap**(size_t src_addr, size_t size, *spi_flash_mmap_memory_t* memory, const void **out_ptr, *spi_flash_mmap_handle_t* *out_handle)

Map region of flash memory into data or instruction address space.

This function allocates sufficient number of 64kB MMU pages and configures them to map the requested region of flash memory into the address space. It may reuse MMU pages which already provide the required mapping.

As with any allocator, if mmap/munmap are heavily used then the address space may become fragmented. To troubleshoot issues with page allocation, use spi_flash_mmap_dump() function.

参数

- **src_addr** -- Physical address in flash where requested region starts. This address *must* be aligned to 64kB boundary (SPI_FLASH_MMU_PAGE_SIZE)
- **size** -- Size of region to be mapped. This size will be rounded up to a 64kB boundary
- **memory** -- Address space where the region should be mapped (data or instruction)

- **out_ptr** -- **[out]** Output, pointer to the mapped memory region
- **out_handle** -- **[out]** Output, handle which should be used for spi_flash_munmap call

返回 ESP_OK on success, ESP_ERR_NO_MEM if pages can not be allocated

esp_err_t **spi_flash_mmap_pages**(const int *pages, size_t page_count, *spi_flash_mmap_memory_t* memory, const void **out_ptr, *spi_flash_mmap_handle_t* *out_handle)

Map sequences of pages of flash memory into data or instruction address space.

This function allocates sufficient number of 64kB MMU pages and configures them to map the indicated pages of flash memory contiguously into address space. In this respect, it works in a similar way as spi_flash_mmap() but it allows mapping a (maybe non-contiguous) set of pages into a contiguous region of memory.

参数

- **pages** -- An array of numbers indicating the 64kB pages in flash to be mapped contiguously into memory. These indicate the indexes of the 64kB pages, not the byte-size addresses as used in other functions. Array must be located in internal memory.
- **page_count** -- Number of entries in the pages array
- **memory** -- Address space where the region should be mapped (instruction or data)
- **out_ptr** -- **[out]** Output, pointer to the mapped memory region
- **out_handle** -- **[out]** Output, handle which should be used for spi_flash_munmap call

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if pages can not be allocated
- ESP_ERR_INVALID_ARG if pagecount is zero or pages array is not in internal memory

void **spi_flash_munmap**(*spi_flash_mmap_handle_t* handle)

Release region previously obtained using spi_flash_mmap.

备注: Calling this function will not necessarily unmap memory region. Region will only be unmapped when there are no other handles which reference this region. In case of partially overlapping regions it is possible that memory will be unmapped partially.

参数 **handle** -- Handle obtained from spi_flash_mmap

void **spi_flash_mmap_dump**(void)

Display information about mapped regions.

This function lists handles obtained using spi_flash_mmap, along with range of pages allocated to each handle. It also lists all non-zero entries of MMU table and corresponding reference counts.

uint32_t **spi_flash_mmap_get_free_pages**(*spi_flash_mmap_memory_t* memory)

get free pages number which can be mmap

This function will return number of free pages available in mmu table. This could be useful before calling actual spi_flash_mmap (maps flash range to DCache or ICache memory) to check if there is sufficient space available for mapping.

参数 **memory** -- memory type of MMU table free page
返回 number of free pages which can be mmaped

size_t **spi_flash_cache2phys**(const void *cached)

Given a memory address where flash is mapped, return the corresponding physical flash offset.

Cache address does not have have been assigned via spi_flash_mmap(), any address in memory mapped flash space can be looked up.

参数 **cached** -- Pointer to flashed cached memory.
返回

- SPI_FLASH_CACHE2PHYS_FAIL If cache address is outside flash cache region, or the address is not mapped.
- Otherwise, returns physical offset in flash

const void *`spi_flash_phys2cache`(size_t phys_offs, *spi_flash_mmap_memory_t* memory)

Given a physical offset in flash, return the address where it is mapped in the memory space.

Physical address does not have to have been assigned via spi_flash_mmap(), any address in flash can be looked up.

备注： Only the first matching cache address is returned. If MMU flash cache table is configured so multiple entries point to the same physical address, there may be more than one cache address corresponding to that physical address. It is also possible for a single physical address to be mapped to both the IROM and DROM regions.

备 注： This function doesn't impose any alignment constraints, but if memory argument is SPI_FLASH_MMAP_INST and phys_offs is not 4-byte aligned, then reading from the returned pointer will result in a crash.

参数

- **`phys_offs`** -- Physical offset in flash memory to look up.
- **`memory`** -- Address space type to look up a flash cache address mapping for (instruction or data)

返回

- NULL if the physical address is invalid or not mapped to flash cache of the specified memory type.
- Cached memory address (in IROM or DROM space) corresponding to phys_offs.

Macros

`ESP_ERR_FLASH_OP_FAIL`

This file contains `spi_flash_mmap_xx` APIs, mainly for doing memory mapping to an SPI0-connected external Flash, as well as some helper functions to convert between virtual and physical address

`ESP_ERR_FLASH_OP_TIMEOUT`

`SPI_FLASH_SEC_SIZE`

SPI Flash sector size

`SPI_FLASH_MMU_PAGE_SIZE`

Flash cache MMU mapping page size

`SPI_FLASH_CACHE2PHYS_FAIL`

Type Definitions

typedef uint32_t **`spi_flash_mmap_handle_t`**

Opaque handle for memory region obtained from spi_flash_mmap.

Enumerations

enum **`spi_flash_mmap_memory_t`**

Enumeration which specifies memory space requested in an mmap call.

Values:

enumerator **SPI_FLASH_MMAP_DATA**

map to data memory, allows byte-aligned access

enumerator **SPI_FLASH_MMAP_INST**

map to instruction memory, allows only 4-byte-aligned access

Header File

- components/hal/include/hal/spi_flash_types.h
- This header file can be included with:

```
#include "hal/spi_flash_types.h"
```

Structures

struct **spi_flash_trans_t**

Definition of a common transaction. Also holds the return value.

Public Members

uint8_t **reserved**

Reserved, must be 0.

uint8_t **mosi_len**

Output data length, in bytes.

uint8_t **miso_len**

Input data length, in bytes.

uint8_t **address_bitlen**

Length of address in bits, set to 0 if command does not need an address.

uint32_t **address**

Address to perform operation on.

const uint8_t ***mosi_data**

Output data to salve.

uint8_t ***miso_data**

[out] Input data from slave, little endian

uint32_t **flags**

Flags for this transaction. Set to 0 for now.

uint16_t **command**

Command to send.

uint8_t **dummy_bitlen**

Basic dummy bits to use.

uint32_t **io_mode**

Flash working mode when SPI_FLASH_IGNORE_BASEIO is specified.

struct **spi_flash_sus_cmd_conf**

Configuration structure for the flash chip suspend feature.

Public Members

uint32_t **sus_mask**

SUS/SUS1/SUS2 bit in flash register.

uint32_t **cmd_rdsr**

Read flash status register(2) command.

uint32_t **sus_cmd**

Flash suspend command.

uint32_t **res_cmd**

Flash resume command.

uint32_t **reserved**

Reserved, set to 0.

struct **spi_flash_encryption_t**

Structure for flash encryption operations.

Public Members

void (***flash_encryption_enable**)(void)

Enable the flash encryption.

void (***flash_encryption_disable**)(void)

Disable the flash encryption.

void (***flash_encryption_data_prepare**)(uint32_t address, const uint32_t *buffer, uint32_t size)

Prepare flash encryption before operation.

备注： address and buffer must be 8-word aligned.

Param address The destination address in flash for the write operation.
Param buffer Data for programming
Param size Size to program.

void (***flash_encryption_done**)(void)

flash data encryption operation is done.

void (*`flash_encryption_destroy`)(void)

Destroy encrypted result

bool (*`flash_encryption_check`)(uint32_t address, uint32_t length)

Check if is qualified to encrypt the buffer

Param address the address of written flash partition.
Param length Buffer size.

struct `spi_flash_host_inst_t`

SPI Flash Host driver instance

Public Members

const struct *spi_flash_host_driver_s* *`driver`

Pointer to the implementation function table.

struct `spi_flash_host_driver_s`

Host driver configuration and context structure.

Public Members

esp_err_t (*`dev_config`)(*spi_flash_host_inst_t* *host)

Configure the device-related register before transactions. This saves some time to re-configure those registers when we send continuously

esp_err_t (*`common_command`)(*spi_flash_host_inst_t* *host, *spi_flash_trans_t* *t)

Send an user-defined spi transaction to the device.

esp_err_t (*`read_id`)(*spi_flash_host_inst_t* *host, uint32_t *id)

Read flash ID.

void (*`erase_chip`)(*spi_flash_host_inst_t* *host)

Erase whole flash chip.

void (*`erase_sector`)(*spi_flash_host_inst_t* *host, uint32_t start_address)

Erase a specific sector by its start address.

void (*`erase_block`)(*spi_flash_host_inst_t* *host, uint32_t start_address)

Erase a specific block by its start address.

esp_err_t (*`read_status`)(*spi_flash_host_inst_t* *host, uint8_t *out_sr)

Read the status of the flash chip.

esp_err_t (*`set_write_protect`)(*spi_flash_host_inst_t* *host, bool wp)

Disable write protection.

void (***program_page**)(*spi_flash_host_inst_t* *host, const void *buffer, uint32_t address, uint32_t length)

Program a page of the flash. Check `max_write_bytes` for the maximum allowed writing length.

bool (***supports_direct_write**)(*spi_flash_host_inst_t* *host, const void *p)

Check whether the SPI host supports direct write.

When cache is disabled, SPI1 doesn't support directly write when buffer isn't internal.

int (***write_data_slicer**)(*spi_flash_host_inst_t* *host, uint32_t address, uint32_t len, uint32_t *align_addr, uint32_t page_size)

Slicer for write data. The `program_page` should be called iteratively with the return value of this function.

Param address Beginning flash address to write
Param len Length request to write
Param align_addr Output of the aligned address to write to
Param page_size Physical page size of the flash chip
Return Length that can be actually written in one `program_page` call

esp_err_t (***read**)(*spi_flash_host_inst_t* *host, void *buffer, uint32_t address, uint32_t read_len)

Read data from the flash. Check `max_read_bytes` for the maximum allowed reading length.

bool (***supports_direct_read**)(*spi_flash_host_inst_t* *host, const void *p)

Check whether the SPI host supports direct read.

When cache is disabled, SPI1 doesn't support directly read when the given buffer isn't internal.

int (***read_data_slicer**)(*spi_flash_host_inst_t* *host, uint32_t address, uint32_t len, uint32_t *align_addr, uint32_t page_size)

Slicer for read data. The `read` should be called iteratively with the return value of this function.

Param address Beginning flash address to read
Param len Length request to read
Param align_addr Output of the aligned address to read
Param page_size Physical page size of the flash chip
Return Length that can be actually read in one `read` call

uint32_t (***host_status**)(*spi_flash_host_inst_t* *host)

Check the host status, 0:busy, 1:idle, 2:suspended.

esp_err_t (***configure_host_io_mode**)(*spi_flash_host_inst_t* *host, uint32_t command, uint32_t addr_bitlen, int dummy_bitlen_base, *esp_flash_io_mode_t* io_mode)

Configure the host to work at different read mode. Responsible to compensate the timing and set IO mode.

void (***poll_cmd_done**)(*spi_flash_host_inst_t* *host)

Internal use, poll the HW until the last operation is done.

esp_err_t (***flush_cache**)(*spi_flash_host_inst_t* *host, uint32_t addr, uint32_t size)

For some host (SPI1), they are shared with a cache. When the data is modified, the cache needs to be flushed. Left NULL if not supported.

void (***check_suspend**)(*spi_flash_host_inst_t* *host)

Suspend check erase/program operation, reserved for ESP32-C3 and ESP32-S3 spi flash ROM IMPL.

void (***resume**)(*spi_flash_host_inst_t* *host)

Resume flash from suspend manually

void (***suspend**)(*spi_flash_host_inst_t* *host)

Set flash in suspend status manually

esp_err_t (***sus_setup**)(*spi_flash_host_inst_t* *host, const *spi_flash_sus_cmd_conf* *sus_conf)

Suspend feature setup for setting cmd and status register mask.

Macros

SPI_FLASH_TRANS_FLAG_CMD16

Send command of 16 bits.

SPI_FLASH_TRANS_FLAG_IGNORE_BASEIO

Not applying the basic io mode configuration for this transaction.

SPI_FLASH_TRANS_FLAG_BYTE_SWAP

Used for DTR mode, to swap the bytes of a pair of rising/falling edge.

SPI_FLASH_TRANS_FLAG_PE_CMD

Indicates that this transaction is to erase/program flash chip.

SPI_FLASH_CONFIG_CONF_BITS

OR the io_mode with this mask, to enable the dummy output feature or replace the first several dummy bits into address to meet the requirements of conf bits. (Used in DIO/QIO/OIO mode)

SPI_FLASH_OPI_FLAG

A flag for flash work in opi mode, the io mode below are opi, above are SPI/QSPI mode. DO NOT use this value in any API.

SPI_FLASH_READ_MODE_MIN

Slowest io mode supported by ESP32, currently SlowRd.

Type Definitions

typedef enum *esp_flash_speed_s* **esp_flash_speed_t**

SPI flash clock speed values, always refer to them by the enum rather than the actual value (more speed may be appended into the list).

A strategy to select the maximum allowed speed is to enumerate from the `ESP_FLSH_SPEED_MAX-1` or highest frequency supported by your flash, and decrease the speed until the probing success.

typedef struct *spi_flash_host_driver_s* **spi_flash_host_driver_t**

Enumerations

enum **esp_flash_speed_s**

SPI flash clock speed values, always refer to them by the enum rather than the actual value (more speed may be appended into the list).

A strategy to select the maximum allowed speed is to enumerate from the `ESP_FLSH_SPEED_MAX-1` or highest frequency supported by your flash, and decrease the speed until the probing success.

Values:

enumerator **ESP_FLASH_5MHZ**

The flash runs under 5MHz.

enumerator **ESP_FLASH_10MHZ**

The flash runs under 10MHz.

enumerator **ESP_FLASH_20MHZ**

The flash runs under 20MHz.

enumerator **ESP_FLASH_26MHZ**

The flash runs under 26MHz.

enumerator **ESP_FLASH_40MHZ**

The flash runs under 40MHz.

enumerator **ESP_FLASH_80MHZ**

The flash runs under 80MHz.

enumerator **ESP_FLASH_120MHZ**

The flash runs under 120MHz, 120MHZ can only be used by main flash after timing tuning in system. Do not use this directely in any API.

enumerator **ESP_FLASH_SPEED_MAX**

The maximum frequency supported by the host is `ESP_FLASH_SPEED_MAX-1`.

enum **esp_flash_io_mode_t**

Mode used for reading from SPI flash.

Values:

enumerator **SPI_FLASH_SLOWRD**

Data read using single I/O, some limits on speed.

enumerator **SPI_FLASH_FASTRD**

Data read using single I/O, no limit on speed.

enumerator **SPI_FLASH_DOUT**

Data read using dual I/O.

enumerator **SPI_FLASH_DIO**

Both address & data transferred using dual I/O.

enumerator **SPI_FLASH_QOUT**

Data read using quad I/O.

enumerator **SPI_FLASH_QIO**

Both address & data transferred using quad I/O.

enumerator **SPI_FLASH_OPI_STR**

Only support on OPI flash, flash read and write under STR mode.

enumerator **SPI_FLASH_OPI_DTR**

Only support on OPI flash, flash read and write under DTR mode.

enumerator **SPI_FLASH_READ_MODE_MAX**

The fastest io mode supported by the host is `ESP_FLASH_READ_MODE_MAX-1`.

Header File

- components/hal/include/hal/esp_flash_err.h
- This header file can be included with:

```
#include "hal/esp_flash_err.h"
```

Macros

ESP_ERR_FLASH_NOT_INITIALISED

esp_flash_chip_t structure not correctly initialised by esp_flash_init().

ESP_ERR_FLASH_UNSUPPORTED_HOST

Requested operation isn't supported via this host SPI bus (chip->spi field).

ESP_ERR_FLASH_UNSUPPORTED_CHIP

Requested operation isn't supported by this model of SPI flash chip.

ESP_ERR_FLASH_PROTECTED

Write operation failed due to chip's write protection being enabled.

Enumerations

enum **[anonymous]**

Values:

enumerator **ESP_ERR_FLASH_SIZE_NOT_MATCH**

The chip doesn't have enough space for the current partition table.

enumerator **ESP_ERR_FLASH_NO_RESPONSE**

Chip did not respond to the command, or timed out.

Header File

- components/spi_flash/include/esp_spi_flash_counters.h
- This header file can be included with:

```
#include "esp_spi_flash_counters.h"
```

- This header file is a part of the API provided by the `spi_flash` component. To declare that your component depends on `spi_flash`, add the following to your CMakeLists.txt:

```
REQUIRES spi_flash
```

or

```
PRIV_REQUIRES spi_flash
```

Functions

void **esp_flash_reset_counters**(void)

Reset SPI flash operation counters.

void **spi_flash_reset_counters**(void)

void **esp_flash_dump_counters**(FILE *stream)

Print SPI flash operation counters.

void **spi_flash_dump_counters**(void)

const *esp_flash_counters_t* ***esp_flash_get_counters**(void)

Return current SPI flash operation counters.

返回 pointer to the *esp_flash_counters_t* structure holding values of the operation counters

const *spi_flash_counters_t* ***spi_flash_get_counters**(void)

Structures

struct **esp_flash_counter_t**

Structure holding statistics for one type of operation

Public Members

uint32_t **count**

number of times operation was executed

uint32_t **time**

total time taken, in microseconds

uint32_t **bytes**

total number of bytes

struct **esp_flash_counters_t**

Structure for counters of flash actions

Public Members

esp_flash_counter_t **read**

counters for read action, like `esp_flash_read`

esp_flash_counter_t **write**

counters for write action, like `esp_flash_write`

esp_flash_counter_t **erase**

counters for erase action, like `esp_flash_erase`

Type Definitions

typedef *esp_flash_counter_t* **spi_flash_counter_t**

typedef *esp_flash_counters_t* **spi_flash_counters_t**

flash 加密 API 参考

Header File

- components/bootloader_support/include/esp_flash_encrypt.h
- This header file can be included with:

```
#include "esp_flash_encrypt.h"
```

- This header file is a part of the API provided by the `bootloader_support` component. To declare that your component depends on `bootloader_support`, add the following to your CMakeLists.txt:

```
REQUIRES bootloader_support
```

or

```
PRIV_REQUIRES bootloader_support
```

Functions

bool **esp_flash_encryption_enabled**(void)

Is flash encryption currently enabled in hardware?

Flash encryption is enabled if the FLASH_CRYPT_CNT efuse has an odd number of bits set.

返回 true if flash encryption is enabled.

esp_err_t **esp_flash_encrypt_check_and_update**(void)

bool **esp_flash_encrypt_state**(void)

Returns the Flash Encryption state and prints it.

返回 True - Flash Encryption is enabled False - Flash Encryption is not enabled

bool **esp_flash_encrypt_initialized_once**(void)

Checks if the first initialization was done.

If the first initialization was done then FLASH_CRYPT_CNT != 0

返回 true - the first initialization was done false - the first initialization was NOT done

esp_err_t **esp_flash_encrypt_init**(void)

The first initialization of Flash Encryption key and related eFuses.

返回 ESP_OK if all operations succeeded

esp_err_t **esp_flash_encrypt_contents**(void)

Encrypts flash content.

返回 ESP_OK if all operations succeeded

esp_err_t **esp_flash_encrypt_enable**(void)

Activates Flash encryption on the chip.

It burns FLASH_CRYPT_CNT eFuse based on the CONFIG_SECURE_FLASH_ENCRYPTION_MODE_RELEASE option.

返回 ESP_OK if all operations succeeded

bool **esp_flash_encrypt_is_write_protected**(bool print_error)

Returns True if the write protection of FLASH_CRYPT_CNT is set.

参数 **print_error** -- Print error if it is write protected
返回 true - if FLASH_CRYPT_CNT is write protected

esp_err_t **esp_flash_encrypt_region**(uint32_t src_addr, size_t data_length)

Encrypt-in-place a block of flash sectors.

备注: This function resets RTC_WDT between operations with sectors.

参数

- **src_addr** -- Source offset in flash. Should be multiple of 4096 bytes.
- **data_length** -- Length of data to encrypt in bytes. Will be rounded up to next multiple of 4096 bytes.

返回 ESP_OK if all operations succeeded, ESP_ERR_FLASH_OP_FAIL if SPI flash fails, ESP_ERR_FLASH_OP_TIMEOUT if flash times out.

void **esp_flash_write_protect_crypt_cnt**(void)

Write protect FLASH_CRYPT_CNT.

Intended to be called as a part of boot process if flash encryption is enabled but secure boot is not used. This should protect against serial re-flashing of an unauthorised code in absence of secure boot.

备注: On ESP32 V3 only, write protecting FLASH_CRYPT_CNT will also prevent disabling UART Download Mode. If both are wanted, call esp_efuse_disable_rom_download_mode() before calling this function.

esp_flash_enc_mode_t **esp_get_flash_encryption_mode**(void)

Return the flash encryption mode.

The API is called during boot process but can also be called by application to check the current flash encryption mode of ESP32

返回

void **esp_flash_encryption_init_checks**(void)

Check the flash encryption mode during startup.

Verifies the flash encryption config during startup:

- Correct any insecure flash encryption settings if hardware Secure Boot is enabled.
- Log warnings if the efuse config doesn't match the project config in any way

备注: This function is called automatically during app startup, it doesn't need to be called from the app.

bool **esp_flash_encryption_cfg_verify_release_mode**(void)

Returns the verification status for all physical security features of flash encryption in release mode.

If the device has flash encryption feature configured in the release mode, then it is highly recommended to call this API in the application startup code. This API verifies the sanity of the eFuse configuration against the release (production) mode of the flash encryption feature.

返回

- True - all eFuses are configured correctly
- False - not all eFuses are configured correctly.

void **esp_flash_encryption_set_release_mode**(void)

Switches Flash Encryption from ”Development” to ”Release”.

If already in ”Release” mode, the function will do nothing. If flash encryption efuse is not enabled yet then abort. It burns:

- ”disable encrypt in dl mode”
- set FLASH_CRYPT_CNT efuse to max

Enumerations

enum **esp_flash_enc_mode_t**

Values:

enumerator **ESP_FLASH_ENC_MODE_DISABLED**

enumerator **ESP_FLASH_ENC_MODE_DEVELOPMENT**

enumerator **ESP_FLASH_ENC_MODE_RELEASE**

2.5.29 SPI 主机驱动程序

SPI 主机驱动程序是一个软件程序，用于在 ESP32-P4 的通用 SPI (GP-SPI) 外设工作在主控模式时，对其进行控制。

有关 GP-SPI 硬件相关信息，请参考 **ESP32-P4 技术参考手册 > SPI 控制器** [PDF]。

术语

下表为 SPI 主机驱动的相关术语。

术语	定义
主机 (Host)	ESP32-P4 内置的 SPI 控制器外设。用作 SPI 主机，在总线上发起 SPI 传输。
设备 (Device)	SPI 从机设备。一条 SPI 总线与一或多个设备连接。每个设备共享 MOSI、MISO 和 SCLK 信号，但只有当主机向设备的专属 CS 线发出信号时，设备才会在总线上处于激活状态。
总线 (Bus)	信号总线，由连接到同一主机的所有设备共用。一般来说，每条总线包括以下线：MISO、MOSI、SCLK、一条或多条 CS 线，以及可选的 QUADWP 和 QUADHD。因此，除每个设备都有单独的 CS 线外，所有设备都连接在相同的线下。多个设备也可以菊花链的方式共享一条 CS 线。
MOSI	主机输出，从机输入，也写作 D。数据从主机发送至设备。在 Octal/OPI 模式下也表示为 data0 信号。
MISO	主机输入，从机输出，也写作 Q。数据从设备发送至主机。在 Octal/OPI 模式下也表示为 data1 信号。
SCLK	串行时钟。由主机产生的振荡信号，使数据位的传输保持同步。
CS	片选。允许主机选择连接到总线上的单个设备，以便发送或接收数据。
QUADWP	写保护信号。只用于 4 位 (qio/qout) 传输。在 Octal/OPI 模式下也表示为 data2 信号。
QUADHD	保持信号。只用于 4 位 (qio/qout) 传输。在 Octal/OPI 模式下也表示为 data3 信号。
DATA4	在 Octal/OPI 模式下表示为 data4 信号。
DATA5	在 Octal/OPI 模式下表示为 data5 信号。
DATA6	在 Octal/OPI 模式下表示为 data6 信号。
DATA7	在 Octal/OPI 模式下表示为 data7 信号。
断言 (Assertion)	指激活一条线路的操作。
去断言 (De-assertion)	指将线路恢复到非活动状态（回到空闲状态）的操作。
传输事务 (Transaction)	即主机断言设备的 CS 线，向设备发送数据/从设备读取数据，接着去断言 CS 线的过程。传输事务为原子操作，不可打断。
发射沿 (Launch Edge)	源寄存器将信号 **发射**到线路上的时钟边沿。
锁存沿 (Latch Edge)	目的寄存器 **锁存**信号的时钟边沿。

主机驱动特性

SPI 主机驱动程序负责管理主机与设备间的通信，具有以下特性：

- 支持多线程环境使用
- 读写数据过程中 DMA 透明传输
- 同一信号总线上不同设备的数据可自动时分复用，请参阅*SPI 总线锁*。

警告： SPI 主机驱动允许总线上连接多个设备（共享单个 ESP32-P4 SPI 外设）。每个设备仅由一个任务访问时，驱动程序线程安全。反之，若多个任务尝试访问同一 SPI 设备，则驱动程序 **非线程安全**。此时，建议执行以下任一操作：

- 重构应用程序，确保每个 SPI 外设在同一时间仅由一个任务访问。使用`spi_bus_config_t::isr_cpu_id` 将 SPI ISR 注册到与 SPI 外设相关任务相同的内核，以确保线程安全。
- 使用`xSemaphoreCreateMutex` 为共享设备添加互斥锁。

SPI 传输事务

SPI 总线传输事务由五个阶段构成，详见下表（任意阶段均可跳过）。

阶段名称	描述
命令阶段 (Command)	在此阶段，主机向总线发送命令字段，长度为 0-16 位。
地址阶段 (Address)	在此阶段，主机向总线发送地址字段，长度为 0-32 位。
Dummy 阶段	此阶段可自行配置，用于适配时序要求。
写入阶段 (Write)	此阶段主机向设备传输数据，这些数据在紧随命令阶段（可选）和地址阶段（可选）之后。从电平的角度来看，数据与命令没有区别。
读取阶段 (Read)	此阶段主机读取设备数据。

传输事务属性由总线配置结构体*spi_bus_config_t*、设备配置结构体*spi_device_interface_config_t* 和传输事务配置结构体*spi_transaction_t* 共同决定。

一个 SPI 主机可以发送全双工传输事务，此时读取和写入阶段同步进行。传输事务总长度取决于以下结构体成员长度总和：

- *spi_device_interface_config_t::command_bits*
- *spi_device_interface_config_t::address_bits*
- *spi_transaction_t::length*

而*spi_transaction_t::rxlength* 则决定了接收到的数据包长度。

在半双工传输事务中，读取和写入阶段独立进行（一次一个方向）。写入和读取阶段的长度由*spi_transaction_t::length* 和*spi_transaction_t::rxlength* 分别决定。

并非每个 SPI 设备都要求命令和/或地址，因此命令阶段和地址阶段为可选项。这反映在设备的配置中：如果*spi_device_interface_config_t::command_bits* 和/或*spi_device_interface_config_t::address_bits* 被设置为零，则不会唤起命令或地址阶段。

并非每个传输事务都需要写入和读取数据，因此读取和写入阶段也是可选项。如果将*spi_transaction_t::rx_buffer* 设置为 NULL，且未设置*SPI_TRANS_USE_RXDATA*，读取阶段将被跳过。如果将*spi_transaction_t::tx_buffer* 设置为 NULL，且未设置*SPI_TRANS_USE_TXDATA*，写入阶段将被跳过。

主机驱动程序支持两种类型的传输事务：中断传输事务和轮询传输事务。用户可以选择在不同设备上使用不同的传输事务类型。若设备需要同时使用两种传输事务类型，请参阅向同一设备发送混合传输事务的注意事项。

中断传输事务 中断传输事务将阻塞传输事务程序，直至传输事务完成，以使 CPU 运行其他任务程序。

应用任务中可以将多个传输事务加入到队列中，驱动程序将在中断服务程序 (ISR) 中自动逐一发送队列中的数据。在所有传输事务完成以前，任务可切换到其他程序中。

轮询传输事务 轮询传输事务不依赖于中断，程序将不断轮询 SPI 主机的状态位，直到传输事务完成。

所有执行中断传输事务的任务都可能被队列阻塞。在此情况下，用户需要等待 ISR 和传输事务传输完成。轮询传输事务节省了原本用于队列处理和上下文切换的时间，减少了传输事务持续时间。传输事务类型的缺点是在这些事务进行期间，CPU 将被占用而处于忙碌状态。

传输事务完成后，*spi_device_polling_end()* 程序需要至少 1 µs 的时间来解除阻塞其他任务。此处强烈建议调用函数*spi_device_acquire_bus()* 和*spi_device_release_bus()* 来打包一系列轮询传输事务以避免开销。详情请参阅获取总线。

传输线模式配置 ESP32-P4 支持的线路模式如下。要使用这些模式，请在结构体*spi_transaction_t* 中设置 flags，如传输事务标志信号一栏所示。要检查相应的 IO 管脚是否被设置，请在*spi_bus_config_t* 中设置 flags，如总线 IO 设置标志信号一栏所示。

模式	命令位宽	地址位宽	数据位宽	传输事务标志信号	总线 IO 设置标志信号
普通 SPI 模式	1	1	1	0	0
双线输出模式	1	1	2	SPI_TRANS_MODE_DIO	SPICOMMON_BUSFLAG_DUAL
双线 I/O 模式	1	2	2	SPI_TRANS_MODE_DIO | SPI_TRANS_MULTILINE_ADDR	SPICOMMON_BUSFLAG_DUAL
四线输出模式	1	1	4	SPI_TRANS_MODE_QIO	SPICOMMON_BUSFLAG_QUAD
四线 I/O 模式	1	4	4	SPI_TRANS_MODE_QIO | SPI_TRANS_MULTILINE_ADDR	SPICOMMON_BUSFLAG_QUAD
八线输出模式	1	1	8	SPI_TRANS_MODE_OCT	SPICOMMON_BUSFLAG_OCTAL
OPI 模式	8	8	8	SPI_TRANS_MODE_OCT | SPI_TRANS_MULTILINE_ADDR | SPI_TRANS_MULTILINE_CMD	SPICOMMON_BUSFLAG_OCTAL

命令阶段和地址阶段 在命令阶段和地址阶段，*spi_transaction_t::cmd* 和*spi_transaction_t::addr* 将被发送到总线，该过程中无数据读取。命令阶段和地址阶段的默认长度通过调用*spi_bus_add_device()* 在*spi_device_interface_config_t* 中设置。如果*spi_transaction_t::flags* 中的标志信号*SPI_TRANS_VARIABLE_CMD* 和*SPI_TRANS_VARIABLE_ADDR* 未设置，则驱动程序将在设备初始化期间自动将这些阶段的长度设置为默认值。

如需更改命令阶段和地址阶段的长度，可通过以下步骤实现：声明结构体*spi_transaction_ext_t*，在*spi_transaction_ext_t::base* 中设置标志信号*SPI_TRANS_VARIABLE_CMD* 和/或*SPI_TRANS_VARIABLE_ADDR*，随后按正常步骤完成余下配置。这样一来，各阶段的长度将等于结构体*spi_transaction_ext_t* 中设置的*spi_transaction_ext_t::command_bits* 和*spi_transaction_ext_t::address_bits* 长度。

如果需要命令阶段和地址阶段的线数与数据阶段保持一致，则应在结构体*spi_transaction_t* 中将 SPI_TRANS_MULTILINE_CMD 和/或 SPI_TRANS_MULTILINE_ADDR 设置进该结构体的 flags 成员变量。请参阅传输线模式配置。

写入和读取阶段 一般而言，需要传输到设备或由设备读取的数据将由*spi_transaction_t::rx_buffer* 和*spi_transaction_t::tx_buffer* 指向的内存块中读取或写入。如果传输时启用了 DMA，则缓冲区应：

1. 申请支持 DMA 的内存。具体操作请参阅支持 *DMA* 的内存。
2. 32 位对齐（从 32 位边界开始，长度为 4 字节的倍数）。

若未满足以上要求，传输事务效率将受到临时缓冲区分配和复制的影响。

如果使用多条数据线传输，请在结构体*spi_device_interface_config_t* 中的 flags 设置 SPI_DEVICE_HALFDUPLEX 标志信号。结构体*spi_transaction_t* 中的 flags 应按照传输线模式配置 中的描述设置。

备注： 不支持同时具有读取和写入阶段的半双工传输事务。请使用全双工模式。

获取总线 若需连续发送专门的 SPI 传输事务以提高效率，可采用获取总线的方式。获取总线后，与其他设备间的传输事务（包括轮询传输事务或中断传输事务）将处于待处理状态，直到总线被释放。要获取和释放总线，请调用函数*spi_device_acquire_bus()* 和*spi_device_release_bus()*。

使用驱动程序

- 通过调用函数*spi_bus_initialize()* 初始化 SPI 总线。确保在结构体*spi_bus_config_t* 中设置正确的 I/O 管脚。不需要的信号设置为 -1。
- 通过调用函数*spi_bus_add_device()* 注册连接到总线的设备。确保用参数 *dev_config* 配置设备可能需要的任何时序要求。完成上述操作后可获得设备句柄，以便在向设备发送传输事务时使用。
- 要与设备交互，请在一个或多个*spi_transaction_t* 结构体中填充所需的传输事务参数，随后使用轮询传输事务或中断传输事务发送这些结构体：
  - **中断传输事务** 调用函数*spi_device_queue_trans()* 将传输事务添加到队列中，随后使用函数*spi_device_get_trans_result()* 查询结果，或将所有请求输入*spi_device_transmit()* 实现同步处理。
  - **轮询传输事务** 调用函数*spi_device_polling_transmit()* 发送轮询传输事务。若有插入内容的需要，也可使用*spi_device_polling_start()* 和*spi_device_polling_end()* 发送传输事务。
- （可选）若要向设备发送背对背传输事务，请在发送传输事务前调用函数*spi_device_acquire_bus()*，并在发送传输事务后调用函数*spi_device_release_bus()*。
- （可选）要从总线上移除特定设备，请以设备句柄为参数调用函数*spi_bus_remove_device()*。
- （可选）要复位（或重置）总线，请确保总线上未连接其他设备，并调用函数*spi_bus_free()*。

SPI 主机驱动程序的示例代码存放在 ESP-IDF 示例项目的 peripherals/spi_master 目录下。

传输数据小于 32 位的传输事务 当传输事务数据等于或小于 32 位时，为数据分配一个缓冲区将是次优的选择。实际上，数据可以直接存储于传输事务结构体中。对已传输的数据，可通过调用函数*spi_transaction_t::tx_data* 并在传输时设置*SPI_TRANS_USE_TXDATA* 标志信号来实现。对已接收的数据，可通过调用函数*spi_transaction_t::rx_data* 并设置*SPI_TRANS_USE_RXDATA* 来实现。在这两种情况下，请勿修改*spi_transaction_t::tx_buffer* 或*spi_transaction_t::rx_buffer*，因为它们与*spi_transaction_t::tx_data* 和*spi_transaction_t::rx_data* 的内存位置相同。

非 uint8_t 的整数传输事务 SPI 主机逐字节地将数据读入和写入内存。默认情况下，数据优先以最高有效位 (MSB) 发送，极少数情况下会优先使用最低有效位 (LSB)。如果需要发送一个小于 8 位的值，这些位应以 MSB 优先的方式写入内存。

例如，如果需要发送 0b00010，则应将其写成 uint8_t 变量，读取长度设置为 5 位。此时，设备仍然会收到 8 位数据，并另有 3 个“随机”位，所以读取过程必须准确。

此外，ESP32-P4 属于小端芯片，即 uint16_t 和 uint32_t 变量的最低有效位存储在最小的地址。因此，如果 uint16_t 存储在内存中，则首先发送位 [7:0]，其次是位 [15:8]。

在某些情况下，要传输的数据大小与 uint8_t 数组不同，可使用以下宏将数据转换为可由 SPI 驱动直接发送的格式：

- 需传输的数据，使用*SPI_SWAP_DATA_TX*
- 接收到的数据，使用*SPI_SWAP_DATA_RX*

向同一设备发送混合传输事务的注意事项 为避免代码过于复杂，请一次只向单个设备发送一种类型的传输事务（中断或轮询）。如有需要，可交替发送中断传输事务和轮询传输事务，详见下文。

所有的轮询传输事务和中断传输事务完成后，方可发送轮询传输事务。

由于未完成的轮询传输事务将阻塞其他传输事务，请务必在*spi_device_polling_start()* 之后调用函数*spi_device_polling_end()* 以允许其他事务或其他设备使用总线。如果在轮询过程中无需切换到其他任务，可调用函数*spi_device_polling_transmit()* 启动传输事务，则该传输事务将自动结束。

ISR 会干扰飞行中的轮询传输事务，以适应中断传输事务。在调用*spi_device_polling_start()* 前，需确保所有发送到 ISR 的中断传输事务已经完成。为此可持续调用*spi_device_get_trans_result()*，直至全部传输事务返回。

为更好地控制函数的调用顺序，只在单个任务中向同一设备发送混合传输事务。

GPIO 矩阵与 IO_MUX 管脚 芯片的大多数外围信号都与之专用的 IO_MUX 管脚连接，但这些信号也可以通过较不直接的 GPIO 矩阵路由到任何其他可用的管脚。只要有一个信号是通过 GPIO 矩阵路由的，那么所有的信号都将通过它路由。

当 SPI 主机被设置为 80 MHz 或更低的频率时，通过 GPIO 矩阵路由 SPI 管脚的行为将与通过 IOMUX 路由相同。

SPI 总线的 IO_MUX 管脚如下表所示。

管脚名称	GPIO 编号 (SPI2)
CS0[1]	7
SCLK	9
MISO	10
MOSI	8
QUADWP	11
QUADHD	6

传输速度的影响因素

传输速度主要有以下三个限制因素

- 传输事务间隔时间
- SPI 时钟频率
- 缓存缺失的 SPI 函数，包括回调

影响大传输事务传输速度的主要参数是时钟频率。而多个小传输事务的传输速度主要由传输事务间隔时长决定。

传输事务持续时间 传输事务持续时间包括设置 SPI 外设寄存器，将数据复制到 FIFO 或设置 DMA 链接，以及 SPI 传输事务时间。

中断传输事务允许附加额外的开销，以适应 FreeRTOS 队列的成本以及任务与 ISR 切换所需的时间。

对于 **中断传输事务**，CPU 可以在传输事务进行过程中切换到其他任务，这能够节约 CPU 时间，会延长传输事务持续时间，请参阅中断传输事务。对于 **轮询传输事务**，它不会阻塞任务，但允许在传输事务进行时轮询。详情请参阅轮询传输事务。

如果 DMA 被启用，每个传输事务设置链接列表需要约 2 µs。当主机传输数据时，它会自动从链接列表中读取数据。如果不启用 DMA，CPU 必须自己从 FIFO 中写入和读取每个字节。这一过程时长通常不到 2 µs，但写入和读取的传输事务长度都被限制在 64 字节。

单个字节数据的典型传输事务持续时间如下。

- 使用 DMA 的中断传输事务：N/A µs。
- 使用 CPU 的中断传输事务：N/A µs。
- 使用 DMA 的轮询传输事务：N/A µs。
- 使用 CPU 的轮询传输事务：N/A µs。

请注意，以上数据测试时，*CONFIG_SPI_MASTER_ISR_IN_IRAM* 选项处于启用状态，SPI 传输事务相关的代码放置在 IRAM 中。若关闭此选项（例如为了节省 IRAM），可能影响传输事务持续时间。

SPI 时钟频率 GPSPI 外设的时钟源可以通过设置 `spi_device_handle_t::cfg::clock_source` 选择，可用的时钟源请参阅 *`spi_clock_source_t`*。

默认情况下，驱动程序将把 `spi_device_handle_t::cfg::clock_source` 设置为 `SPI_CLK_SRC_DEFAULT`。这往往代表 GPSPI 时钟源中的最高频率，在不同的芯片中这一数值会有所不同。

[1] 只有连接到总线的第一台设备可以使用 CS0 管脚。

设备的实际时钟频率可能不完全等于所设置的数字，驱动会将其重新计算为与硬件兼容的最接近的数字，并且不超过时钟源的时钟频率。调用函数*spi_device_get_actual_freq()* 以了解驱动计算的实际频率。

写入或读取阶段的理论最大传输速度可根据下表计算：

写入/读取阶段的线宽	速度 (Bps)
1-Line	*SPI 频率 /8*
2-Line	*SPI 频率 /4*
4-Line	*SPI 频率 /2*
8-Line	*SPI 频率*

其他阶段（命令阶段、地址阶段、Dummy 阶段）的传输速度计算与此类似。

缓存缺失 默认配置只将 ISR 置于 IRAM 中。其他 SPI 相关功能，包括驱动本身和回调都可能发生缓存缺失，需等待代码从 flash 中读取。为避免缓存缺失，可参考*CONFIG_SPI_MASTER_IN_IRAM*，将整个 SPI 驱动置入 IRAM，并将整个回调及其 callee 函数一起置入 IRAM。

备注： SPI 驱动是基于 FreeRTOS 的 API 实现的，在使用*CONFIG_SPI_MASTER_IN_IRAM* 时，不得启用*CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH*。

单个中断传输事务传输 n 字节的总成本为 **20+8n/Fspi[MHz]** [µs]，故传输速度为 **n/(20+8n/Fspi)**。8 MHz 时钟速度的传输速度见下表。

频率 (MHz)	传输事务间隔 (µs)	传输事务长度 (bytes)	传输时长 (µs)	传输速度 (KBps)
8	25	1	26	38.5
8	25	8	33	242.4
8	25	16	41	490.2
8	25	64	89	719.1
8	25	128	153	836.6

传输事务长度较短时将提高传输事务间隔成本，因此应尽可能将几个短传输事务压缩成一个传输事务，以提升传输速度。

注意，ISR 在 flash 操作期间默认处于禁用状态。要在 flash 操作期间继续发送传输事务，请启用*CONFIG_SPI_MASTER_ISR_IN_IRAM*，并在*spi_bus_config_t::intr_flags* 中设置*ESP_INTR_FLAG_IRAM*。此时，flash 操作前列队的传输事务将由 ISR 并行处理。此外，每个设备的回调和它们的 `callee` 函数都应该在 IRAM 中，避免回调因缓存丢失而崩溃。详情请参阅*IRAM 安全中断处理程序*。

应用示例

查看使用 SPI 主机驱动程序在半双工模式下读取/写入 AT93C46D EEPROM（8 位模式）的示例代码，请前往 ESP-IDF 示例的 peripherals/spi_master/hd_eeprom 目录。

查看使用 SPI 主机驱动程序在全双工模式下驱动 LCD 屏幕（如 ST7789V 或 ILI9341）的示例代码，请前往 ESP-IDF 示例的 peripherals/spi_master/lcd 目录。

API 参考 - SPI Common

Header File

- components/hal/include/hal/spi_types.h
- This header file can be included with:

```
#include "hal/spi_types.h"
```

Structures

struct **spi_line_mode_t**

Line mode of SPI transaction phases: CMD, ADDR, DOUT/DIN.

Public Members

uint8_t **cmd_lines**

The line width of command phase, e.g. 2-line-cmd-phase.

uint8_t **addr_lines**

The line width of address phase, e.g. 1-line-addr-phase.

uint8_t **data_lines**

The line width of data phase, e.g. 4-line-data-phase.

Type Definitions

typedef *soc_periph_spi_clk_src_t* **spi_clock_source_t**

Type of SPI clock source.

Enumerations

enum **spi_host_device_t**

Enum with the three SPI peripherals that are software-accessible in it.

Values:

enumerator **SPI1_HOST**

SPI1.

enumerator **SPI2_HOST**

SPI2.

enumerator **SPI3_HOST**

SPI3.

enumerator **SPI_HOST_MAX**

invalid host value

enum **spi_event_t**

SPI Events.

Values:

enumerator **SPI_EV_BUF_TX**

The buffer has sent data to master.

enumerator **SPI_EV_BUF_RX**

The buffer has received data from master.

enumerator **SPI_EV_SEND_DMA_READY**

Slave has loaded its TX data buffer to the hardware (DMA).

enumerator **SPI_EV_SEND**

Master has received certain number of the data, the number is determined by Master.

enumerator **SPI_EV_RECV_DMA_READY**

Slave has loaded its RX data buffer to the hardware (DMA).

enumerator **SPI_EV_RECV**

Slave has received certain number of data from master, the number is determined by Master.

enumerator **SPI_EV_CMD9**

Received CMD9 from master.

enumerator **SPI_EV_CMDA**

Received CMDA from master.

enumerator **SPI_EV_TRANS**

A transaction has done.

enum **spi_command_t**

SPI command.

Values:

enumerator **SPI_CMD_HD_WRBUF**

enumerator **SPI_CMD_HD_RDBUF**

enumerator **SPI_CMD_HD_WRDMA**

enumerator **SPI_CMD_HD_RDDMA**

enumerator **SPI_CMD_HD_SEG_END**

enumerator **SPI_CMD_HD_EN_QPI**

enumerator **SPI_CMD_HD_WR_END**

enumerator **SPI_CMD_HD_INT0**

enumerator **SPI_CMD_HD_INT1**

enumerator **SPI_CMD_HD_INT2**

Header File

- components/esp_driver_spi/include/driver/spi_common.h
- This header file can be included with:

```
#include "driver/spi_common.h"
```

- This header file is a part of the API provided by the `esp_driver_spi` component. To declare that your component depends on `esp_driver_spi`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_spi
```

or

```
PRIV_REQUIRES esp_driver_spi
```

Functions

esp_err_t **spi_bus_initialize**(*spi_host_device_t* host_id, const *spi_bus_config_t* *bus_config, *spi_dma_chan_t* dma_chan)

Initialize a SPI bus.

警告: SPI0/1 is not supported

警告: If a DMA channel is selected, any transmit and receive buffer used should be allocated in DMA-capable memory.

警告: The ISR of SPI is always executed on the core which calls this function. Never starve the ISR on this core or the SPI transactions will not be handled.

参数

- **host_id** -- SPI peripheral that controls this bus
- **bus_config** -- Pointer to a *spi_bus_config_t* struct specifying how the host should be initialized
- **dma_chan** -- - Selecting a DMA channel for an SPI bus allows transactions on the bus with size only limited by the amount of internal memory.
  - Selecting SPI_DMA_DISABLED limits the size of transactions.
  - Set to SPI_DMA_DISABLED if only the SPI flash uses this bus.
  - Set to SPI_DMA_CH_AUTO to let the driver to allocate the DMA channel.

返回

- ESP_ERR_INVALID_ARG if configuration is invalid
- ESP_ERR_INVALID_STATE if host already is in use
- ESP_ERR_NOT_FOUND if there is no available DMA channel
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

esp_err_t **spi_bus_free**(*spi_host_device_t* host_id)

Free a SPI bus.

警告: In order for this to succeed, all devices have to be removed first.

参数 **host_id** -- SPI peripheral to free

返回

- ESP_ERR_INVALID_ARG if parameter is invalid

- ESP_ERR_INVALID_STATE if bus hasn't been initialized before, or not all devices on the bus are freed
- ESP_OK on success

Structures

struct **spi_bus_config_t**

This is a configuration structure for a SPI bus.

You can use this structure to specify the GPIO pins of the bus. Normally, the driver will use the GPIO matrix to route the signals. An exception is made when all signals either can be routed through the IO_MUX or are -1. In that case, the IO_MUX is used. On ESP32, using GPIO matrix will bring about 25ns of input delay, which may cause incorrect read for >40MHz speeds.

备注: Be advised that the slave driver does not use the quadwp/quadhd lines and fields in *spi_bus_config_t* referring to these lines will be ignored and can thus safely be left uninitialized.

Public Members

int **mosi_io_num**

GPIO pin for Master Out Slave In (=spi_d) signal, or -1 if not used.

int **data0_io_num**

GPIO pin for spi data0 signal in quad/octal mode, or -1 if not used.

int **miso_io_num**

GPIO pin for Master In Slave Out (=spi_q) signal, or -1 if not used.

int **data1_io_num**

GPIO pin for spi data1 signal in quad/octal mode, or -1 if not used.

int **sclk_io_num**

GPIO pin for SPI Clock signal, or -1 if not used.

int **quadwp_io_num**

GPIO pin for WP (Write Protect) signal, or -1 if not used.

int **data2_io_num**

GPIO pin for spi data2 signal in quad/octal mode, or -1 if not used.

int **quadhd_io_num**

GPIO pin for HD (Hold) signal, or -1 if not used.

int **data3_io_num**

GPIO pin for spi data3 signal in quad/octal mode, or -1 if not used.

int **data4_io_num**

GPIO pin for spi data4 signal in octal mode, or -1 if not used.

int **data5_io_num**

GPIO pin for spi data5 signal in octal mode, or -1 if not used.

int **data6_io_num**

GPIO pin for spi data6 signal in octal mode, or -1 if not used.

int **data7_io_num**

GPIO pin for spi data7 signal in octal mode, or -1 if not used.

int **max_transfer_sz**

Maximum transfer size, in bytes. Defaults to 4092 if 0 when DMA enabled, or to `SOC_SPI_MAXIMUM_BUFFER_SIZE` if DMA is disabled.

uint32_t **flags**

Abilities of bus to be checked by the driver. Or-ed value of `SPICOMMON_BUSFLAG_*` flags.

esp_intr_cpu_affinity_t **isr_cpu_id**

Select cpu core to register SPI ISR.

int **intr_flags**

Interrupt flag for the bus to set the priority, and IRAM attribute, see `esp_intr_alloc.h`. Note that the EDGE, INTRDISABLED attribute are ignored by the driver. Note that if ESP_INTR_FLAG_IRAM is set, ALL the callbacks of the driver, and their callee functions, should be put in the IRAM.

Macros

SPI_MAX_DMA_LEN

SPI_SWAP_DATA_TX(DATA, LEN)

Transform unsigned integer of length <= 32 bits to the format which can be sent by the SPI driver directly.

E.g. to send 9 bits of data, you can:

```
uint16_t data = SPI_SWAP_DATA_TX(0x145, 9);
```

Then points tx_buffer to `&data`.

参数

- **DATA** -- Data to be sent, can be uint8_t, uint16_t or uint32_t.
- **LEN** -- Length of data to be sent, since the SPI peripheral sends from the MSB, this helps to shift the data to the MSB.

SPI_SWAP_DATA_RX(DATA, LEN)

Transform received data of length <= 32 bits to the format of an unsigned integer.

E.g. to transform the data of 15 bits placed in a 4-byte array to integer:

```
uint16_t data = SPI_SWAP_DATA_RX(*(uint32_t*)t->rx_data, 15);
```

参数

- **DATA** -- Data to be rearranged, can be uint8_t, uint16_t or uint32_t.
- **LEN** -- Length of data received, since the SPI peripheral writes from the MSB, this helps to shift the data to the LSB.

SPICOMMON_BUSFLAG_SLAVE

Initialize I/O in slave mode.

SPICOMMON_BUSFLAG_MASTER

Initialize I/O in master mode.

SPICOMMON_BUSFLAG_IOMUX_PINS

Check using iomux pins. Or indicates the pins are configured through the IO mux rather than GPIO matrix.

SPICOMMON_BUSFLAG_GPIO_PINS

Force the signals to be routed through GPIO matrix. Or indicates the pins are routed through the GPIO matrix.

SPICOMMON_BUSFLAG_SCLK

Check existing of SCLK pin. Or indicates CLK line initialized.

SPICOMMON_BUSFLAG_MISO

Check existing of MISO pin. Or indicates MISO line initialized.

SPICOMMON_BUSFLAG_MOSI

Check existing of MOSI pin. Or indicates MOSI line initialized.

SPICOMMON_BUSFLAG_DUAL

Check MOSI and MISO pins can output. Or indicates bus able to work under DIO mode.

SPICOMMON_BUSFLAG_WPHD

Check existing of WP and HD pins. Or indicates WP & HD pins initialized.

SPICOMMON_BUSFLAG_QUAD

Check existing of MOSI/MISO/WP/HD pins as output. Or indicates bus able to work under QIO mode.

SPICOMMON_BUSFLAG_IO4_IO7

Check existing of IO4~IO7 pins. Or indicates IO4~IO7 pins initialized.

SPICOMMON_BUSFLAG_OCTAL

Check existing of MOSI/MISO/WP/HD/SPIIO4/SPIIO5/SPIIO6/SPIIO7 pins as output. Or indicates bus able to work under octal mode.

SPICOMMON_BUSFLAG_NATIVE_PINS

Type Definitions

typedef *spi_common_dma_t* **spi_dma_chan_t**

Enumerations

enum **spi_common_dma_t**

SPI DMA channels.

Values:

enumerator **SPI_DMA_DISABLED**

Do not enable DMA for SPI.

enumerator **SPI_DMA_CH_AUTO**

Enable DMA, channel is automatically selected by driver.

API 参考 - SPI Master

Header File

- components/esp_driver_spi/include/driver/spi_master.h
- This header file can be included with:

```
#include "driver/spi_master.h"
```

- This header file is a part of the API provided by the `esp_driver_spi` component. To declare that your component depends on `esp_driver_spi`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_spi
```

or

```
PRIV_REQUIRES esp_driver_spi
```

Functions

esp_err_t **spi_bus_add_device**(*spi_host_device_t* host_id, const *spi_device_interface_config_t* *dev_config, *spi_device_handle_t* *handle)

Allocate a device on a SPI bus.

This initializes the internal structures for a device, plus allocates a CS pin on the indicated SPI master peripheral and routes it to the indicated GPIO. All SPI master devices have three CS pins and can thus control up to three devices.

There's no notable delay on chips other than ESP32.

备注： On ESP32, due to the delay of GPIO matrix, the maximum frequency SPI Master can correctly samples the slave's output is lower than the case using IOMUX. Typical maximum frequency communicating with an ideal slave without data output delay: 80MHz (IOMUX pins) and 26MHz (GPIO matrix pins). With the help of extra dummy cycles in half-duplex mode, the delay can be compensated by setting `input_delay_ns` in `dev_config` structure correctly.

参数

- **host_id** -- SPI peripheral to allocate device on
- **dev_config** -- SPI interface protocol config for the device
- **handle** -- Pointer to variable to hold the device handle

返回

- ESP_ERR_INVALID_ARG if parameter is invalid or configuration combination is not supported (e.g. `dev_config->post_cb` isn't set while flag `SPI_DEVICE_NO_RETURN_RESULT` is enabled)
- ESP_ERR_INVALID_STATE if selected clock source is unavailable or spi bus not initialized
- ESP_ERR_NOT_FOUND if host doesn't have any free CS slots
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

esp_err_t **spi_bus_remove_device**(*spi_device_handle_t* handle)

Remove a device from the SPI bus.

参数 **handle** -- Device handle to free

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_INVALID_STATE if device already is freed
- ESP_OK on success

esp_err_t **spi_device_queue_trans**(*spi_device_handle_t* handle, *spi_transaction_t* *trans_desc, TickType_t ticks_to_wait)

Queue a SPI transaction for interrupt transaction execution. Get the result by `spi_device_get_trans_result`.

备注: Normally a device cannot start (queue) polling and interrupt transactions simultaneously.

参数

- **handle** -- Device handle obtained using spi_host_add_dev
- **trans_desc** -- Description of transaction to execute
- **ticks_to_wait** -- Ticks to wait until there's room in the queue; use portMAX_DELAY to never time out.

返回

- ESP_ERR_INVALID_ARG if parameter is invalid. This can happen if SPI_TRANS_CS_KEEP_ACTIVE flag is specified while the bus was not acquired (`spi_device_acquire_bus()` should be called first) or set flag SPI_TRANS_DMA_BUFFER_ALIGN_MANUAL but tx or rx buffer not DMA-capable, or addr&len not align to cache line size
- ESP_ERR_TIMEOUT if there was no room in the queue before ticks_to_wait expired
- ESP_ERR_NO_MEM if allocating DMA-capable temporary buffer failed
- ESP_ERR_INVALID_STATE if previous transactions are not finished
- ESP_OK on success

esp_err_t **spi_device_get_trans_result**(*spi_device_handle_t* handle, *spi_transaction_t* **trans_desc, TickType_t ticks_to_wait)

Get the result of a SPI transaction queued earlier by `spi_device_queue_trans`.

This routine will wait until a transaction to the given device successfully completed. It will then return the description of the completed transaction so software can inspect the result and e.g. free the memory or reuse the buffers.

参数

- **handle** -- Device handle obtained using spi_host_add_dev
- **trans_desc** -- Pointer to variable able to contain a pointer to the description of the transaction that is executed. The descriptor should not be modified until the descriptor is returned by spi_device_get_trans_result.
- **ticks_to_wait** -- Ticks to wait until there's a returned item; use portMAX_DELAY to never time out.

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_SUPPORTED if flag `SPI_DEVICE_NO_RETURN_RESULT` is set
- ESP_ERR_TIMEOUT if there was no completed transaction before ticks_to_wait expired
- ESP_OK on success

esp_err_t **spi_device_transmit**(*spi_device_handle_t* handle, *spi_transaction_t* *trans_desc)

Send a SPI transaction, wait for it to complete, and return the result.

This function is the equivalent of calling spi_device_queue_trans() followed by spi_device_get_trans_result(). Do not use this when there is still a transaction separately queued (started) from spi_device_queue_trans() or polling_start/transmit that hasn't been finalized.

备注： This function is not thread safe when multiple tasks access the same SPI device. Normally a device cannot start (queue) polling and interrupt transactions simutanuously.

参数

- **handle** -- Device handle obtained using spi_host_add_dev
- **trans_desc** -- Description of transaction to execute

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

esp_err_t **spi_device_polling_start**(*spi_device_handle_t* handle, *spi_transaction_t* *trans_desc, TickType_t ticks_to_wait)

Immediately start a polling transaction.

备注： Normally a device cannot start (queue) polling and interrupt transactions simutanuously. Moreover, a device cannot start a new polling transaction if another polling transaction is not finished.

参数

- **handle** -- Device handle obtained using spi_host_add_dev
- **trans_desc** -- Description of transaction to execute
- **ticks_to_wait** -- Ticks to wait until there's room in the queue; currently only portMAX_DELAY is supported.

返回

- ESP_ERR_INVALID_ARG if parameter is invalid. This can happen if SPI_TRANS_CS_KEEP_ACTIVE flag is specified while the bus was not acquired (`spi_device_acquire_bus()` should be called first) or set flag SPI_TRANS_DMA_BUFFER_ALIGN_MANUAL but tx or rx buffer not DMA-capable, or addr&len not align to cache line size
- ESP_ERR_TIMEOUT if the device cannot get control of the bus before `ticks_to_wait` expired
- ESP_ERR_NO_MEM if allocating DMA-capable temporary buffer failed
- ESP_ERR_INVALID_STATE if previous transactions are not finished
- ESP_OK on success

esp_err_t **spi_device_polling_end**(*spi_device_handle_t* handle, TickType_t ticks_to_wait)

Poll until the polling transaction ends.

This routine will not return until the transaction to the given device has successfully completed. The task is not blocked, but actively busy-spins for the transaction to be completed.

参数

- **handle** -- Device handle obtained using spi_host_add_dev
- **ticks_to_wait** -- Ticks to wait until there's a returned item; use portMAX_DELAY to never time out.

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if the transaction cannot finish before ticks_to_wait expired
- ESP_OK on success

esp_err_t **spi_device_polling_transmit**(*spi_device_handle_t* handle, *spi_transaction_t* *trans_desc)

Send a polling transaction, wait for it to complete, and return the result.

This function is the equivalent of calling spi_device_polling_start() followed by spi_device_polling_end(). Do not use this when there is still a transaction that hasn't been finalized.

备注： This function is not thread safe when multiple tasks access the same SPI device. Normally a device cannot start (queue) polling and interrupt transactions simutanuously.

参数

- **handle** -- Device handle obtained using spi_host_add_dev
- **trans_desc** -- Description of transaction to execute

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if the device cannot get control of the bus
- ESP_ERR_NO_MEM if allocating DMA-capable temporary buffer failed
- ESP_ERR_INVALID_STATE if previous transactions of same device are not finished
- ESP_OK on success

esp_err_t **spi_device_acquire_bus**(*spi_device_handle_t* device, TickType_t wait)

Occupy the SPI bus for a device to do continuous transactions.

Transactions to all other devices will be put off until `spi_device_release_bus` is called.

备注： The function will wait until all the existing transactions have been sent.

参数

- **device** -- The device to occupy the bus.
- **wait** -- Time to wait before the the bus is occupied by the device. Currently MUST set to portMAX_DELAY.

返回

- ESP_ERR_INVALID_ARG : `wait` is not set to portMAX_DELAY.
- ESP_OK : Success.

void **spi_device_release_bus**(*spi_device_handle_t* dev)

Release the SPI bus occupied by the device. All other devices can start sending transactions.

参数 **dev** -- The device to release the bus.

esp_err_t **spi_device_get_actual_freq**(*spi_device_handle_t* handle, int *freq_khz)

Calculate working frequency for specific device.

参数

- **handle** -- SPI device handle
- **freq_khz** -- **[out]** output parameter to hold calculated frequency in kHz

返回

- ESP_ERR_INVALID_ARG : `handle` or `freq_khz` parameter is NULL
- ESP_OK : Success

int **spi_get_actual_clock**(int fapb, int hz, int duty_cycle)

Calculate the working frequency that is most close to desired frequency.

参数

- **fapb** -- The frequency of apb clock, should be `APB_CLK_FREQ`.
- **hz** -- Desired working frequency
- **duty_cycle** -- Duty cycle of the spi clock

返回 Actual working frequency that most fit.

void **spi_get_timing**(bool gpio_is_used, int input_delay_ns, int eff_clk, int *dummy_o, int *cycles_remain_o)

Calculate the timing settings of specified frequency and settings.

备注： If **dummy_o* is not zero, it means dummy bits should be applied in half duplex mode, and full duplex mode may not work.

参数

- **gpio_is_used** -- True if using GPIO matrix, or False if iomux pins are used.
- **input_delay_ns** -- Input delay from SCLK launch edge to MISO data valid.
- **eff_clk** -- Effective clock frequency (in Hz) from `spi_get_actual_clock()`.
- **dummy_o** -- Address of dummy bits used output. Set to NULL if not needed.
- **cycles_remain_o** -- Address of cycles remaining (after dummy bits are used) output.
  - -1 If too many cycles remaining, suggest to compensate half a clock.
  - 0 If no remaining cycles or dummy bits are not used.
  - positive value: cycles suggest to compensate.

int **spi_get_freq_limit**(bool gpio_is_used, int input_delay_ns)

Get the frequency limit of current configurations. SPI master working at this limit is OK, while above the limit, full duplex mode and DMA will not work, and dummy bits will be applied in the half duplex mode.

参数

- **gpio_is_used** -- True if using GPIO matrix, or False if native pins are used.
- **input_delay_ns** -- Input delay from SCLK launch edge to MISO data valid.

返回 Frequency limit of current configurations.

esp_err_t **spi_bus_get_max_transaction_len**(*spi_host_device_t* host_id, size_t *max_bytes)

Get max length (in bytes) of one transaction.

参数

- **host_id** -- SPI peripheral
- **max_bytes** -- **[out]** Max length of one transaction, in bytes

返回

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid argument

Structures

struct **spi_device_interface_config_t**

This is a configuration for a SPI slave device that is connected to one of the SPI buses.

Public Members

uint8_t **command_bits**

Default amount of bits in command phase (0-16), used when `SPI_TRANS_VARIABLE_CMD` is not used, otherwise ignored.

uint8_t **address_bits**

Default amount of bits in address phase (0-64), used when `SPI_TRANS_VARIABLE_ADDR` is not used, otherwise ignored.

uint8_t **dummy_bits**

Amount of dummy bits to insert between address and data phase.

uint8_t **mode**

SPI mode, representing a pair of (CPOL, CPHA) configuration:

- 0: (0, 0)

- 1: (0, 1)
- 2: (1, 0)
- 3: (1, 1)

spi_clock_source_t **clock_source**

Select SPI clock source, `SPI_CLK_SRC_DEFAULT` by default.

uint16_t **duty_cycle_pos**

Duty cycle of positive clock, in 1/256th increments (128 = 50%/50% duty). Setting this to 0 (=not setting it) is equivalent to setting this to 128.

uint16_t **cs_ena_pretrans**

Amount of SPI bit-cycles the cs should be activated before the transmission (0-16). This only works on half-duplex transactions.

uint8_t **cs_ena_posttrans**

Amount of SPI bit-cycles the cs should stay active after the transmission (0-16)

int **clock_speed_hz**

SPI clock speed in Hz. Derived from `clock_source`.

int **input_delay_ns**

Maximum data valid time of slave. The time required between SCLK and MISO valid, including the possible clock delay from slave to master. The driver uses this value to give an extra delay before the MISO is ready on the line. Leave at 0 unless you know you need a delay. For better timing performance at high frequency (over 8MHz), it's suggest to have the right value.

int **spics_io_num**

CS GPIO pin for this device, or -1 if not used.

uint32_t **flags**

Bitwise OR of SPI_DEVICE_* flags.

int **queue_size**

Transaction queue size. This sets how many transactions can be 'in the air' (queued using spi_device_queue_trans but not yet finished using spi_device_get_trans_result) at the same time.

transaction_cb_t **pre_cb**

Callback to be called before a transmission is started.

This callback is called within interrupt context should be in IRAM for best performance, see "Transferring Speed" section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

transaction_cb_t **post_cb**

Callback to be called after a transmission has completed.

This callback is called within interrupt context should be in IRAM for best performance, see "Transferring Speed" section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

struct **spi_transaction_t**

This structure describes one SPI transaction. The descriptor should not be modified until the transaction finishes.

Public Members

uint32_t **flags**

Bitwise OR of SPI_TRANS_* flags.

uint16_t **cmd**

Command data, of which the length is set in the `command_bits` of *spi_device_interface_config_t*.

NOTE: this field, used to be "command" in ESP-IDF 2.1 and before, is re-written to be used in a new way in ESP-IDF 3.0.

Example: write 0x0123 and command_bits=12 to send command 0x12, 0x3_ (in previous version, you may have to write 0x3_12).

uint64_t **addr**

Address data, of which the length is set in the `address_bits` of *spi_device_interface_config_t*.

NOTE: this field, used to be "address" in ESP-IDF 2.1 and before, is re-written to be used in a new way in ESP-IDF3.0.

Example: write 0x123400 and address_bits=24 to send address of 0x12, 0x34, 0x00 (in previous version, you may have to write 0x12340000).

size_t **length**

Total data length, in bits.

size_t **rxlength**

Total data length received, should be not greater than `length` in full-duplex mode (0 defaults this to the value of `length`).

void ***user**

User-defined variable. Can be used to store eg transaction ID.

const void ***tx_buffer**

Pointer to transmit buffer, or NULL for no MOSI phase.

uint8_t **tx_data**[4]

If SPI_TRANS_USE_TXDATA is set, data set here is sent directly from this variable.

void ***rx_buffer**

Pointer to receive buffer, or NULL for no MISO phase. Written by 4 bytes-unit if DMA is used.

uint8_t **rx_data**[4]

If SPI_TRANS_USE_RXDATA is set, data is received directly to this variable.

struct **spi_transaction_ext_t**

This struct is for SPI transactions which may change their address and command length. Please do set the flags in base to `SPI_TRANS_VARIABLE_CMD_ADR` to use the bit length here.

Public Members

struct *spi_transaction_t* **base**

Transaction data, so that pointer to *spi_transaction_t* can be converted into *spi_transaction_ext_t*.

uint8_t **command_bits**

The command length in this transaction, in bits.

uint8_t **address_bits**

The address length in this transaction, in bits.

uint8_t **dummy_bits**

The dummy length in this transaction, in bits.

Macros

SPI_MASTER_FREQ_8M

SPI common used frequency (in Hz)

> **备注：** SPI peripheral only has an integer divider, and the default clock source can be different on other targets, so the actual frequency may be slightly different from the desired frequency. 8MHz

SPI_MASTER_FREQ_9M

8.89MHz

SPI_MASTER_FREQ_10M

10MHz

SPI_MASTER_FREQ_11M

11.43MHz

SPI_MASTER_FREQ_13M

13.33MHz

SPI_MASTER_FREQ_16M

16MHz

SPI_MASTER_FREQ_20M

20MHz

SPI_MASTER_FREQ_26M

26.67MHz

SPI_MASTER_FREQ_40M

40MHz

SPI_MASTER_FREQ_80M

80MHz

SPI_DEVICE_TXBIT_LSBFIRST

Transmit command/address/data LSB first instead of the default MSB first.

SPI_DEVICE_RXBIT_LSBFIRST

Receive data LSB first instead of the default MSB first.

SPI_DEVICE_BIT_LSBFIRST

Transmit and receive LSB first.

SPI_DEVICE_3WIRE

Use MOSI (=spid) for both sending and receiving data.

SPI_DEVICE_POSITIVE_CS

Make CS positive during a transaction instead of negative.

SPI_DEVICE_HALFDUPLEX

Transmit data before receiving it, instead of simultaneously.

SPI_DEVICE_CLK_AS_CS

Output clock on CS line if CS is active.

SPI_DEVICE_NO_DUMMY

There are timing issue when reading at high frequency (the frequency is related to whether iomux pins are used, valid time after slave sees the clock).

- In half-duplex mode, the driver automatically inserts dummy bits before reading phase to fix the timing issue. Set this flag to disable this feature.
- In full-duplex mode, however, the hardware cannot use dummy bits, so there is no way to prevent data being read from getting corrupted. Set this flag to confirm that you're going to work with output only, or read without dummy bits at your own risk.

SPI_DEVICE_DDRCLK

SPI_DEVICE_NO_RETURN_RESULT

Don't return the descriptor to the host on completion (use post_cb to notify instead)

SPI_TRANS_MODE_DIO

Transmit/receive data in 2-bit mode.

SPI_TRANS_MODE_QIO

Transmit/receive data in 4-bit mode.

SPI_TRANS_USE_RXDATA

Receive into rx_data member of *spi_transaction_t* instead into memory at rx_buffer.

SPI_TRANS_USE_TXDATA

Transmit tx_data member of *spi_transaction_t* instead of data at tx_buffer. Do not set tx_buffer when using this.

SPI_TRANS_MODE_DIOQIO_ADDR

Also transmit address in mode selected by SPI_MODE_DIO/SPI_MODE_QIO.

SPI_TRANS_VARIABLE_CMD

Use the `command_bits` in *spi_transaction_ext_t* rather than default value in *spi_device_interface_config_t*.

SPI_TRANS_VARIABLE_ADDR

Use the `address_bits` in *spi_transaction_ext_t* rather than default value in *spi_device_interface_config_t*.

SPI_TRANS_VARIABLE_DUMMY

Use the `dummy_bits` in *spi_transaction_ext_t* rather than default value in *spi_device_interface_config_t*.

SPI_TRANS_CS_KEEP_ACTIVE

Keep CS active after data transfer.

SPI_TRANS_MULTILINE_CMD

The data lines used at command phase is the same as data phase (otherwise, only one data line is used at command phase)

SPI_TRANS_MODE_OCT

Transmit/receive data in 8-bit mode.

SPI_TRANS_MULTILINE_ADDR

The data lines used at address phase is the same as data phase (otherwise, only one data line is used at address phase)

SPI_TRANS_DMA_BUFFER_ALIGN_MANUAL

By default driver will automatically re-alloc dma buffer if it doesn't meet hardware alignment or dma_capable requirements, this flag is for you to disable this feature, you will need to take care of the alignment otherwise driver will return you error ESP_ERR_INVALID_ARG.

Type Definitions

typedef void (***transaction_cb_t**)(*spi_transaction_t* *trans)

typedef struct spi_device_t ***spi_device_handle_t**

Handle for a device on a SPI bus.

2.5.30 SPI 从机驱动程序

SPI 从机驱动程序控制在 ESP32-P4 中作为从机的 GP-SPI 外设。

有关 GP-SPI 硬件相关信息，请参考 **ESP32-P4 技术参考手册 > SPI 控制器** [PDF]。

术语

下表为 SPI 从机驱动的相关术语。

术语	定义
主机 (Host)	ESP32-P4 外部的 SPI 控制器外设。用作 SPI 主机，在总线上发起 SPI 传输。
从机设备 (Device)	SPI 从机设备（通用 SPI 控制器）。每个从机设备共享 MOSI、MISO 和 SCLK 信号，但只有当主机向从机设备的专属 CS 线发出信号时，从机设备才会在总线上处于激活状态。
总线 (Bus)	信号总线，由连接到同一主机的所有从机设备共用。一般来说，一条总线包括以下线路：MISO、MOSI、SCLK、一条或多条 CS 线，以及可选的 QUADWP 和 QUADHD。每个从机设备都有单独的 CS 线，除此之外，所有从机设备都连接在相同的线路下。如果以菊花链的方式连接，几个从机设备也可以共享一条 CS 线。
MISO	主机输入，从机输出，也写作 Q。数据从从机设备发送至主机。
MOSI	主机输出，从机输入，也写作 D。数据从主机发送至从机设备。
SCLK	串行时钟。由主机产生的振荡信号，使数据位的传输保持同步。
CS	片选。允许主机选择连接到总线上的单个从机设备，以便发送或接收数据。
QUADWP	写保护信号。只用于 4 位 (qio/qout) 传输。
QUADHD	保持信号。只用于 4 位 (qio/qout) 传输。
断言 (Assertion)	指激活一条线的操作。反之，将线路恢复到非活动状态（回到空闲状态）的操作则称为 **去断言**。
传输事务 (Transaction)	即主机断言从机设备的 CS 线，向从机设备传输数据，接着去断言 CS 线的过程。传输事务为原子操作，不可打断。
发射沿 (Launch Edge)	源寄存器将信号 **发射**到线路上的时钟边沿。
锁存沿 (Latch Edge)	目的寄存器 **锁存**信号的时钟边沿。

驱动程序的功能

SPI 从机驱动程序允许将 SPI 外设作为全双工设备使用。驱动程序可以发送/接收长度不超过 64 字节的传输事务，或者利用 DMA 来发送/接收更长的传输事务。然而，存在一些与 DMA 有关的已知问题。

SPI 从机驱动程序支持将 SPI ISR 注册至指定 CPU 内核。如果多个任务同时尝试访问一个 SPI 设备，建议重构应用程序，以使每个 SPI 外设一次只由一个任务访问。此外，请使用 *spi_bus_config_t::isr_cpu_id* 将 SPI ISR 注册至与 SPI 外设相关任务相同的内核，确保线程安全。

SPI 传输事务

主机断言 CS 线并在 SCLK 线上发出时钟脉冲时，一次全双工 SPI 传输事务就此开始。每个时钟脉冲都意味着通过 MOSI 线从主机转移一个数据位到从机设备上，并同时通过 MISO 线返回一个数据位。传输事务结束后，主机去断言 CS 线。

传输事务的属性由作为从机设备的 SPI 外设的配置结构体 *spi_slave_interface_config_t* 和传输事务配置结构体 *spi_slave_transaction_t* 决定。

由于并非每次传输事务都需要写入和读取数据，可以选择配置 *spi_transaction_t* 为仅 TX、仅 RX 或同时 TX 和 RX 传输事务。如果将 *spi_slave_transaction_t::rx_buffer* 设置为 NULL，读取阶段将被跳过。与之类似，如果将 *spi_slave_transaction_t::tx_buffer* 设置为 NULL，则写入阶段将被跳过。

备注： 主机应在从机设备准备好接收数据之后再进行传输事务。建议使用另外一个 GPIO 管脚作为握手信号来同步设备。更多细节，请参阅传输事务间隔。

使用驱动程序

- 调用函数*spi_slave_initialize()*，将 SPI 外设初始化为从机设备。请确保在 *bus_config* 中设置正确的 I/O 管脚，并将未使用的信号设置为 -1。
- 传输事务开始前，需用要求的事务参数填充一个或多个*spi_slave_transaction_t*结构体。可以通过调用函数*spi_slave_queue_trans()* 来将所有传输事务排进队列，并在稍后使用函数*spi_slave_get_trans_result()* 查询结果；也可以将所有请求输入*spi_slave_transmit()* 中单独处理。主机上的传输事务完成前，后两个函数将被阻塞，以便发送并接收队列中的数据。

-（可选）如需卸载 SPI 从机驱动程序，请调用*spi_slave_free()*。

传输事务数据和主/从机长度不匹配

通常，通过从机设备进行传输的数据会被读取或写入到由*spi_slave_transaction_t::rx_buffer*和*spi_slave_transaction_t::tx_buffer*指示的大块内存中。可以配置 SPI 驱动程序，使用 DMA 进行传输。在这种情况下，则必须使用 `pvPortMallocCaps(size, MALLOC_CAP_DMA)` 将缓存区分配到具备 DMA 功能的内存中。

驱动程序可以读取或写入缓存区的数据量取决于*spi_slave_transaction_t::length*，但其并不会定义一次 SPI 传输的实际长度。传输事务的长度由主机的时钟线和 CS 线决定，且只有在传输事务完成后，才能从*spi_slave_transaction_t::trans_len* 中读取实际长度。

如果传输长度超过缓存区长度，则只有在*spi_slave_transaction_t::length* 中指定的初始比特数会被发送和接收。此时，*spi_slave_transaction_t::trans_len* 被设置为*spi_slave_transaction_t::length* 而非实际传输事务长度。若需满足实际传输事务长度的要求，请将*spi_slave_transaction_t::length* 设置为大于*spi_slave_transaction_t::trans_len* 预期最大值的值。如果传输长度短于缓存区长度，则只传输与缓存区长度相等的数据。

GPIO 交换矩阵和 IO_MUX

ESP32-P4 的大多数外设信号都直接连接到其专用的 IO_MUX 管脚。不过，也可以使用 GPIO 交换矩阵，将信号路由到任何可用的其他管脚。如果通过 GPIO 交换矩阵路由了至少一个信号，则所有信号都将通过 GPIO 交换矩阵路由。

当 SPI 主机频率配置为 80 MHz 或更低时，则通过 GPIO 交换矩阵或 IO_MUX 路由 SPI 管脚效果相同。

下表列出了 SPI 总线的 IO_MUX 管脚。

管脚名称	GPIO 编号 (SPI2)
CS0	7
SCLK	9
MISO	10
MOSI	8
QUADWP	11
QUADHD	6

速度与时钟

传输事务间隔 ESP32-P4 的 SPI 从机外设是由 CPU 控制的通用从机设备。与专用的从机相比，在内嵌 CPU 的 SPI 从机设备中，预定义寄存器的数量有限，所有的传输事务都必须由 CPU 处理。也就是说，传输和响应并不是实时的，且可能存在明显的延迟。

解决方案为，首先使用函数*spi_slave_queue_trans()*，然后使用*spi_slave_get_trans_result()*，来代替*spi_slave_transmit()*。由此一来，可使从机设备的响应速度提高一倍。

也可以配置一个 GPIO 管脚，当从机设备开始新一次传输事务前，它将通过该管脚向主机发出信号。示例代码存放在 peripherals/spi_slave 目录下。

时钟频率要求 SPI 从机的工作频率最高可达 60 MHz。如果时钟频率过快或占空比不足 50%，数据就无法被正确识别或接收。

限制条件和已知问题

1. 若启用了 DMA，则 RX 缓冲区应该以字对齐（从 32 位边界开始，字节长度为 4 的倍数）。否则，DMA 可能无法正确写入或无法实现边界对齐。若此项条件不满足，驱动程序将会报错。
   此外，主机写入字节长度应为 4 的倍数。长度不符合的数据将被丢弃。

应用示例

从机设备/主机通信的示例代码存放在 ESP-IDF 示例项目的 peripherals/spi_slave 目录下。

API 参考

Header File

- components/esp_driver_spi/include/driver/spi_slave.h
- This header file can be included with:

```
#include "driver/spi_slave.h"
```

- This header file is a part of the API provided by the `esp_driver_spi` component. To declare that your component depends on `esp_driver_spi`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_spi
```

or

```
PRIV_REQUIRES esp_driver_spi
```

Functions

esp_err_t **spi_slave_initialize**(*spi_host_device_t* host, const *spi_bus_config_t* *bus_config, const *spi_slave_interface_config_t* *slave_config, *spi_dma_chan_t* dma_chan)

Initialize a SPI bus as a slave interface.

警告： SPI0/1 is not supported

警告： If a DMA channel is selected, any transmit and receive buffer used should be allocated in DMA-capable memory.

警告： The ISR of SPI is always executed on the core which calls this function. Never starve the ISR on this core or the SPI transactions will not be handled.

参数

- **host** -- SPI peripheral to use as a SPI slave interface
- **bus_config** -- Pointer to a *spi_bus_config_t* struct specifying how the host should be initialized

- **slave_config** -- Pointer to a *spi_slave_interface_config_t* struct specifying the details for the slave interface
- **dma_chan** -- - Selecting a DMA channel for an SPI bus allows transactions on the bus with size only limited by the amount of internal memory.
  - Selecting SPI_DMA_DISABLED limits the size of transactions.
  - Set to SPI_DMA_DISABLED if only the SPI flash uses this bus.
  - Set to SPI_DMA_CH_AUTO to let the driver to allocate the DMA channel.

返回

- ESP_ERR_INVALID_ARG if configuration is invalid
- ESP_ERR_INVALID_STATE if host already is in use
- ESP_ERR_NOT_FOUND if there is no available DMA channel
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

esp_err_t **spi_slave_free**(*spi_host_device_t* host)

Free a SPI bus claimed as a SPI slave interface.

参数 **host** -- SPI peripheral to free

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_INVALID_STATE if not all devices on the bus are freed
- ESP_OK on success

esp_err_t **spi_slave_queue_trans**(*spi_host_device_t* host, const *spi_slave_transaction_t* *trans_desc, TickType_t ticks_to_wait)

Queue a SPI transaction for execution.

Queues a SPI transaction to be executed by this slave device. (The transaction queue size was specified when the slave device was initialised via spi_slave_initialize.) This function may block if the queue is full (depending on the ticks_to_wait parameter). No SPI operation is directly initiated by this function, the next queued transaction will happen when the master initiates a SPI transaction by pulling down CS and sending out clock signals.

This function hands over ownership of the buffers in `trans_desc` to the SPI slave driver; the application is not to access this memory until `spi_slave_queue_trans` is called to hand ownership back to the application.

备注: On esp32, if trans length not WORD aligned, the rx buffer last word memory will still overwritten by DMA HW

参数

- **host** -- SPI peripheral that is acting as a slave
- **trans_desc** -- Description of transaction to execute. Not const because we may want to write status back into the transaction description.
- **ticks_to_wait** -- Ticks to wait until there's room in the queue; use portMAX_DELAY to never time out.

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if set flag `SPI_SLAVE_TRANS_DMA_BUFFER_ALIGN_AUTO` but there is no free memory
- ESP_ERR_INVALID_STATE if sync data between Cache and memory failed
- ESP_OK on success

esp_err_t **spi_slave_get_trans_result**(*spi_host_device_t* host, *spi_slave_transaction_t* **trans_desc, TickType_t ticks_to_wait)

Get the result of a SPI transaction queued earlier.

This routine will wait until a transaction to the given device (queued earlier with spi_slave_queue_trans) has successfully completed. It will then return the description of the completed transaction so software can inspect the result and e.g. free the memory or reuse the buffers.

It is mandatory to eventually use this function for any transaction queued by `spi_slave_queue_trans`.

参数

- **`host`** -- SPI peripheral to that is acting as a slave
- **`trans_desc`** -- **[out]** Pointer to variable able to contain a pointer to the description of the transaction that is executed
- **`ticks_to_wait`** -- Ticks to wait until there's a returned item; use portMAX_DELAY to never time out.

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_SUPPORTED if flag `SPI_SLAVE_NO_RETURN_RESULT` is set
- ESP_OK on success

esp_err_t **`spi_slave_transmit`** (*spi_host_device_t* host, *spi_slave_transaction_t* *trans_desc, TickType_t ticks_to_wait)

Do a SPI transaction.

Essentially does the same as spi_slave_queue_trans followed by spi_slave_get_trans_result. Do not use this when there is still a transaction queued that hasn't been finalized using spi_slave_get_trans_result.

参数

- **`host`** -- SPI peripheral to that is acting as a slave
- **`trans_desc`** -- Pointer to variable able to contain a pointer to the description of the transaction that is executed. Not const because we may want to write status back into the transaction description.
- **`ticks_to_wait`** -- Ticks to wait until there's a returned item; use portMAX_DELAY to never time out.

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

Structures

struct **`spi_slave_interface_config_t`**

This is a configuration for a SPI host acting as a slave device.

Public Members

int **`spics_io_num`**

CS GPIO pin for this device.

uint32_t **`flags`**

Bitwise OR of SPI_SLAVE_* flags.

int **`queue_size`**

Transaction queue size. This sets how many transactions can be 'in the air' (queued using spi_slave_queue_trans but not yet finished using spi_slave_get_trans_result) at the same time.

uint8_t **`mode`**

SPI mode, representing a pair of (CPOL, CPHA) configuration:

- 0: (0, 0)
- 1: (0, 1)

- 2: (1, 0)
- 3: (1, 1)

slave_transaction_cb_t **post_setup_cb**

Callback called after the SPI registers are loaded with new data.

This callback is called within interrupt context should be in IRAM for best performance, see "Transferring Speed" section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

slave_transaction_cb_t **post_trans_cb**

Callback called after a transaction is done.

This callback is called within interrupt context should be in IRAM for best performance, see "Transferring Speed" section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

struct **spi_slave_transaction_t**

This structure describes one SPI transaction

Public Members

uint32_t **flags**

Bitwise OR of SPI_SLAVE_TRANS_* flags.

size_t **length**

Total data length, in bits.

size_t **trans_len**

Transaction data length, in bits.

const void ***tx_buffer**

Pointer to transmit buffer, or NULL for no MOSI phase.

void ***rx_buffer**

Pointer to receive buffer, or NULL for no MISO phase. When the DMA is enabled, must start at WORD boundary (`rx_buffer%4==0`), and has length of a multiple of 4 bytes.

void ***user**

User-defined variable. Can be used to store eg transaction ID.

Macros

SPI_SLAVE_TXBIT_LSBFIRST

Transmit command/address/data LSB first instead of the default MSB first.

SPI_SLAVE_RXBIT_LSBFIRST

Receive data LSB first instead of the default MSB first.

SPI_SLAVE_BIT_LSBFIRST

Transmit and receive LSB first.

SPI_SLAVE_NO_RETURN_RESULT

Don't return the descriptor to the host on completion (use `post_trans_cb` to notify instead)

SPI_SLAVE_TRANS_DMA_BUFFER_ALIGN_AUTO

Automatically re-malloc dma buffer if user buffer doesn't meet hardware alignment or dma_capable, this process may loss some memory and performance.

Type Definitions

typedef void (***slave_transaction_cb_t**)(*spi_slave_transaction_t* *trans)

2.5.31 SPI 从机半双工模式

简介

半双工 (HD) 模式是 ESP SPI 从机外设提供的一种特殊模式。比起全双工 (FD) 模式（用于 GPSPI 事务，详情请参阅*SPI 从机驱动程序*），在半双工模式下，硬件会提供更多服务。这些服务减轻了 CPU 负载，提高了 SPI 从机的响应速度。然而，通信格式由硬件确定，始终为半双工模式，即在同一时刻只能进行单向数据传输，半双工模式因此得名。

当进行 SPI 事务时，根据事务的 **命令**阶段，可以将事务分为不同的类型。每个事务可能包含以下阶段：命令、地址、dummy、数据。命令阶段是必需的，其他阶段存在与否则取决于命令的需求。在命令、地址、dummy 阶段，总线的控制权始终由主设备（通常是主机）控制，数据阶段的方向则取决于命令。数据阶段可以是输入阶段，即主设备将数据写入从设备（例如，主机向从机发送数据）；也可以是输出阶段，即主设备从从设备读取数据（例如，主机从从机接收数据）。

协议 有关主设备与 SPI 从机通信的详细信息，请参阅*ESP SPI 从机 HD（半双工）模式协议*。

通过不同类型的事务，从设备为主设备提供以下服务：

- 一个 DMA 通道，支持主设备向从设备写入大量数据。
- 一个 DMA 通道，支持主设备从从设备读取大量数据。
- 一些通用寄存器，由主设备和从设备共享。
- 一些通用中断，用于主设备打断从设备的软件执行。

术语

- 事务 (transaction)
- 通道 (channel)
- 发送 (sending)
- 接收 (receiving)
- 数据描述符 (data descriptor)

驱动程序特性

- 支持主设备以分段方式读写事务
- 支持发送数据队列和接收数据队列

驱动程序使用

从机初始化 调用*spi_slave_hd_init()*初始化 SPI 总线、外设和驱动程序。在初始化之后，SPI 从机将独占使用 SPI 外设和总线上的管脚。这意味着在反初始化前，其他设备无法使用这些资源。因此，为了确保其他设备可以正确利用 SPI 资源并进行正常通信，从机的大部分配置应在其初始化过程中完成。

结构体*spi_bus_config_t* 指定了总线的初始化方式，结构体*spi_slave_hd_slot_config_t* 指定了 SPI 从机驱动程序的运行方式。

从机反初始化（可选） 调用*spi_slave_hd_deinit()*卸载驱动程序。该反初始化函数会释放相关资源，包括管脚、SPI 外设、驱动程序所用内存、中断源等。

通过 DMA 通道发送/接收数据 要通过 DMA 通道向主设备发送数据，应用程序需要先将数据正确地封装在*spi_slave_hd_data_t* 描述符结构体中，然后再将数据描述符和通道参数*SPI_SLAVE_CHAN_TX* 传递给*spi_slave_hd_queue_trans()*。数据描述符的指针存储在队列中，一旦接收到主设备的 Rd_DMA 命令，就会按照调用*spi_slave_hd_queue_trans()* 时数据进入队列的顺序，依次将数据发送给主设备。

应用程序需要检查数据发送的结果。为此，应用程序可以调用*spi_slave_hd_get_trans_res()*，并将通道参数设置为*SPI_SLAVE_CHAN_TX*。该函数将阻塞程序，直到主设备发起的 Rd_DMA 命令事务成功完成或超时。函数中的参数 `out_trans` 将输出刚刚完成的数据描述符的指针，从而提供有关已完成的发送操作的信息。

通过 DMA 通道从主设备接收数据的操作与发送数据类似。应用程序需要使用正确的数据描述符调用*spi_slave_hd_queue_trans()*，并将通道参数设置为*SPI_SLAVE_CHAN_RX*。随后，应用程序调用*spi_slave_hd_get_trans_res()* 获取接收 buffer 的描述符，然后处理接收 buffer 中的数据。

备注： 驱动程序本身并没有用于发送或接收数据的内部 buffer。应用程序需要通过数据描述符为驱动程序提供 buffer，从而向主设备发送数据，或接收来自主设备的数据。

在使用*spi_slave_hd_queue_trans()* 将数据描述符成功发送到驱动程序的内部队列后、并由*spi_slave_hd_get_trans_res()* 返回前，应用程序需要正确地维护数据描述符以及它所指向的 buffer。在此期间，根据需要，硬件和驱动程序可以随时读取或写入 buffer 和描述符。

注意，在使用该驱动程序进行数据传输时，可以根据实际需要提前终止数据传输，而不需要等待整个 buffer 填满或者完全发送完毕。例如，在分段事务模式下，无论发送/接收 buffer 是否已使用完（已满），主设备都需要发送 `CMD7` 终止 `Wr_DMA` 事务，或发送 `CMD8` 以分段方式终止 `Rd_DMA` 事务。

以自定义用户参数使用数据描述符 在某些情况下，发送包函数和回收包函数可能会分散在不同位置。发送包函数用于发送数据描述符，回收包函数用于处理返回的数据描述符。在回收包函数中获取返回的数据描述符时，可能需要一些额外信息，帮助处理数据传输完成后返回给应用程序的描述符。例如，多次发送相同数据时，你可能想知道返回的描述符来自哪一轮发送。

为此，可以通过强制类型转换，将数据描述符中的 `arg` 设置为变量，提供事务信息；或者将其指向一个包含处理发送/接收数据所需的所有信息的结构体。在回收包函数处理返回的描述符时，即可使用这个额外信息。

使用回调函数

备注： 这些回调函数在 ISR 中调用，因此需要迅速处理所需操作，并尽快返回，确保系统正常运行。因此，在编写 ISR 的代码时，需要十分谨慎。

由于中断处理过程是与主程序并发执行的，长时间的延迟或阻塞操作可能会导致系统响应变慢，或导致不可预测的行为。因此，在编写回调函数时，应避免使用可能引起延迟或阻塞的操作，例如等待、睡眠、资源锁等。

在初始化 SPI 从机半双工驱动程序时，会传递结构体*spi_slave_hd_slot_config_t* 中的*spi_slave_hd_callback_config_t*，为任意事件设置回调函数。

每个不为 **NULL** 的回调函数都将使能对应的中断，所以回调函数会在对应的中断事件触发时立即调用。对于不感兴趣的事件，则无需为其提供回调函数。

配置结构体中的 `arg` 可以给回调函数传递部分上下文信息，或在使用相同的回调函数处理多个 SPI 从机外设时，指明特定的 SPI 从机实例。通过强制类型转换，可以将 `arg` 设置为表示 SPI 从机实例的变量，或者将其指向某个上下文结构体变量。所有回调函数都会使用在初始化回调函数时设置的 `arg` 参数。

配置结构体中的 `event` 和 `awoken` 参数也可以给回调函数传递上下文信息。

- 参数 `event` 向回调函数传递当前事件信息。*`spi_slave_hd_event_t`* 包含事件类型和刚刚处理完的数据描述符等信息，此时，通常会使用*data argument*。
- 参数 `awoken` 是一个输出参数，用于告知 ISR，在回调函数后已有其他操作唤醒任务，ISR 应调用 *portYIELD_FROM_ISR()* 调度这些任务。只需将 `awoken` 参数传递给可能解除任务阻塞的 FreeRTOS API，ISR 即可接收 `awoken` 的返回值。

写入/读取共享寄存器 调用*`spi_slave_hd_write_buffer()`* 写入共享 buffer，调用*`spi_slave_hd_read_buffer()`* 读取共享 buffer。

备注： 在 ESP32-P4 上，应用程序以字为单位读取/写入共享寄存器，但主机以字节为单位读取/写入共享寄存器。这样一来，就无法确保从主机读取的四个连续字节是来自从机应用程序写入的同一个字。同时，如果从机在主机写入字节时读取了一个字，可能会得到这样的字：主机刚刚写入它的一半，另一半尚未写入。

通过两次读取同个字，并对两次读取的值做比较，主机可以确保读取的字处于非过渡态。

对从机而言，要确保读取的字处于非过渡态则更为困难，因为主机写入四个字节的过程可能会非常长，达到 32 个 SPI 时钟周期。为此，可以在写入的字的最后一个（最大地址）字节中添加一些冗余校验码 (CRC)，确保在写入含有 CRC 的字节时，即代表整个字完全写入。

从软件读取/写入和从主机读取/写入可能存在冲突，在多核心系统中尤为如此。因此，建议在数据传输过程中，一个字只在一个方向上使用，即要么只由主机写入，要么只由从机写入。

接收来自主机的通用中断 当主机发送 `CMD8`、`CMD9` 或 `CMDA` 时，从机会触发相应的动作。目前，`CMD8` 固定用于指示 `Rd_DMA` 段的终止。要接收通用中断，可以在从机初始化时为 `CMD9` 和 `CMDA` 注册回调函数，详情请参阅使用回调函数。

应用示例

查看从机设备/主机通信的示例代码，请前往 ESP-IDF 示例的 peripherals/spi_slave_hd 目录。

API 参考

Header File

- components/esp_driver_spi/include/driver/spi_slave_hd.h
- This header file can be included with:

```
#include "driver/spi_slave_hd.h"
```

- This header file is a part of the API provided by the `esp_driver_spi` component. To declare that your component depends on `esp_driver_spi`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_spi
```

or

```
PRIV_REQUIRES esp_driver_spi
```

Functions

esp_err_t **spi_slave_hd_init**(*spi_host_device_t* host_id, const *spi_bus_config_t* *bus_config, const *spi_slave_hd_slot_config_t* *config)

Initialize the SPI Slave HD driver.

参数

- **host_id** -- The host to use
- **bus_config** -- Bus configuration for the bus used
- **config** -- Configuration for the SPI Slave HD driver

返回

- ESP_OK: on success
- ESP_ERR_INVALID_ARG: invalid argument given
- ESP_ERR_INVALID_STATE: function called in invalid state, may be some resources are already in use
- ESP_ERR_NOT_FOUND if there is no available DMA channel
- ESP_ERR_NO_MEM: memory allocation failed
- or other return value from `esp_intr_alloc`

esp_err_t **spi_slave_hd_deinit**(*spi_host_device_t* host_id)

Deinitialize the SPI Slave HD driver.

参数 **host_id** -- The host to deinitialize the driver

返回

- ESP_OK: on success
- ESP_ERR_INVALID_ARG: if the host_id is not correct

esp_err_t **spi_slave_hd_queue_trans**(*spi_host_device_t* host_id, *spi_slave_chan_t* chan, *spi_slave_hd_data_t* *trans, TickType_t timeout)

Queue transactions (segment mode)

参数

- **host_id** -- Host to queue the transaction
- **chan** -- SPI_SLAVE_CHAN_TX or SPI_SLAVE_CHAN_RX
- **trans** -- Transaction descriptors
- **timeout** -- Timeout before the data is queued

返回

- ESP_OK: on success
- ESP_ERR_INVALID_ARG: The input argument is invalid. Can be the following reason:
  - The buffer given is not DMA capable
  - The length of data is invalid (not larger than 0, or exceed the max transfer length)
  - The transaction direction is invalid
- ESP_ERR_TIMEOUT: Cannot queue the data before timeout. Master is still processing previous transaction.
- ESP_ERR_INVALID_STATE: Function called in invalid state. This API should be called under segment mode.

esp_err_t **spi_slave_hd_get_trans_res**(*spi_host_device_t* host_id, *spi_slave_chan_t* chan, *spi_slave_hd_data_t* **out_trans, TickType_t timeout)

Get the result of a data transaction (segment mode)

备注: This API should be called successfully the same times as the `spi_slave_hd_queue_trans`.

参数

- **host_id** -- Host to queue the transaction
- **chan** -- Channel to get the result, SPI_SLAVE_CHAN_TX or SPI_SLAVE_CHAN_RX
- **out_trans** -- **[out]** Pointer to the transaction descriptor (*`spi_slave_hd_data_t`*) passed to the driver before. Hardware has finished this transaction. Member `trans_len` indicates the actual number of bytes of received data, it's meaningless for TX.
- **timeout** -- Timeout before the result is got

返回

- ESP_OK: on success
- ESP_ERR_INVALID_ARG: Function is not valid
- ESP_ERR_TIMEOUT: There's no transaction done before timeout
- ESP_ERR_INVALID_STATE: Function called in invalid state. This API should be called under segment mode.

void **spi_slave_hd_read_buffer**(*spi_host_device_t* host_id, int addr, uint8_t *out_data, size_t len)

Read the shared registers.

参数

- **host_id** -- Host to read the shared registers
- **addr** -- Address of register to read, 0 to `SOC_SPI_MAXIMUM_BUFFER_SIZE-1`
- **out_data** -- **[out]** Output buffer to store the read data
- **len** -- Length to read, not larger than `SOC_SPI_MAXIMUM_BUFFER_SIZE-addr`

void **spi_slave_hd_write_buffer**(*spi_host_device_t* host_id, int addr, uint8_t *data, size_t len)

Write the shared registers.

参数

- **host_id** -- Host to write the shared registers
- **addr** -- Address of register to write, 0 to `SOC_SPI_MAXIMUM_BUFFER_SIZE-1`
- **data** -- Buffer holding the data to write
- **len** -- Length to write, `SOC_SPI_MAXIMUM_BUFFER_SIZE-addr`

esp_err_t **spi_slave_hd_append_trans**(*spi_host_device_t* host_id, *spi_slave_chan_t* chan, *spi_slave_hd_data_t* *trans, TickType_t timeout)

Load transactions (append mode)

备注: In this mode, user transaction descriptors will be appended to the DMA and the DMA will keep processing the data without stopping

---

参数

- **host_id** -- Host to load transactions
- **chan** -- SPI_SLAVE_CHAN_TX or SPI_SLAVE_CHAN_RX
- **trans** -- Transaction descriptor
- **timeout** -- Timeout before the transaction is loaded

返回

- ESP_OK: on success
- ESP_ERR_INVALID_ARG: The input argument is invalid. Can be the following reason:
  - The buffer given is not DMA capable
  - The length of data is invalid (not larger than 0, or exceed the max transfer length)
  - The transaction direction is invalid
- ESP_ERR_TIMEOUT: Master is still processing previous transaction. There is no available transaction for slave to load
- ESP_ERR_INVALID_STATE: Function called in invalid state. This API should be called under append mode.

esp_err_t **spi_slave_hd_get_append_trans_res**(*spi_host_device_t* host_id, *spi_slave_chan_t* chan, *spi_slave_hd_data_t* **out_trans, TickType_t timeout)

Get the result of a data transaction (append mode)

备注: This API should be called the same times as the `spi_slave_hd_append_trans`

参数

- **host_id** -- Host to load the transaction
- **chan** -- SPI_SLAVE_CHAN_TX or SPI_SLAVE_CHAN_RX
- **out_trans** -- **[out]** Pointer to the transaction descriptor (*spi_slave_hd_data_t*) passed to the driver before. Hardware has finished this transaction. Member `trans_len` indicates the actual number of bytes of received data, it's meaningless for TX.
- **timeout** -- Timeout before the result is got

返回

- ESP_OK: on success
- ESP_ERR_INVALID_ARG: Function is not valid
- ESP_ERR_TIMEOUT: There's no transaction done before timeout
- ESP_ERR_INVALID_STATE: Function called in invalid state. This API should be called under append mode.

Structures

struct **spi_slave_hd_data_t**

Descriptor of data to send/receive.

Public Members

uint8_t ***data**

Buffer to send, must be DMA capable.

size_t **len**

Len of data to send/receive. For receiving the buffer length should be multiples of 4 bytes, otherwise the extra part will be truncated.

size_t **trans_len**

For RX direction, it indicates the data actually received. For TX direction, it is meaningless.

uint32_t **flags**

Bitwise OR of SPI_SLAVE_HD_TRANS_* flags.

void ***arg**

Extra argument indicating this data.

struct **spi_slave_hd_event_t**

Information of SPI Slave HD event.

Public Members

spi_event_t **event**

Event type.

spi_slave_hd_data_t ***trans**

Corresponding transaction for SPI_EV_SEND and SPI_EV_RECV events.

struct **spi_slave_hd_callback_config_t**

Callback configuration structure for SPI Slave HD.

Public Members

slave_cb_t **cb_buffer_tx**

Callback when master reads from shared buffer.

slave_cb_t **cb_buffer_rx**

Callback when master writes to shared buffer.

slave_cb_t **cb_send_dma_ready**

Callback when TX data buffer is loaded to the hardware (DMA)

slave_cb_t **cb_sent**

Callback when data are sent.

slave_cb_t **cb_recv_dma_ready**

Callback when RX data buffer is loaded to the hardware (DMA)

slave_cb_t **cb_recv**

Callback when data are received.

slave_cb_t **cb_cmd9**

Callback when CMD9 received.

slave_cb_t **cb_cmdA**

Callback when CMDA received.

void ***arg**

Argument indicating this SPI Slave HD peripheral instance.

struct **spi_slave_hd_slot_config_t**

Configuration structure for the SPI Slave HD driver.

Public Members

uint8_t **mode**

SPI mode, representing a pair of (CPOL, CPHA) configuration:

- 0: (0, 0)
- 1: (0, 1)
- 2: (1, 0)
- 3: (1, 1)

uint32_t **spics_io_num**

CS GPIO pin for this device.

uint32_t **flags**

Bitwise OR of SPI_SLAVE_HD_* flags.

uint32_t **command_bits**

command field bits, multiples of 8 and at least 8.

uint32_t **address_bits**

address field bits, multiples of 8 and at least 8.

uint32_t **dummy_bits**

dummy field bits, multiples of 8 and at least 8.

uint32_t **queue_size**

Transaction queue size. This sets how many transactions can be 'in the air' (queued using spi_slave_hd_queue_trans but not yet finished using spi_slave_hd_get_trans_result) at the same time.

spi_dma_chan_t **dma_chan**

DMA channel to used.

spi_slave_hd_callback_config_t **cb_config**

Callback configuration.

Macros

SPI_SLAVE_HD_TRANS_DMA_BUFFER_ALIGN_AUTO

Automatically re-malloc dma buffer if user buffer doesn't meet hardware alignment or dma_capable, this process may lose some memory and performance.

SPI_SLAVE_HD_TXBIT_LSBFIRST

Transmit command/address/data LSB first instead of the default MSB first.

SPI_SLAVE_HD_RXBIT_LSBFIRST

Receive data LSB first instead of the default MSB first.

SPI_SLAVE_HD_BIT_LSBFIRST

Transmit and receive LSB first.

SPI_SLAVE_HD_APPEND_MODE

Adopt DMA append mode for transactions. In this mode, users can load(append) DMA descriptors without stopping the DMA.

Type Definitions

typedef bool (***slave_cb_t**)(void *arg, *spi_slave_hd_event_t* *event, BaseType_t *awoken)

Callback for SPI Slave HD.

Enumerations

enum **spi_slave_chan_t**

Channel of SPI Slave HD to do data transaction.

Values:

enumerator **SPI_SLAVE_CHAN_TX**

The output channel (RDDMA)

enumerator **SPI_SLAVE_CHAN_RX**

The input channel (WRDMA)

2.5.32 JPEG Encoder and Decoder

Introduction

JPEG is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality.

JPEG codec on ESP32-P4 is an image codec, which is based on the JPEG baseline standard, for compressing and decompressing images to reduce the bandwidth required to transmit images or the space required to store images, making it possible to process large-resolution images. But please note, at one time, the codec engine can only work as either encoder or decoder.

Functional Overview

This document covers the following sections:

- *Resource Allocation* - covers how to allocate JPEG resources with properly set of configurations. It also covers how to recycle the resources when they finished working.
- *Finite State Machine* - covers JPEG workflow. Introduce how jpeg driver uses internal resources and its software process.
- *JPEG Decoder Engine* - covers behavior of JPEG decoder engine. Introduce how to use decoder engine functions to decode an image (from jpg format to raw format).
- *JPEG Encoder Engine* - covers behavior of JPEG encoder engine. Introduce how to use encoder engine functions to encode an image (from raw format to jpg format).
- *Performance Overview* - covers encoder and decoder performance.
- *Pixel Storage Layout for Different Color Formats* - covers color space order overview required in this JPEG decoder and encoder.
- *Thread Safety* - lists which APIs are guaranteed to be thread safe by the driver.
- *Power Management* - describes how jpeg driver would be affected by power consumption.
- *Kconfig Options* - lists the supported Kconfig options that can bring different effects to the driver.

Resource Allocation

Install JPEG decoder engine JPEG decoder engine requires the configuration that specified by *jpeg_decode_engine_cfg_t*.

If the configurations in *jpeg_decode_engine_cfg_t* is specified, users can call *jpeg_new_decoder_engine()* to allocate and initialize a JPEG decoder engine. This function will return an JPEG decoder handle if it runs correctly. You can take following code as reference.

```
jpeg_decoder_handle_t decoder_engine;

jpeg_decode_engine_cfg_t decode_eng_cfg = {
    .intr_priority = 0,
    .timeout_ms = 40,
};

ESP_ERROR_CHECK(jpeg_new_decoder_engine(&decode_eng_cfg, &decoder_engine));
```

Uninstall JPEG decoder engine If a previously installed JPEG engine is no longer needed, it's recommended to recycle the resource by calling *jpeg_del_decoder_engine()*, so that the underlying hardware is released.

```
ESP_ERROR_CHECK(jpeg_del_decoder_engine(decoder_engine));
```

Install JPEG encoder engine The JPEG encoder engine requires the configuration specified by *jpeg_encode_engine_cfg_t*.

If the configurations in *jpeg_encode_engine_cfg_t* is specified, users can call *jpeg_new_encoder_engine()* to allocate and initialize a JPEG encoder engine. This function will return an JPEG encoder handle if it runs correctly. You can take following code as reference.

```
jpeg_encoder_handle_t encoder_engine;

jpeg_encode_engine_cfg_t encode_eng_cfg = {
    .intr_priority = 0,
    .timeout_ms = 40,
};

ESP_ERROR_CHECK(jpeg_new_encoder_engine(&encode_eng_cfg, &encoder_engine));
```

Uninstall JPEG encoder engine If a previously installed JPEG engine is no longer needed, it's recommended to recycle the resource by calling *jpeg_del_encoder_engine()*, so that the underlying hardware is released.

```
ESP_ERROR_CHECK(jpeg_del_encoder_engine(encoder_engine));
```

Finite State Machine The JPEG driver usage of hardware resources and its process workflow are shown in the following graph:

JPEG Decoder Engine After installing the JPEG decoder driver by *jpeg_new_decoder_engine()*, ESP32-P4 is ready to decode JPEG pictures by *jpeg_decoder_process()*. *jpeg_decoder_process()* is flexible for decoding different types of pictures by a configurable parameter called *jpeg_decode_cfg_t*.

Moreover, our jpeg decoder api provides a helper function which helps you get the basic information of your given image. Calling *jpeg_decoder_get_info()* would return the picture information structure called *jpeg_decoder_get_info()*. If you already know the picture basic information, this functions is unnecessary to be called.

The format conversions supported by this driver are listed in the table below:

Format of the already compressed image	Format after decompressing
YUV444	RGB565
	RGB888
	YUV444
YUV422	RGB565
	RGB888
	YUV444
	YUV422
YUV420	RGB565
	RGB888
	YUV444
	YUV420
GRAY	GRAY

Overall, You can take following code as reference, the code is going to decode a 1080*1920 picture.

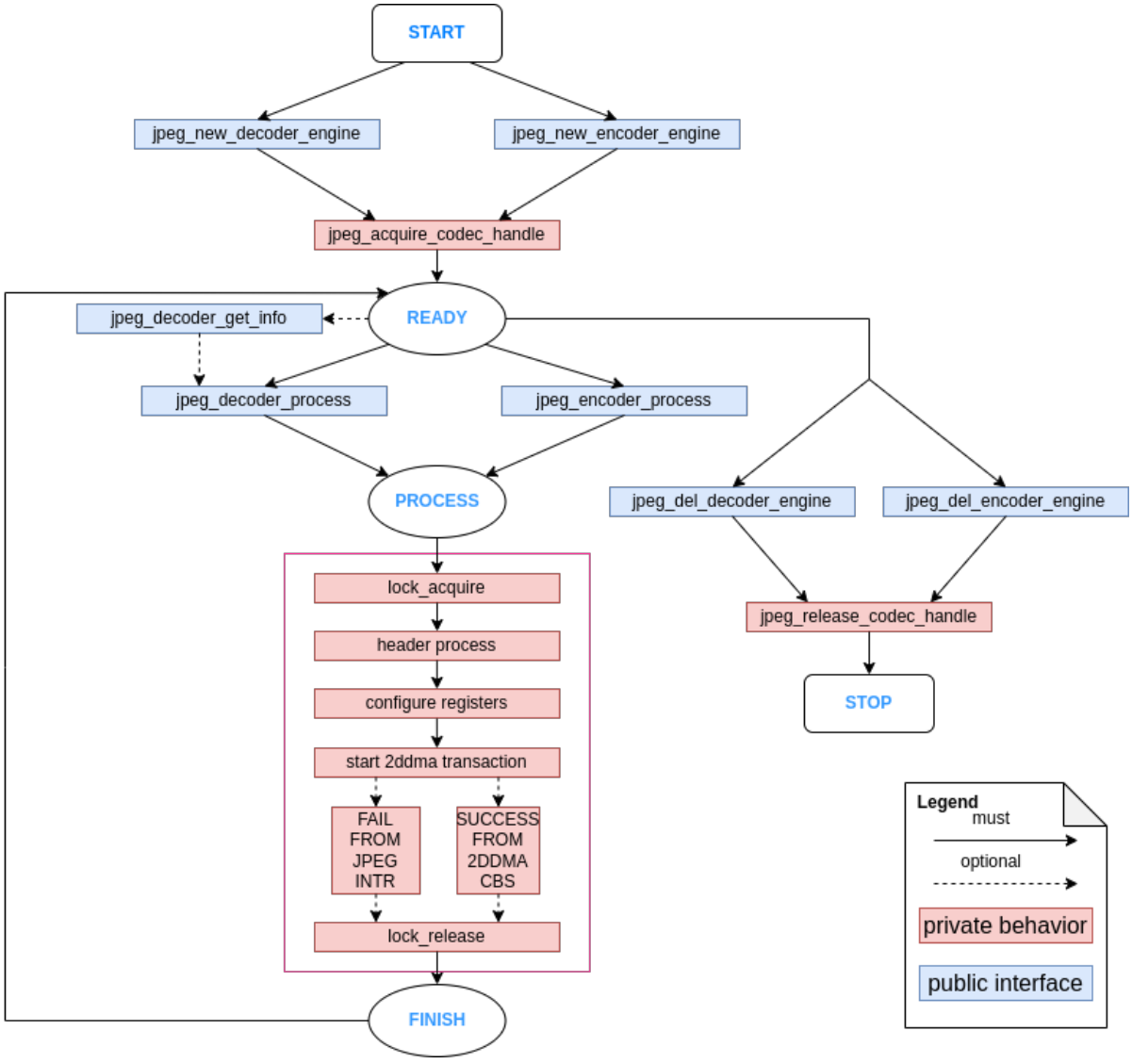


图 28: JPEG finite state machine

```
jpeg_decode_cfg_t decode_cfg_rgb = {
    .output_format = JPEG_DECODE_OUT_FORMAT_RGB888,
    .rgb_order = JPEG_DEC_RGB_ELEMENT_ORDER_BGR,
};

size_t tx_buffer_size;
size_t rx_buffer_size;

jpeg_decode_memory_alloc_cfg_t rx_mem_cfg = {
    .buffer_direction = JPEG_DEC_ALLOC_OUTPUT_BUFFER,
};

jpeg_decode_memory_alloc_cfg_t tx_mem_cfg = {
    .buffer_direction = JPEG_DEC_ALLOC_INPUT_BUFFER,
};

uint8_t *bit_stream = (uint8_t*)jpeg_alloc_decoder_mem(jpeg_size, &tx_mem_cfg, &tx_
↪buffer_size);
uint8_t *out_buf = (uint8_t*)jpeg_alloc_decoder_mem(1920 * 1088 * 3, &rx_mem_cfg, &
↪rx_buffer_size);

jpeg_decode_picture_info_t header_info;
ESP_ERROR_CHECK(jpeg_decoder_get_info(bit_stream, bit_stream_size, &header_info));
uint32_t out_size = 0;
ESP_ERROR_CHECK(jpeg_decoder_process(decoder_engine, &decode_cfg_rgb, bit_stream,␣
↪bit_stream_size, out_buf, &out_size));
```

There are some Tips that can help you use this driver more accurately:

1. In above code, you should make sure the *bit_stream* and *out_buf* should be aligned by certain rules. We provide a helper function *jpeg_alloc_decoder_mem()* to help you malloc a buffer which is aligned in both size and address.
2. The content of *bit_stream* buffer should not be changed until *jpeg_decoder_process()* returns.
3. The width and height of output picture would be 16 bytes aligned if original picture is formatted by YUV420 or YUV422. For example, if the input picture is 1080*1920, the output picture will be 1088*1920. That is the restriction of jpeg protocol. Please provide sufficient output buffer memory.

JPEG Encoder Engine After installing the JPEG encoder driver by *jpeg_new_encoder_engine()*, ESP32-P4 is ready to decode JPEG pictures by *jpeg_encoder_process()*. *jpeg_encoder_process()* is flexible for decoding different types of pictures by a configurable parameter called *jpeg_encode_cfg_t*.

The format conversions supported by this driver are listed in the table below:

Format of Original Image	Down sampling method
RGB565/RGB888	YUV444
	YUV422
	YUV420
GRAY	GRAY
YUV422	YUV422

Below is the example of code that encodes a 1080*1920 picture:

```
int raw_size_1080p = 0;/* Your raw image size */
jpeg_encode_cfg_t enc_config = {
    .src_type = JPEG_ENCODE_IN_FORMAT_RGB888,
    .sub_sample = JPEG_DOWN_SAMPLING_YUV422,
    .image_quality = 80,
    .width = 1920,
    .height = 1080,
```

(下页继续)

(续上页)

```
};

uint8_t *raw_buf_1080p = (uint8_t*)jpeg_alloc_encoder_mem(raw_size_1080p);
if (raw_buf_1080p == NULL) {
    ESP_LOGE(TAG, "alloc 1080p tx buffer error");
    return;
}
uint8_t *jpg_buf_1080p = (uint8_t*)jpeg_alloc_encoder_mem(raw_size_1080p / 10); //␣
↪Assume that compression ratio of 10 to 1
if (jpg_buf_1080p == NULL) {
    ESP_LOGE(TAG, "alloc jpg_buf_1080p error");
    return;
}

ESP_ERROR_CHECK(jpeg_encoder_process(jpeg_handle, &enc_config, raw_buf_1080p, raw_
↪size_1080p, jpg_buf_1080p, &jpg_size_1080p));
```

There are some Tips that can help you use this driver more accurately:

1. In above code, you should make sure the *raw_buf_1080p* and *jpg_buf_1080p* should aligned by calling `jpeg_alloc_encoder_mem()`.
2. The content of *raw_buf_1080p* buffer should not be changed until `jpeg_encoder_process()` returns.
3. The compression ratio depends on the chosen *image_quality* and the content of the image itself. Generally, a higher *image_quality* value obviously results in better image quality but a smaller compression ratio. As for the image content, it is hard to give any specific guidelines, so this question is out of the scope of this document. Generally, the baseline JPEG compression ratio can vary from 40:1 to 10:1. Please take the actual situation into account.

Performance Overview This section provides some measurements of the decoder and encoder performance. The data presented in the tables below gives the average values of decoding or encoding a randomly chosen picture fragments for 50 times. All tests were performed at a CPU frequency of 360MHz and a SPIRAM clock frequency of 200MHz. Only JPEG related code is run in this test, no other modules are involved (e.g. USB Camera, etc.).

Both decoder and encoder are not cause too much CPU involvement. Only header parse causes CPU source. Calculations related to JPEG compression, such as DCT, quantization, huffman encoding/decoding, etc., are done entirely in hardware.

JPEG decoder performance

JPEG Size		Pixel Format		Performance(fps)
Height	Width	in(Format of the already compressed image)	out(Format after decompressing)	
1080	1920	YUV422	RGB888/RGB565	48
720	1280	YUV422	RGB888/RGB565	109
480	800	YUV422	RGB888/RGB565	253
480	640	YUV422	RGB888/RGB565	307
480	320	YUV422	RGB888/RGB565	571
720	1280	GRAY	GRAY	161
480	800	YUV444	YUV444	129
480	800	YUV422	YUV444/YUV422	190
480	800	YUV420	YUV444/YUV420	253

JPEG encoder performance

JPEG Size		Pixel Format		Performance(fps)
Height	Width	in(Format of Original Image)	out(Down sampling method)	
1080	1920	RGB888	YUV422	26
1080	1920	RGB565	YUV422	36
1080	1920	RGB565	YUV420	40
1080	1920	RGB565	YUV444	24
1080	1920	RGB888	YUV422	26
720	1280	RGB565	YUV420	88
720	1280	RGB565	YUV444	55
720	1280	RGB565	YUV422	81
480	800	RGB888	YUV420	142
480	640	RGB888	YUV420	174
480	320	RGB888	YUV420	315
720	1280	GRAY	GRAY	163
480	800	YUV422	YUV422	146

Pixel Storage Layout for Different Color Formats The encoder and decoder described in this guide use the same uncompressed raw image formats (RGB, YUV). Therefore, the encoder and decoder are not discussed separately in this section. The pixel layout of the following formats applies to the input direction of the encoder and the output direction of the decoder (if supported). The specific pixel layout is shown in the following figure:

RGB888 In the following picture, each small block means one bit.

图 29: RGB888 pixel order

For RGB888, the order can be changed via `jpeg_decode_cfg_t::rgb_order` sets the pixel to *RGB* order.

图 30: RGB888 pixel big endian order

RGB565 In the following picture, each small block means one bit.

图 31: RGB565 pixel order

For RGB565, the order can be changed via `jpeg_decode_cfg_t::rgb_order` sets the pixel to *RGB* order.

YUV444 In the following picture, each small block means one byte.

YUV422 In the following picture, each small block means one byte.

YUV420 In the following picture, each small block means one byte.

Thread Safety The factory function `jpeg_new_decoder_engine()`, `jpeg_decoder_get_info()`, `jpeg_decoder_process()`, and `jpeg_del_decoder_engine()` are guaranteed to be thread safe by the driver, which means, user can call them from different RTOS tasks without protection by extra locks.

图 32: RGB565 pixel big endian order

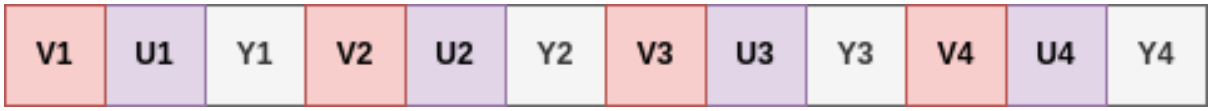


图 33: YUV444 pixel order

Power Management When power management is enabled (i.e., *CONFIG_PM_ENABLE* is set), the system needs to adjust or stop the source clock of JPEG to enter Light-sleep, thus potentially changing the JPEG decoder or encoder process. This might lead to unexpected behavior in hardware calculation. To prevent such issues, entering light sleep is disabled for the time when JPEG encoder or decoder is working.

Whenever the user is decoding or encoding via JPEG (i.e., calling `jpeg_encoder_process()` or `jpeg_decoder_process()`), the driver guarantees that the power management lock is acquired by setting it to `esp_pm_lock_type_t::ESP_PM_NO_LIGHT_SLEEP`. Once the encoding or decoding is finished, the driver releases the lock and the system can enter light sleep.

Kconfig Options

- *CONFIG_JPEG_ENABLE_DEBUG_LOG* is used to enable the debug log at the cost of increased firmware binary size.

Maintainers' Notes

The JPEG driver usage of hardware resources and its process workflow are shown in the following graph:

Application Examples

- JPEG decoder application example: peripherals/jpeg/jpeg_decode.
- JPEG encoder application example: peripherals/jpeg/jpeg_encode.

API Reference

Header File

- components/esp_driver_jpeg/include/driver/jpeg_decode.h
- This header file can be included with:

```
#include "driver/jpeg_decode.h"
```

- This header file is a part of the API provided by the `esp_driver_jpeg` component. To declare that your component depends on `esp_driver_jpeg`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_jpeg
```

or

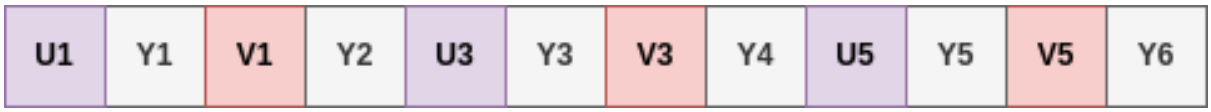


图 34: YUV422 pixel order

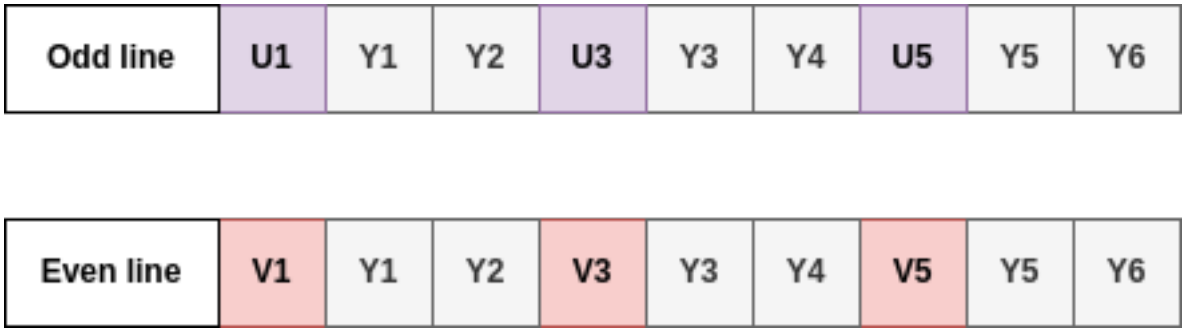


图 35: YUV420 pixel order

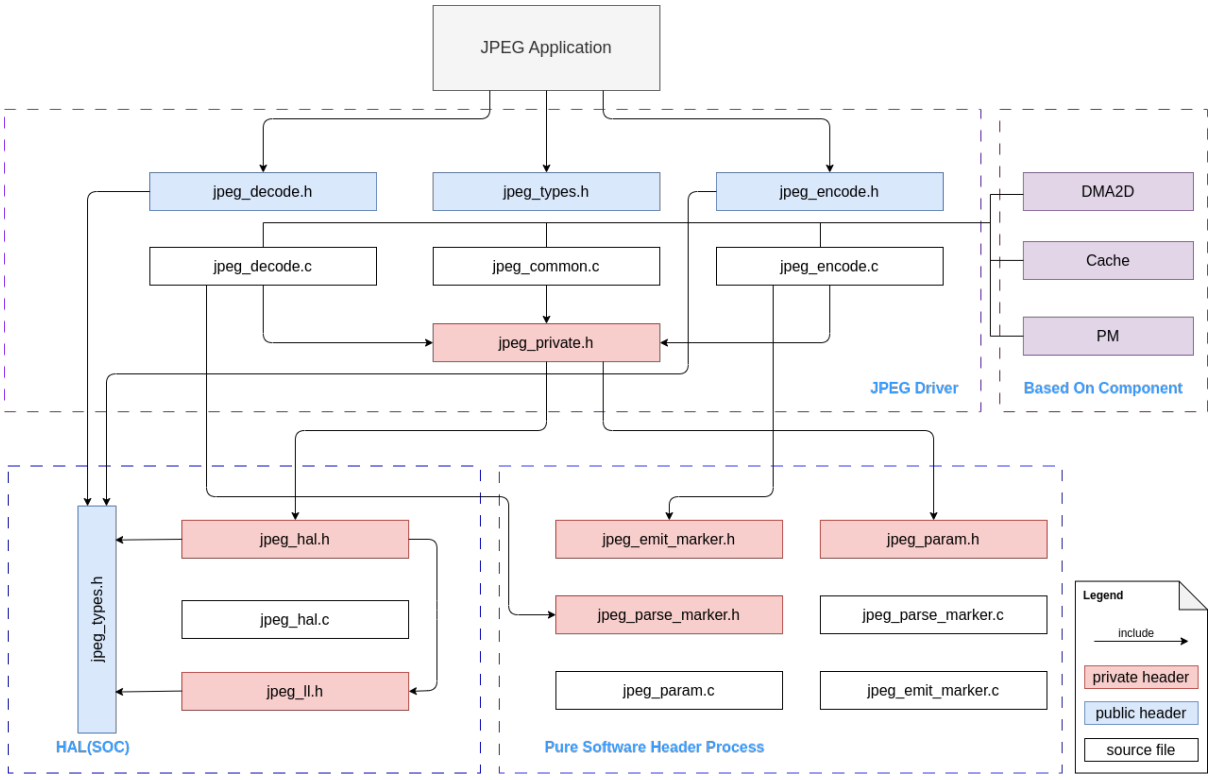


图 36: JPEG driver file structure

```
PRIV_REQUIRES esp_driver_jpeg
```

Functions

esp_err_t **jpeg_new_decoder_engine**(const *jpeg_decode_engine_cfg_t* *dec_eng_cfg, *jpeg_decoder_handle_t* *ret_decoder)

Acquire a JPEG decode engine with the specified configuration.

This function acquires a JPEG decode engine with the specified configuration. The configuration parameters are provided through the `dec_eng_cfg` structure, and the resulting JPEG decoder handle is returned through the `ret_decoder` pointer.

参数

- **dec_eng_cfg** -- **[in]** Pointer to the JPEG decode engine configuration.
- **ret_decoder** -- **[out]** Pointer to a variable that will receive the JPEG decoder handle.

返回

- ESP_OK: JPEG decoder initialized successfully.
- ESP_ERR_INVALID_ARG: JPEG decoder initialization failed because of invalid argument.
- ESP_ERR_NO_MEM: Create JPEG decoder failed because of out of memory.

esp_err_t **jpeg_decoder_get_info**(const uint8_t *bit_stream, uint32_t stream_size, *jpeg_decode_picture_info_t* *picture_info)

Helper function for getting information about a JPEG image.

This function analyzes the provided JPEG image data and retrieves information about the image, such as its width, height. The image data is specified by the `bit_stream` pointer and the `stream_size` parameter. The resulting image information is returned through the `picture_info` structure.

备注： This function doesn't depend on any jpeg hardware, it helps user to know jpeg information from jpeg header. For example, user can get picture width and height via this function and malloc a reasonable size buffer for jpeg engine process.

参数

- **bit_stream** -- **[in]** Pointer to the buffer containing the JPEG image data.
- **stream_size** -- **[in]** Size of the JPEG image data in bytes. Note that parse beginning partial of picture also works, but the beginning partial should be enough given.
- **picture_info** -- **[out]** Pointer to the structure that will receive the image information.

返回

- ESP_OK: JPEG decoder get jpg image header successfully.
- ESP_ERR_INVALID_ARG: JPEG decoder get header info failed because of invalid argument.

esp_err_t **jpeg_decoder_process**(*jpeg_decoder_handle_t* decoder_engine, const *jpeg_decode_cfg_t* *decode_cfg, const uint8_t *bit_stream, uint32_t stream_size, uint8_t *decode_outbuf, uint32_t outbuf_size, uint32_t *out_size)

Process a JPEG image with the specified decoder instance.

This function processes the provided JPEG image data using the specified JPEG decoder instance. The input JPEG image data is specified by the `bit_stream` pointer and the `stream_size` parameter. The resulting decoded image data is written to the `decode_outbuf` buffer, and the length of the output image data is returned through the `out_size` pointer.

备注： 1.Please make sure that the content of `bit_stream` pointer cannot be modified until this function returns. 2.Please note that the output size of image is always the multiple of 16 depends on protocol of JPEG.

参数

- **decoder_engine** -- **[in]** Handle of the JPEG decoder instance to use for processing.
- **decode_cfg** -- **[in]** Config structure of decoder.
- **bit_stream** -- **[in]** Pointer to the buffer containing the input JPEG image data.
- **stream_size** -- **[in]** Size of the input JPEG image data in bytes.
- **decode_outbuf** -- **[in]** Pointer to the buffer that will receive the decoded image data.
- **outbuf_size** -- **[in]** The size of `decode_outbuf`
- **out_size** -- **[out]** Pointer to a variable that will receive the length of the output image data.

返回

- ESP_OK: JPEG decoder process successfully.
- ESP_ERR_INVALID_ARG: JPEG decoder process failed because of invalid argument.

esp_err_t **jpeg_del_decoder_engine**(*jpeg_decoder_handle_t* decoder_engine)

Release resources used by a JPEG decoder instance.

This function releases the resources used by the specified JPEG decoder instance. The decoder instance is specified by the `decoder_engine` parameter.

参数 **decoder_engine** -- Handle of the JPEG decoder instance to release resources for.

返回

- ESP_OK: Delete JPEG decoder successfully.
- ESP_ERR_INVALID_ARG: Delete JPEG decoder failed because of invalid argument.

void ***jpeg_alloc_decoder_mem**(size_t size, const *jpeg_decode_memory_alloc_cfg_t* *mem_cfg, size_t *allocated_size)

A helper function to allocate memory space for JPEG decoder.

参数

- **size** -- **[in]** The size of memory to allocate.
- **mem_cfg** -- **[in]** Memory configuration for memory allocation
- **allocated_size** -- **[out]** Actual allocated buffer size.

返回 Pointer to the allocated memory space, or NULL if allocation fails.

Structures

struct **jpeg_decode_cfg_t**

Configuration parameters for a JPEG decoder image process.

Public Members

jpeg_dec_output_format_t **output_format**

JPEG decoder output format

jpeg_dec_rgb_element_order_t **rgb_order**

JPEG decoder output order

jpeg_yuv_rgb_conv_std_t **conv_std**

JPEG decoder yuv->rgb standard

struct **jpeg_decode_engine_cfg_t**

Configuration parameters for the JPEG decoder engine.

Public Members

int **intr_priority**

JPEG interrupt priority, if set to 0, driver will select the default priority (1,2,3).

int **timeout_ms**

JPEG timeout threshold for handling a picture, should larger than valid decode time in ms. For example, for 30fps decode, this value must larger than 34. -1 means wait forever

struct **jpeg_decode_picture_info_t**

Structure for jpeg decode header.

Public Members

uint32_t **width**

Number of pixels in the horizontal direction

uint32_t **height**

Number of pixels in the vertical direction

jpeg_down_sampling_type_t **sample_method**

compressed JPEG picture sampling method

struct **jpeg_decode_memory_alloc_cfg_t**

JPEG decoder memory allocation config.

Public Members

jpeg_dec_buffer_alloc_direction_t **buffer_direction**

Buffer direction for jpeg decoder memory allocation

Header File

- components/esp_driver_jpeg/include/driver/jpeg_encode.h
- This header file can be included with:

```
#include "driver/jpeg_encode.h"
```

- This header file is a part of the API provided by the `esp_driver_jpeg` component. To declare that your component depends on `esp_driver_jpeg`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_jpeg
```

or

```
PRIV_REQUIRES esp_driver_jpeg
```

Functions

esp_err_t **jpeg_new_encoder_engine**(const *jpeg_encode_engine_cfg_t* *enc_eng_cfg, *jpeg_encoder_handle_t* *ret_encoder)

Allocate JPEG encoder.

参数

- **enc_eng_cfg** -- **[in]** config for jpeg encoder
- **ret_encoder** -- **[out]** handle for jpeg encoder

返回

- ESP_OK: JPEG encoder initialized successfully.
- ESP_ERR_INVALID_ARG: JPEG encoder initialization failed because of invalid argument.
- ESP_ERR_NO_MEM: Create JPEG encoder failed because of out of memory.

esp_err_t **jpeg_encoder_process**(*jpeg_encoder_handle_t* encoder_engine, const *jpeg_encode_cfg_t* *encode_cfg, const uint8_t *encode_inbuf, uint32_t inbuf_size, uint8_t *encode_outbuf, uint32_t outbuf_size, uint32_t *out_size)

Process encoding of JPEG data using the specified encoder engine.

This function processes the encoding of JPEG data using the provided encoder engine and configuration. It takes an input buffer containing the raw image data, performs encoding based on the configuration settings, and outputs the compressed bitstream.

参数

- **encoder_engine** -- **[in]** Handle to the JPEG encoder engine to be used for encoding.
- **encode_cfg** -- **[in]** Pointer to the configuration structure for the JPEG encoding process.
- **encode_inbuf** -- **[in]** Pointer to the input buffer containing the raw image data.
- **inbuf_size** -- **[in]** Size of the input buffer in bytes.
- **encode_outbuf** -- **[in]** Pointer to the output buffer where the compressed bitstream will be stored.
- **outbuf_size** -- **[in]** The size of output buffer.
- **out_size** -- **[out]** Pointer to a variable where the size of the output bitstream will be stored.

返回

- ESP_OK: JPEG encoder process successfully.
- ESP_ERR_INVALID_ARG: JPEG encoder process failed because of invalid argument.
- ESP_ERR_TIMEOUT: JPEG encoder process timeout.

esp_err_t **jpeg_del_encoder_engine**(*jpeg_encoder_handle_t* encoder_engine)

Release resources used by a JPEG encoder instance.

This function releases the resources used by the specified JPEG encoder instance. The encoder instance is specified by the `encoder_engine` parameter.

参数 **encoder_engine** -- **[in]** Handle of the JPEG encoder instance to release resources for.

返回

- ESP_OK: Delete JPEG encoder successfully.
- ESP_ERR_INVALID_ARG: Delete JPEG encoder failed because of invalid argument.

void ***jpeg_alloc_encoder_mem**(size_t size, const *jpeg_encode_memory_alloc_cfg_t* *mem_cfg, size_t *allocated_size)

A helper function to allocate memory space for JPEG encoder.

参数

- **size** -- **[in]** The size of memory to allocate.
- **mem_cfg** -- **[in]** Memory configuration for memory allocation
- **allocated_size** -- **[out]** Actual allocated buffer size.

返回 Pointer to the allocated memory space, or NULL if allocation fails.

Structures

struct **jpeg_encode_cfg_t**

JPEG encoder configure structure.

Public Members

uint32_t **height**

Number of pixels in the horizontal direction

uint32_t **width**

Number of pixels in the vertical direction

jpeg_enc_input_format_t **src_type**

Source type of raw image to be encoded, see `jpeg_enc_src_type_t`

jpeg_down_sampling_type_t **sub_sample**

JPEG subsampling method

uint32_t **image_quality**

JPEG compressing quality, value from 1-100, higher value means higher quality

struct **jpeg_encode_engine_cfg_t**

Configuration parameters for the JPEG encode engine.

Public Members

int **intr_priority**

JPEG interrupt priority, if set to 0, driver will select the default priority (1,2,3).

int **timeout_ms**

JPEG timeout threshold for handling a picture, should larger than valid encode time in ms. For example, for 30fps encode, this value must larger than 34. -1 means wait forever

struct **jpeg_encode_memory_alloc_cfg_t**

JPEG encoder memory allocation config.

Public Members

jpeg_enc_buffer_alloc_direction_t **buffer_direction**

Buffer direction for jpeg encoder memory allocation

Header File

- components/esp_driver_jpeg/include/driver/jpeg_types.h
- This header file can be included with:

```
#include "driver/jpeg_types.h"
```

- This header file is a part of the API provided by the `esp_driver_jpeg` component. To declare that your component depends on `esp_driver_jpeg`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_jpeg
```

or

```
PRIV_REQUIRES esp_driver_jpeg
```

Type Definitions

typedef struct jpeg_decoder_t ***jpeg_decoder_handle_t**

Type of jpeg decoder handle.

typedef struct jpeg_encoder_t ***jpeg_encoder_handle_t**

Type of jpeg encoder handle.

Enumerations

enum **jpeg_dec_output_format_t**

Enumeration for jpeg output format.

Values:

enumerator **JPEG_DECODE_OUT_FORMAT_RGB888**

output RGB888 format

enumerator **JPEG_DECODE_OUT_FORMAT_RGB565**

output RGB565 format

enumerator **JPEG_DECODE_OUT_FORMAT_GRAY**

output the gray picture

enumerator **JPEG_DECODE_OUT_FORMAT_YUV444**

output yuv444 format

enumerator **JPEG_DECODE_OUT_FORMAT_YUV422**

output yuv422 format

enumerator **JPEG_DECODE_OUT_FORMAT_YUV420**

output yuv420 format

enum **jpeg_yuv_rgb_conv_std_t**

Enumeration for YUV and RGB conversion standard.

Values:

enumerator **JPEG_YUV_RGB_CONV_STD_BT601**

BT601

enumerator **JPEG_YUV_RGB_CONV_STD_BT709**

BT709

enum **jpeg_dec_rgb_element_order_t**

Enumeration for jpeg big/small endian output.

Values:

enumerator **JPEG_DEC_RGB_ELEMENT_ORDER_BGR**

Output the color component in small endian

enumerator **JPEG_DEC_RGB_ELEMENT_ORDER_RGB**

Output the color component in big endian

enum **jpeg_dec_buffer_alloc_direction_t**

Enumeration for jpeg decoder alloc buffer direction.

Values:

enumerator **JPEG_DEC_ALLOC_INPUT_BUFFER**

Alloc the picture input buffer, (compressed format in decoder)

enumerator **JPEG_DEC_ALLOC_OUTPUT_BUFFER**

Alloc the picture output buffer, (decompressed format in decoder)

enum **jpeg_enc_input_format_t**

Enumeration for jpeg input format.

Values:

enumerator **JPEG_ENCODE_IN_FORMAT_RGB888**

input RGB888 format

enumerator **JPEG_ENCODE_IN_FORMAT_RGB565**

input RGB565 format

enumerator **JPEG_ENCODE_IN_FORMAT_GRAY**

input GRAY format

enumerator **JPEG_ENCODE_IN_FORMAT_YUV422**

input YUV422 format

enum **jpeg_enc_buffer_alloc_direction_t**

Enumeration for jpeg encoder alloc buffer direction.

Values:

enumerator **JPEG_ENC_ALLOC_INPUT_BUFFER**

Alloc the picture input buffer, (decompressed format in encoder)

enumerator **JPEG_ENC_ALLOC_OUTPUT_BUFFER**

Alloc the picture output buffer, (compressed format in encoder)

Header File

- components/hal/include/hal/jpeg_types.h
- This header file can be included with:

```
#include "hal/jpeg_types.h"
```

Unions

union **jpeg_huffman_table_info_t**

#include <jpeg_types.h> Structure for huffman information.

Public Members

uint8_t **id**

Huffman table id.

uint8_t **type**

Huffman table type.

struct *jpeg_huffman_table_info_t*::[anonymous] **[anonymous]**

uint8_t **info**

Information of huffman table.

Structures

struct **jpeg_component_factor_t**

Structure for recording factor of component.

Public Members

uint32_t **horizontal**

horizontal factor

uint32_t **vertical**

vertical factor

Macros

JPEG_HUFFMAN_BITS_LEN_TABLE_LEN

JPEG_HUFFMAN_AC_VALUE_TABLE_LEN

JPEG_HUFFMAN_DC_VALUE_TABLE_LEN

JPEG_QUANTIZATION_TABLE_LEN

JPEG_COMPONENT_NUMBER_MAX

DHT_TC_NUM

DHT_TH_NUM

JPEG_DOWN_SAMPLING_NUM

Enumerations

enum **jpeg_codec_mode_t**

Enum for JPEG codec working mode.

Values:

enumerator **JPEG_CODEC_ENCODER**

Encode mode.

enumerator **JPEG_CODEC_DECODER**

Decode mode.

enum **jpeg_sample_mode_t**

Enum for JPEG sampling mode.

Values:

enumerator **JPEG_SAMPLE_MODE_YUV444**

sample in YUV444

enumerator **JPEG_SAMPLE_MODE_YUV422**

sample in YUV422

enumerator **JPEG_SAMPLE_MODE_YUV420**

sample in YUV420

enum **jpeg_down_sampling_type_t**

Enumeration for jpeg decoder sample methods.

Values:

enumerator **JPEG_DOWN_SAMPLING_YUV444**

Sample by YUV444

enumerator **JPEG_DOWN_SAMPLING_YUV422**

Sample by YUV422

enumerator **JPEG_DOWN_SAMPLING_YUV420**

Sample by YUV420

enumerator **JPEG_DOWN_SAMPLING_GRAY**

Sample the gray picture

enum **jpeg_enc_src_type_t**

JPEG encoder source formats.

Values:

enumerator **JPEG_ENC_SRC_RGB888**

JPEG encoder source RGB888

enumerator **JPEG_ENC_SRC_YUV422**

JPEG encoder source YUV422

enumerator **JPEG_ENC_SRC_RGB565**

JPEG encoder source RGB565

enumerator **JPEG_ENC_SRC_GRAY**

JPEG encoder source GRAY

2.5.33 温度传感器

简介

ESP32-P4 内置传感器，用于测量芯片内部的温度。该温度传感器模组包含一个 8 位 Sigma-Delta 模拟-数字转换器 (ADC) 和一个数字-模拟转换器 (DAC)，可以补偿测量结果，减少温度测量的误差。

由于硬件限制，温度传感器存在预定义的测量范围及其对应误差，详见下表：

预定义测量范围 (°C)	测量误差 (°C)
50 ~ 125	< 3
20 ~ 100	< 2
-10 ~ 80	< 1
-30 ~ 50	< 2
-40 ~ 20	< 3

备注： 温度传感器主要用于测量芯片内部的温度变化。芯片内部温度通常高于环境温度，并且受到微控制器的时钟频率或 I/O 负载、外部散热环境等因素影响。

功能概述

下文将分节概述温度传感器的功能：

- 资源分配 - 介绍了部分参数，设置这些参数可以获取温度传感器句柄；还介绍了在温度传感器完成工作后如何回收资源。
- 启用及禁用温度传感器 - 介绍如何启用及禁用温度传感器。
- 获取测量的温度值 - 介绍如何获取实时温度值。
- 安装温度阈值回调函数 - 介绍如何注册温度阈值回调函数。
- 电源管理 - 介绍更改功耗模式（如 Light-sleep 模式）对温度传感器造成的影响。
- *IRAM* 安全 - 介绍在禁用 cache 时如何提高温度传感器的性能。
- 线程安全 - 介绍如何使驱动程序具备线程安全性。
- *ETM* 事件和任务 - 介绍哪些事件和任务可以连接到 ETM 通道。

资源分配 ESP32-P4 只有一个内置温度传感器硬件。`temperature_sensor_handle_t` 表示温度传感器模块，该变量也是不同函数之间的纽带。通过使用相同的`temperature_sensor_handle_t` 变量，可以在不同的函数调用中访问和修改温度传感器属性，以控制和管理温度传感器。该变量会作为温度 API 的参数，携带有关硬件和配置的信息，你只需创建类型为`temperature_sensor_handle_t` 的指针，并将其传递给所需 API。

请在安装内置温度传感器模块前评估测量环境的温度范围。例如，如果在室内测量，环境温度可能在 10 °C ~ 30 °C；如果在灯泡中测量，环境温度则可能在 60 °C ~ 110 °C。在环境温度范围的基础上，请先根据以下值定义配置结构体*temperature_sensor_config_t*，再安装内置温度传感器：

- range_min：所测量温度范围的最小值。
- range_max：所测量温度范围的最大值。

设置好温度范围后，将配置结构体传递给*temperature_sensor_install()*，该函数将创建温度传感器模块并返回句柄。

如前文所述，不同测量范围对应不同测量误差。然而你无需自行比对测量误差，乐鑫提供了一个内部机制，可以根据所给温度范围选择最小误差。

温度传感器使用完毕后，请调用*temperature_sensor_uninstall()* 释放相应资源。

创建温度传感器句柄

- 第 1 步：评估测量范围。本示例的温度范围为 20 °C ~ 50 °C。
- 第 2 步：配置测量范围，获取温度传感器句柄。

```
temperature_sensor_handle_t temp_handle = NULL;
temperature_sensor_config_t temp_sensor_config = TEMPERATURE_SENSOR_CONFIG_
↪DEFAULT(20, 50);
ESP_ERROR_CHECK(temperature_sensor_install(&temp_sensor_config, &temp_handle));
```

启用及禁用温度传感器

1. 调用*temperature_sensor_enable()* 启用温度传感器。此时，内部温度传感器回路开始工作，驱动程序从初始化状态转为启用状态。
2. 调用*temperature_sensor_disable()* 禁用温度传感器。

获取测量的温度值 通过*temperature_sensor_enable()* 启用温度传感器后，可以调用*temperature_sensor_get_celsius()* 获取当前测量的温度值。

```
// 启用温度传感器
ESP_ERROR_CHECK(temperature_sensor_enable(temp_handle));
// 获取传输的传感器数据
float tsens_out;
ESP_ERROR_CHECK(temperature_sensor_get_celsius(temp_handle, &tsens_out));
printf("Temperature in %f °C\n", tsens_out);
// 温度传感器使用完毕后，禁用温度传感器，节约功耗
ESP_ERROR_CHECK(temperature_sensor_disable(temp_handle));
```

安装温度阈值回调函数 ESP32-P4 支持自动触发温度传感器，持续监测内部温度，内部温度达到给定阈值时将触发中断。因此，可以安装中断回调函数执行所需操作，如报警、重启等。下文介绍了如何准备阈值回调函数。

- 函数*temperature_sensor_event_callbacks_t::on_threshold*在中断服务程序 (ISR) 的上下文中调用，请确保该函数不会涉及 block 操作。为此，可以检查调用 API 的后缀，确保仅从函数内调用具有 ISR 后缀的 FreeRTOS API 等。函数原型在*temperature_thres_cb_t* 中声明。

通过参数 user_arg 可以将自定义上下文保存到*temperature_sensor_register_callbacks()* 中，用户数据将直接传递给回调函数。

```
IRAM_ATTR static bool temp_sensor_monitor_cbs(temperature_sensor_handle_t tsens,␣
↪const temperature_sensor_threshold_event_data_t *edata, void *user_data)
{
    ESP_DRAM_LOGI("tsens", "Temperature value is higher or lower than threshold,␣
↪value is %d\n...\n\n", edata->celsius_value);
    return false;
```

(下页继续)

(续上页)

```
}

// 配置回调函数
temperature_sensor_abs_threshold_config_t threshold_cfg = {
    .high_threshold = 50,
    .low_threshold = -10,
};
// 设置监控阈值
temperature_sensor_set_absolute_threshold(temp_sensor, &threshold_cfg);
// 注册中断回调函数
temperature_sensor_event_callbacks_t cbs = {
    .on_threshold = temp_sensor_monitor_cbs,
};
// 安装温度回调函数
temperature_sensor_register_callbacks(temp_sensor, &cbs, NULL);
```

电源管理 由于温度传感器不使用 APB 时钟，无论是否激活 CONFIG_PM_ENABLE 启用电源管理，温度传感器仍可以继续工作。

IRAM 安全 默认情况下，禁用 cache 时，写入/擦除 flash 等原因将导致温度传感器中断延迟，事件回调函数也将延迟执行。在实时应用程序中，应避免此类情况。

因此，可以启用 Kconfig 选项*CONFIG_TEMP_SENSOR_ISR_IRAM_SAFE*，该选项：

1. 支持在禁用 cache 时启用所需中断
2. 支持将 ISR 使用的所有函数存放在 IRAM 中

启用该选项可以保证 cache 禁用时的中断运行，但会占用更多的 IRAM。

线程安全 温度传感器中并未添加任何确保线程安全的额外保护，因为温度传感器通常只在一个任务中调用。如果要在不同任务中使用该驱动程序，请设置额外的锁进行保护。

ETM 事件和任务 温度传感器能够生成事件，这些事件可以与:doc:ETM </api-reference/peripherals/etm> 模块进行交互。:cpp:type:temperature_sensor_etm_event_type_t 中列出了所有支持的事件。可以调用:cpp:func:temperature_sensor_new_etm_event 来获取相应的 ETM 事件句柄。:cpp:type:temperature_sensor_etm_task_type_t 中列出了所有支持的任务。可以调用:cpp:func:temperature_sensor_new_etm_task 来获取相应的 ETM 任务句柄。

备注:

- 对于 temperature_sensor_etm_event_type_t::event_type 的*TEMPERATURE_SENSOR_EVENT_OVER_LIMIT* 取决于首先设置的阈值类型。如果是通过*temperature_sensor_set_absolute_threshold()* 设置了绝对阈值，那么*TEMPERATURE_SENSOR_EVENT_OVER_LIMIT* 将指代绝对阈值。同样，如果是通过*temperature_sensor_set_delta_threshold()* 设置了增量阈值，那么*TEMPERATURE_SENSOR_EVENT_OVER_LIMIT* 将指代增量阈值。

有关如何将事件和任务连接到 ETM 通道的详细信息，请参阅:doc:ETM </api-reference/peripherals/etm> 文档。

意外情况

1. 从芯片获取的温度值通常与环境温度不同，因为温度传感器内置于芯片，从某种程度来说，温度传感器测量的是芯片内的温度。

2. 安装温度传感器失败时，如果驱动程序打印的错误信息为 `the boundary you gave cannot meet the range of internal temperature sensor`，说明内置温度传感器温度测量范围的限制影响了安装过程，该错误通常由以下几种不正确的*temperature_sensor_config_t* 配置造成：
   (1) 超出温度测量范围，如 200 °C ~ 300 °C。
   (2) 超过了预定义测量范围的界限，如 40 °C ~ 110 °C。

应用示例

- 读取温度传感器测量值：peripherals/temperature_sensor/temp_sensor。
- 监测温度传感器测量值：peripherals/temperature_sensor/temp_sensor_monitor。

API 参考

Header File

- components/esp_driver_tsens/include/driver/temperature_sensor.h
- This header file can be included with:

```
#include "driver/temperature_sensor.h"
```

- This header file is a part of the API provided by the `esp_driver_tsens` component. To declare that your component depends on `esp_driver_tsens`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_tsens
```

or

```
PRIV_REQUIRES esp_driver_tsens
```

Functions

esp_err_t **temperature_sensor_install**(const *temperature_sensor_config_t* *tsens_config, *temperature_sensor_handle_t* *ret_tsens)

Install temperature sensor driver.

参数
- **tsens_config** -- Pointer to config structure.
- **ret_tsens** -- Return the pointer of temperature sensor handle.

返回
- ESP_OK if succeed

esp_err_t **temperature_sensor_uninstall**(*temperature_sensor_handle_t* tsens)

Uninstall the temperature sensor driver.

参数 **tsens** -- The handle created by `temperature_sensor_install()`.

返回
- ESP_OK if succeed.

esp_err_t **temperature_sensor_enable**(*temperature_sensor_handle_t* tsens)

Enable the temperature sensor.

参数 **tsens** -- The handle created by `temperature_sensor_install()`.

返回
- ESP_OK Success
- ESP_ERR_INVALID_STATE if temperature sensor is enabled already.

esp_err_t **temperature_sensor_disable**(*temperature_sensor_handle_t* tsens)

Disable temperature sensor.

参数 **tsens** -- The handle created by `temperature_sensor_install()`.

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE if temperature sensor is not enabled yet.

esp_err_t **temperature_sensor_get_celsius**(*temperature_sensor_handle_t* tsens, float *out_celsius)

Read temperature sensor data that is converted to degrees Celsius.

备注: Should not be called from interrupt.

参数

- **tsens** -- The handle created by `temperature_sensor_install()`.
- **out_celsius** -- The measure output value.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG invalid arguments
- ESP_ERR_INVALID_STATE Temperature sensor is not enabled yet.
- ESP_FAIL Parse the sensor data into ambient temperature failed (e.g. out of the range).

esp_err_t **temperature_sensor_set_absolute_threshold**(*temperature_sensor_handle_t* tsens, const *temperature_sensor_abs_threshold_config_t* *abs_cfg)

Set temperature sensor absolute mode automatic monitor.

备注: This function should not be called with `temperature_sensor_set_delta_threshold`.

参数

- **tsens** -- The handle created by `temperature_sensor_install()`.
- **abs_cfg** -- Configuration of temperature sensor absolute mode interrupt, see *temperature_sensor_abs_threshold_config_t*.

返回

- ESP_OK: Set absolute threshold successfully.
- ESP_ERR_INVALID_STATE: Set absolute threshold failed because of wrong state.
- ESP_ERR_INVALID_ARG: Set absolute threshold failed because of invalid argument.

esp_err_t **temperature_sensor_set_delta_threshold**(*temperature_sensor_handle_t* tsens, const *temperature_sensor_delta_threshold_config_t* *delta_cfg)

Set temperature sensor differential mode automatic monitor.

备注: This function should not be called with `temperature_sensor_set_absolute_threshold`

参数

- **tsens** -- The handle created by `temperature_sensor_install()`.
- **delta_cfg** -- Configuration of temperature sensor delta mode interrupt, see *temperature_sensor_delta_threshold_config_t*.

返回

- ESP_OK: Set differential value threshold successfully.
- ESP_ERR_INVALID_STATE: Set absolute threshold failed because of wrong state.
- ESP_ERR_INVALID_ARG: Set differential value threshold failed because of invalid argument.

esp_err_t **temperature_sensor_register_callbacks**(*temperature_sensor_handle_t* tsens, const *temperature_sensor_event_callbacks_t* *cbs, void *user_arg)

Install temperature sensor interrupt callback. Temperature sensor interrupt will be enabled at same time.

参数

- **tsens** -- The handle created by `temperature_sensor_install()`.
- **cbs** -- Pointer to the group of temperature sensor interrupt callbacks.
- **user_arg** -- Callback argument.

返回

- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

Structures

struct **temperature_sensor_config_t**

Configuration of measurement range for the temperature sensor.

备注: If you see the log `the boundary you gave cannot meet the range of internal temperature sensor`. You may need to refer to predefined range listed doc `api-reference/peripherals/Temperature sensor`.

Public Members

int **range_min**

the minimum value of the temperature you want to test

int **range_max**

the maximum value of the temperature you want to test

temperature_sensor_clk_src_t **clk_src**

the clock source of the temperature sensor.

struct **temperature_sensor_threshold_event_data_t**

Temperature sensor event data.

Public Members

int **celsius_value**

Celsius value in interrupt callback.

temperature_val_intr_condition_t **intr_condition**

Can be used to judge temperature sensor interrupts in which reason

struct **temperature_sensor_event_callbacks_t**

Group of temperature sensor callback functions, all of them will be run in ISR.

Public Members

temperature_thres_cb_t **on_threshold**

Temperature value interrupt callback

struct **temperature_sensor_abs_threshold_config_t**

Config options for temperature value absolute interrupt.

Public Members

float **high_threshold**

High threshold value(Celsius). Interrupt will be triggered if temperature value is higher than this value

float **low_threshold**

Low threshold value(Celsius). Interrupt will be triggered if temperature value is lower than this value

struct **temperature_sensor_delta_threshold_config_t**

Config options for temperature value delta interrupt.

Public Members

float **increase_delta**

Interrupt will be triggered if the temperature increment of two consecutive samplings if larger than `increase_delta`

float **decrease_delta**

Interrupt will be triggered if the temperature decrement of two consecutive samplings if smaller than `decrease_delta`

Macros

TEMPERATURE_SENSOR_CONFIG_DEFAULT(min, max)

temperature_sensor_config_t default constructor

Type Definitions

typedef struct temperature_sensor_obj_t ***temperature_sensor_handle_t**

Type of temperature sensor driver handle.

typedef bool (***temperature_thres_cb_t**)(*temperature_sensor_handle_t* tsens, const *temperature_sensor_threshold_event_data_t* *edata, void *user_data)

Callback for temperature sensor threshold interrupt.

Param tsens **[in]** The handle created by `temperature_sensor_install()`.
Param edata **[in]** temperature sensor event data, fed by driver.
Param user_data **[in]** User data, set in `temperature_sensor_register_callbacks()`.
Return Whether a high priority task has been waken up by this function.

Enumerations

enum **temperature_val_intr_condition_t**

Enum for temperature sensor interrupt condition.

Values:

enumerator **TEMPERATURE_VAL_HIGHER_THAN_HIGH_THRESHOLD**

temperature sensor value is higher than high threshold

enumerator **TEMPERATURE_VAL_LOWER_THAN_LOW_THRESHOLD**

temperature sensor value is lower than low threshold

Header File

- components/hal/include/hal/temperature_sensor_types.h
- This header file can be included with:

```
#include "hal/temperature_sensor_types.h"
```

Type Definitions

typedef *soc_periph_temperature_sensor_clk_src_t* **temperature_sensor_clk_src_t**

temperature sensor clock source

Enumerations

enum **temperature_sensor_etm_event_type_t**

temperature sensor event types enum

Values:

enumerator **TEMPERATURE_SENSOR_EVENT_OVER_LIMIT**

Temperature sensor over limit event

enumerator **TEMPERATURE_SENSOR_EVENT_MAX**

Maximum number of temperature sensor events

enum **temperature_sensor_etm_task_type_t**

temperature sensor task types enum

Values:

enumerator **TEMPERATURE_SENSOR_TASK_START**

Temperature sensor start task

enumerator **TEMPERATURE_SENSOR_TASK_STOP**

Temperature sensor stop task

enumerator **TEMPERATURE_SENSOR_TASK_MAX**

Maximum number of temperature sensor tasks

Header File

- components/esp_driver_tsens/include/driver/temperature_sensor_etm.h
- This header file can be included with:

```
#include "driver/temperature_sensor_etm.h"
```

- This header file is a part of the API provided by the `esp_driver_tsens` component. To declare that your component depends on `esp_driver_tsens`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_tsens
```

or

```
PRIV_REQUIRES esp_driver_tsens
```

Functions

esp_err_t **temperature_sensor_new_etm_event**(*temperature_sensor_handle_t* tsens, const *temperature_sensor_etm_event_config_t* *config, *esp_etm_event_handle_t* *out_event)

Get the ETM event for Temperature Sensor.

备注: The created ETM event object can be deleted later by calling `esp_etm_del_event`

参数

- **tsens** -- **[in]** Temperature Sensor handle, allocated by `temperature_sensor_install()`
- **config** -- **[in]** Temperature Sensor ETM event configuration
- **out_event** -- **[out]** Returned ETM event handle

返回

- ESP_OK: Get ETM event successfully
- ESP_ERR_INVALID_ARG: Get ETM event failed because of invalid argument
- ESP_FAIL: Get ETM event failed because of other error

esp_err_t **temperature_sensor_new_etm_task**(*temperature_sensor_handle_t* tsens, const *temperature_sensor_etm_task_config_t* *config, *esp_etm_task_handle_t* *out_task)

Get the ETM task for Temperature Sensor.

备注: The created ETM task object can be deleted later by calling `esp_etm_del_task`

参数

- **tsens** -- **[in]** Temperature Sensor, allocated by `temperature_sensor_install()`
- **config** -- **[in]** Temperature Sensor ETM task configuration
- **out_task** -- **[out]** Returned ETM task handle

返回

- ESP_OK: Get ETM task successfully
- ESP_ERR_INVALID_ARG: Get ETM task failed because of invalid argument
- ESP_FAIL: Get ETM task failed because of other error

Structures

struct **temperature_sensor_etm_event_config_t**

Temperature Sensor ETM event configuration.

Public Members

temperature_sensor_etm_event_type_t **event_type**

Temperature Sensor ETM event type

struct **temperature_sensor_etm_task_config_t**

Temperature Sensor ETM task configuration.

Public Members

temperature_sensor_etm_task_type_t **task_type**

Temperature Sensor ETM task type

2.5.34 电容式触摸传感器

概述

触摸传感器系统由保护覆盖层、触摸电极、绝缘基板和走线组成，保护覆盖层位于最上层，绝缘基板上设有电极及走线。触摸覆盖层将引起电容变化，根据电容变化，可以判断此次触摸是否为有效触摸行为。

触摸传感器可以以矩阵或滑条等方式组合使用，从而覆盖更大触感区域及更多触感点。触摸传感由软件或专用硬件计时器发起，由有限状态机 (FSM) 硬件控制。

如需了解触摸传感器设计、操作及其控制寄存器等相关信息，请参考《ESP32-P4 技术参考手册》(PDF) 中“片上传感器与模拟信号处理”章节。

请参考 触摸传感器应用方案简介，查看触摸传感器设计详情和固件开发指南。

电容式触摸传感器版本概览

硬件版本	芯片	主要特征
V1	ESP32	第一代触摸传感器，触摸时读数变小
V2	ESP32-S2	第二代触摸传感器，触摸时读数变大新增防水防潮、接近感应、睡眠唤醒功能
	ESP32-S3	第二代触摸传感器，新增接近感应测量完成中断
V3	ESP32-P4	第三代触摸传感器，新增跳频扫描

触摸通道概览

通道	CH0	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8	CH9	CH10	CH11	CH12	CH13	CH14
GPIO	IO2	IO3	IO4	IO5	IO6	IO7	IO8	IO9	IO10	IO11	IO12	IO13	IO14	IO15	未引出

驱动中的术语介绍

- **触摸传感器控制器**：触摸传感器驱动的控制器，负责触摸传感器的配置和管理。
- **触摸传感器通道**：具体的一路触摸传感器采样通道。一个触摸传感器模块具有多个通道，一般连接到触摸板上，用于测量该触摸板电容的变化。驱动中把对 一个通道的采样称为 测量，而对 **所有**注册通道的 测量称为一次 扫描。
- **触摸传感器采样配置**：触摸传感器采样配置是驱动中对采样有关的硬件配置的统称。采样配置负责触摸传感器通道的采样，其配置决定了触摸通道的充放电次数、充放电频率、测量间隔等。ESP32-P4 仅支持一套采样配置，不支持跳频采样。

文件结构

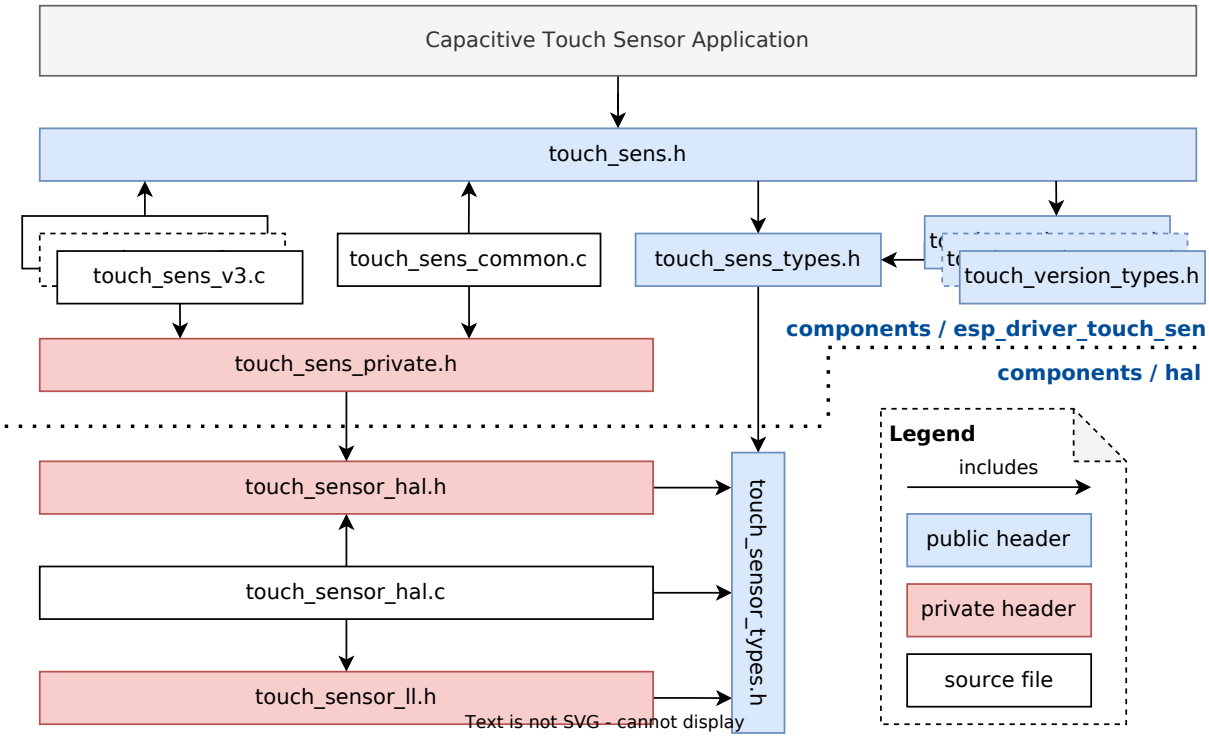


图 37: 触摸传感器驱动文件结构图

驱动状态机

下图为触摸传感器驱动的状态机，描述了调用不同函数后驱动的运行状态，以及状态变迁的约束。

上图为触摸传感器驱动的状态机，描述了调用不同函数后状态的变换关系。其中 `<other_configurations>` 部分为可选的配置项，包括对触摸驱动控制器和触摸通道的重新配置、回调函数注册等。

备注： *`touch_channel_read_data()`* 可在获取触摸通道句柄后（即 `INIT` 后）任意状态调用，但请注意读数值的有效性。

功能介绍

ESP32-P4 的电容式触摸传感器驱动提供的 API 按功能主要可分为：

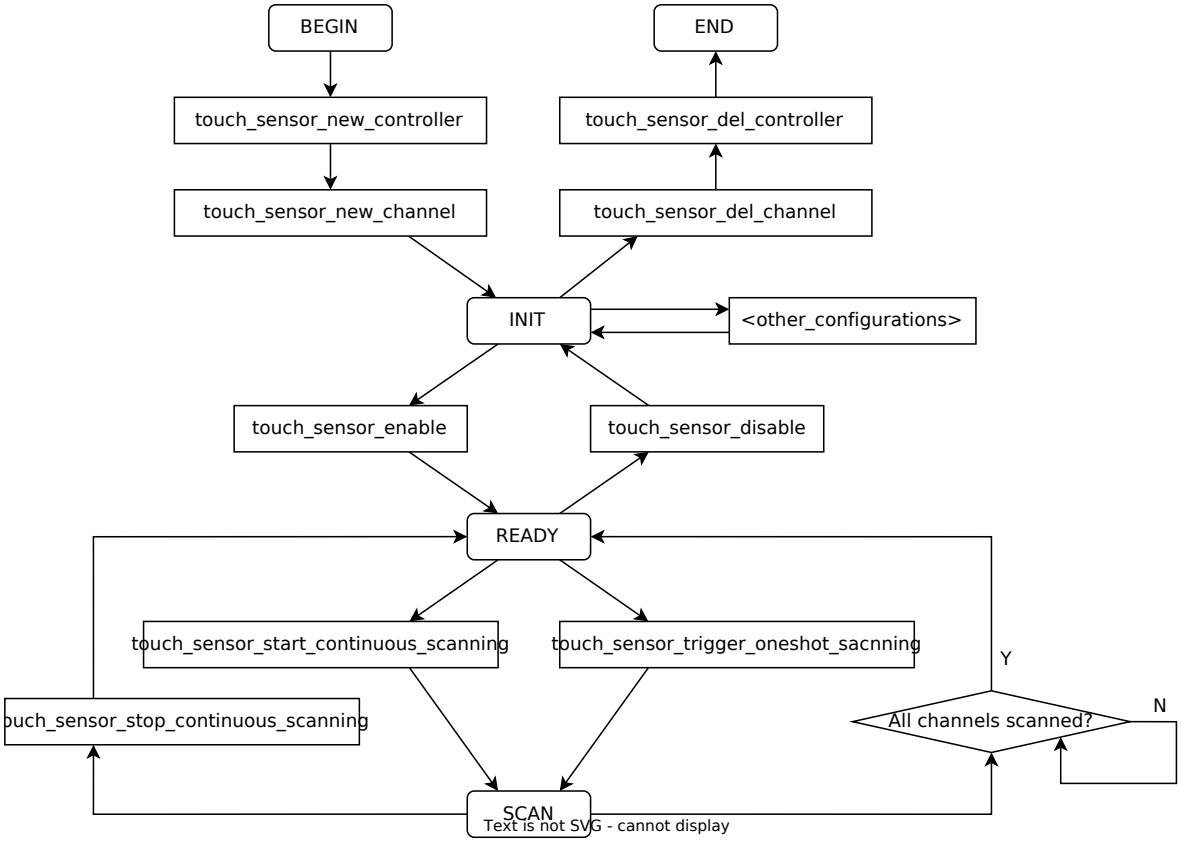


图 38: 触摸传感器驱动状态机示意图

- 触摸传感器控制器管理
- 触摸传感器通道管理
- 滤波器配置
- 回调函数
- 启用和禁用
- 连续扫描
- 单次扫描
- 基线值配置
- 测量值读数
- 防水防潮配置
- 接近感应配置
- 睡眠唤醒配置

触摸传感器控制器管理 触摸传感器驱动通过触摸传感器控制器句柄*touch_sensor_handle_t* 控制。调用*touch_sensor_new_controller()* 函数即可初始化触摸传感器控制器并得到控制器句柄。

```
// 有些芯片支持多套采样配置，这里以一套为例
#define SAMPLE_NUM 1
touch_sensor_handle_t sens_handle = NULL;
// 采样配置
touch_sensor_sample_config_t sample_cfg[SAMPLE_NUM] = {
    // 指定采样配置或通过 `TOUCH_SENSOR_Vn_DEFAULT_SAMPLE_CONFIG` 使用默认采样配置
    // ...
};
// 默认控制器配置
touch_sensor_config_t touch_cfg = TOUCH_SENSOR_DEFAULT_BASIC_CONFIG(SAMPLE_NUM,␣
↪sample_cfg);
// 申请一个新的触摸传感器控制器句柄
```

(下页继续)

(续上页)

```
ESP_ERROR_CHECK(touch_sensor_new_controller(&touch_cfg, &sens_handle));
```

删除触摸传感器驱动控制器时需调用*touch_sensor_del_controller()* 函数，从而释放该控制器所占用的软硬件资源。注意，需要将基于该控制器申请的其他资源销毁或释放后才能删除该控制器。如该控制器下仍有触摸通道未被删除，则无法直接删除。

```
ESP_ERROR_CHECK(touch_sensor_del_controller(sens_handle));
```

在触摸传感器驱动控制器初始化后，且未启用触摸传感器时，可调用*touch_sensor_reconfig_controller()* 函数对该控制器进行重新配置。

```
touch_sensor_config_t touch_cfg = {
    // 控制器的新配置
    // ...
};
ESP_ERROR_CHECK(touch_sensor_reconfig_controller(sens_handle, &touch_cfg));
```

触摸传感器通道管理 一个触摸传感器具有多个测量通道，每个触摸传感器通道由句柄*touch_channel_handle_t* 控制。调用*touch_sensor_new_channel()* 函数即可初始化触摸传感器通道并得到通道句柄。

```
// ...
touch_channel_config_t chan_cfg = {
    // 触摸通道配置
    // ...
};
touch_channel_handle_t chan_handle = NULL;
int chan_id = 0;
// 申请一个新的触摸通道句柄
ESP_ERROR_CHECK(touch_sensor_new_channel(sens_handle, chan_id, &chan_cfg, &chan_
↪handle));
```

删除触摸传感器通道时需调用*touch_sensor_del_channel()* 函数，从而释放该通道所占用的软硬件资源。

```
ESP_ERROR_CHECK(touch_sensor_del_channel(chan_handle));
```

在触摸传感器驱动通道初始化后，且未启用触摸传感器时，可调用*touch_sensor_reconfig_channel()* 函数对该通道进行重新配置。

```
touch_channel_config_t chan_cfg = {
    // 触摸通道新配置
    // ...
};
ESP_ERROR_CHECK(touch_sensor_reconfig_channel(chan_handle, &chan_cfg));
```

滤波器配置 触摸传感器可以通过配置滤波器来提升不同场景下的数据稳定性。调用*touch_sensor_config_filter()* 并指定*touch_sensor_filter_config_t* 来配置基线值和读数值的滤波策略和更新方式，配置后对所有启用的触摸通道都生效。

若需要注销滤波器，可再次调用*touch_sensor_config_filter()* 并将第二个参数（即*touch_sensor_filter_config_t* 的配置结构体指针）设为 NULL 来注销滤波器功能。

```
// ...
touch_sensor_filter_config_t filter_config = {
    // 滤波器配置
    // ...
};
```

(下页继续)

(续上页)

```
// 注册滤波器
ESP_ERROR_CHECK(touch_sensor_config_filter(sens_handle, &filter_config));
// ...
// 注销滤波器
ESP_ERROR_CHECK(touch_sensor_config_filter(sens_handle, NULL));
```

回调函数 通过调用*touch_sensor_register_callbacks()* 注册各类触摸传感器事件回调函数，当触摸传感器通道触发如触摸 on_active、释放 on_inactive 等事件时，就会调用对应的回调函数通知上层应用，以便对触摸事件进行处理。

例如，测量值超出当前的测量通道的 基线值 + 触发阈值，则该通道将被触发，并调用 on_active 事件的回调函数，通知应用层该触摸通道被 触发。同理，若处于 触发状态的通道测量值小于 基线值 + 触发阈值，则该通道将回到未触发状态，并调用 on_inactive 事件的回调函数，通知应用层该触摸通道被释放。

备注: 为保证触发和释放事件的稳定性，触摸传感器可配置 触发阈值的迟滞比较裕量和 去抖动计数来避免短时间内由噪声和读数抖动引起的反复触发和释放

具体可注册的回调时间请参考*touch_event_callbacks_t*。

```
touch_event_callbacks_t callbacks = {
    .on_active = example_touch_on_active_cb,
    // 其他回调函数
    // ...
};
// 注册回调函数
ESP_ERROR_CHECK(touch_sensor_register_callbacks(sens_handle, &callbacks, NULL));

// 通过把相应回调设为 NULL 以注销回调函数
callbacks.on_active = NULL;
// 其他需要注销的回调函数
// ...
ESP_ERROR_CHECK(touch_sensor_register_callbacks(sens_handle, &callbacks, NULL));
```

启用和禁用 配置完成触摸传感器控制器以及通道后，可调用*touch_sensor_enable()* 函数启用该控制器，启用后控制器处于 就绪状态，会对注册的通道上电，可以开始扫描并采集触摸数据。注意，控制器启用后无法更新配置，只能进行扫描采样和读数操作。若要更新配置，需先调用*touch_sensor_disable()* 函数禁用控制器，方可重新配置控制器、通道等。

```
// 启用触摸传感器
ESP_ERROR_CHECK(touch_sensor_enable(sens_handle));
// ...
// 禁用触摸传感器
ESP_ERROR_CHECK(touch_sensor_disable(sens_handle));
```

连续扫描 在控制器启用后，调用*touch_sensor_start_continuous_scanning()* 函数可开始对所有已注册的触摸通道进行连续扫描，每次扫描都会更新对应通道的测量值。调用*touch_sensor_stop_continuous_scanning()* 函数后则停止扫描。

```
// 开始连续扫描
ESP_ERROR_CHECK(touch_sensor_start_continuous_scanning(sens_handle));
// ...
// 停止连续扫描
ESP_ERROR_CHECK(touch_sensor_stop_continuous_scanning(sens_handle));
```

单次扫描 在控制器启用后，调用*touch_sensor_trigger_oneshot_scanning()* 函数可触发一次对所有已注册的触摸通道的扫描。注意，单次扫描为阻塞函数，调用后会保持阻塞直到扫描结束后返回。此外在开始连续扫描后，无法再触发单次扫描。

```
// 触发单次扫描，并设置超时时间为 1000ms
ESP_ERROR_CHECK(touch_sensor_trigger_oneshot_scanning(sens_handle, 1000));
```

基线值配置 一般情况下，不需要额外设置触摸传感器的基线值，若有必要强制复位基线值到当前平滑值，可调用*touch_channel_config_benchmark()*。

```
touch_chan_benchmark_config_t benchmark_cfg = {
    // 基线操作
    // ...
};
ESP_ERROR_CHECK(touch_channel_config_benchmark(chan_handle, &benchmark_cfg));
```

测量值读数 调用*touch_channel_read_data()* 可读取每个通道不同种类的数据，例如基线值、经过滤波后的平滑值等。支持的数据类型请参考*touch_chan_data_type_t*。

ESP32-P4 支持通过配置多套采样配置来实现跳频采样，*touch_channel_read_data()* 可一次性读出一个通道所有采样配置的测量值。根据配置的*touch_sensor_config_t::sample_cfg_num* 采样配置数量，第三个参数 (*data) 数据指针传入数组长度大于等于采样配置数量的数组指针即可，该函数会将所指定通道的所有采样配置的测量值存入该数组中。

```
#define SAMPLE_NUM  1  // 以一套采样配置为例
uint32_t smooth_data[SAMPLE_NUM] = {};
// 读取滤波后的平滑数据
ESP_ERROR_CHECK(touch_channel_read_data(chan_handle, TOUCH_CHAN_DATA_TYPE_SMOOTH,␣
↪smooth_data));
```

防水防潮配置 ESP32-P4 支持防水防潮功能。可通过调用*touch_sensor_config_waterproof()* 并配置*touch_waterproof_config_t* 来注册防水防潮功能。防水防潮功能主要包含两部分：

- 遇水（浸没）保护功能：*touch_waterproof_config_t::guard_chan* 用于指定用于遇水保护功能的触摸通道，该通道在 PCB 上一般设计成环形，其他普通触摸通道布局在该环形保护圈内，当电路板大面积浸水时，该环形保护通道会被触发，并停止其他触摸通道的扫描，由此防止其他普通通道的误触发；
- 防潮（水滴）屏蔽功能：*touch_waterproof_config_t::shield_chan* 用于指定防潮屏蔽功能的触摸通道，该通道在 PCB 上一般设计成网格状铺铜。防潮屏蔽通道将与当前测量通道进行同步充放电，当有小水珠覆盖时，通过配置适当的*touch_waterproof_config_t::shield_drv* 来提高因水滴造成的电耦合强度，从而识别水滴造成的误触。在实际应用中识别到水滴造成的误触后可适当增加触摸通道触发的阈值来实现通道在水滴覆盖下的正常触发，从而实现防潮功能。

若需要注销防水防潮功能，可再次调用*touch_sensor_config_waterproof()* 并将第二个参数（即*touch_waterproof_config_t* 的配置结构体指针）设为 NULL 来注销防水防潮功能。

```
touch_waterproof_config_t waterproof_cfg = {
    // 防水防潮配置
    // ...
};
// 注册防水防潮功能
ESP_ERROR_CHECK(touch_sensor_config_waterproof(sens_handle, &waterproof_cfg));
// ...
// 注销防水防潮功能
ESP_ERROR_CHECK(touch_sensor_config_waterproof(sens_handle, NULL));
```

接近感应配置 ESP32-P4 支持接近感应功能。可通过调用*touch_sensor_config_proximity_sensing()* 并配置*touch_proximity_config_t* 来注册接近感应功能。

由于接近感应引起的电容变化远小于物理触摸，PCB 上常用较大面积的铺铜来增大触摸通道的感应面积，另外需要在硬件上对接近感应通道进行多轮扫描并在驱动中进行累加来提高测量灵敏度。接近感应的灵敏度由测量轮数*touch_proximity_config_t::scan_times* 以及单次测量的充放电次数*touch_proximity_config_t::charge_times* 决定。测量轮数以及充放电次数越高，灵敏度越高，但是过高的灵敏度容易导致误触发，请选择适当的灵敏度来保证触发的稳定性。

接近感应通道多次测量的累加值也可通过*touch_channel_read_data()* 获取，数据类型*touch_chan_data_type_t* 为*TOUCH_CHAN_DATA_TYPE_PROXIMITY*。

若需要注销接近感应功能，可再次调用*touch_sensor_config_proximity_sensing()* 并将第二个参数（即*touch_proximity_config_t* 的配置结构体指针）设为 NULL 来注销接近感应功能。

```
touch_proximity_config_t prox_cfg = {
    // 接近感应配置
    // ...
};
// 注册接近感应功能
ESP_ERROR_CHECK(touch_sensor_config_proximity_sensing(sens_handle, &prox_cfg));
// ...
// 注销接近感应功能
ESP_ERROR_CHECK(touch_sensor_config_proximity_sensing(sens_handle, NULL));
```

睡眠唤醒配置 ESP32-P4 支持触摸传感器将芯片从浅睡眠或深睡眠状态中唤醒。可通过调用*touch_sensor_config_sleep_wakeup()* 并配置*touch_sleep_config_t* 来注册接近感应功能。

注册触摸传感器的睡眠唤醒功能后，处于睡眠状态下的芯片仍将继续保持对触摸传感器的采样，这将会导致芯片睡眠后的功耗增加，可通过减少充放电次数、增加采样间隔等方式来降低功耗。

另外，请注意在主核睡眠期间的采样、唤醒等操作均由硬件完成，本驱动由于运行在主核上，无法提供读数、配置等功能。

- 浅睡眠状态唤醒：通过指定 slp_wakeup_lvl 为*TOUCH_LIGHT_SLEEP_WAKEUP* 即可启用触摸传感器浅睡眠唤醒功能。注意任何已注册的触摸传感器通道都会在浅睡眠状态下保持采样并支持唤醒浅睡眠。
- 深睡眠状态唤醒：启用触摸传感器深睡眠唤醒功能除了指定 slp_wakeup_lvl 为*TOUCH_DEEP_SLEEP_WAKEUP* 外，还需要指定深睡眠唤醒通道 deep_slp_chan，注意只有该指定的通道才会在深睡眠状态下保持采样以及唤醒，以此降低在深睡眠状态下的功耗。此外，若需要在深度睡眠下使用另一套低功耗的配置来进一步降低功耗，可以通过 deep_slp_sens_cfg 额外指定一套低功耗配置，在进入深睡眠前，驱动会应用这套配置，从深睡眠状态唤醒后，则会重新配置到之前的配置。请注意当 slp_wakeup_lvl 配置为*TOUCH_DEEP_SLEEP_WAKEUP* 后，触摸传感器不仅能唤醒深睡眠状态，还能唤醒浅睡眠状态。

若需要注销睡眠唤醒功能，可再次调用*touch_sensor_config_sleep_wakeup()* 并将第二个参数（即*touch_sleep_config_t* 的配置结构体指针）设为 NULL 来注销睡眠唤醒功能。

```
touch_sleep_config_t light_slp_cfg = {
    .slp_wakeup_lvl = TOUCH_LIGHT_SLEEP_WAKEUP,
};
// 注册浅睡眠唤醒功能
ESP_ERROR_CHECK(touch_sensor_config_sleep_wakeup(sens_handle, &light_slp_cfg));
// ...
// 注销睡眠唤醒功能
ESP_ERROR_CHECK(touch_sensor_config_sleep_wakeup(sens_handle, NULL));
touch_sleep_config_t deep_slp_cfg = {
    .slp_wakeup_lvl = TOUCH_DEEP_SLEEP_WAKEUP,
    .deep_slp_chan = dslp_chan_handle,
    // 其他深睡眠唤醒配置
    // ...
```

(下页继续)

(续上页)

```
};
// 注册深睡眠唤醒功能
ESP_ERROR_CHECK(touch_sensor_config_sleep_wakeup(sens_handle, &deep_slp_cfg));
```

应用示例

- 触摸传感器基础例程: peripherals/touch_sensor/touch_sensor_v3.

API 参考

Header File

- components/esp_driver_touch_sens/include/driver/touch_sens.h
- This header file can be included with:

```
#include "driver/touch_sens.h"
```

- This header file is a part of the API provided by the `esp_driver_touch_sens` component. To declare that your component depends on `esp_driver_touch_sens`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_touch_sens
```

or

```
PRIV_REQUIRES esp_driver_touch_sens
```

Functions

esp_err_t **touch_sensor_new_controller**(const *touch_sensor_config_t* *sens_cfg, *touch_sensor_handle_t* *ret_sens_handle)

Allocate a new touch sensor controller.

备注: The touch sensor driver will be in INIT state after this function is called successfully.

参数

- **sens_cfg** -- **[in]** Touch sensor controller configurations
- **ret_sens_handle** -- **[out]** The return handle of the touch controller instance

返回

- ESP_OK On success
- ESP_ERR_NO_MEM No memory for the touch sensor controller
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The touch sensor controller is already allocated

esp_err_t **touch_sensor_del_controller**(*touch_sensor_handle_t* sens_handle)

Delete the touch sensor controller.

备注: This function can be called when the touch sensor controller is NOT enabled (i.e. INIT state).

参数 **sens_handle** -- **[in]** Touch sensor controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE Controller not disabled or still some touch channels not deleted

esp_err_t **touch_sensor_new_channel**(*touch_sensor_handle_t* sens_handle, int chan_id, const *touch_channel_config_t* *chan_cfg, *touch_channel_handle_t* *ret_chan_handle)

Allocate a new touch channel from the touch sensor controller.

备注: This function can be called when the touch sensor controller is NOT enabled (i.e. INIT state).

参数

- **sens_handle** -- **[in]** Touch sensor controller handle
- **chan_id** -- **[in]** Touch channel index
- **chan_cfg** -- **[in]** Touch channel configurations
- **ret_chan_handle** -- **[out]** The return handle of the touch channel instance

返回

- ESP_OK On success
- ESP_ERR_NO_MEM No memory for the touch sensor channel
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The touch sensor controller has not disabled or this channel has been allocated

esp_err_t **touch_sensor_del_channel**(*touch_channel_handle_t* chan_handle)

Delete the touch channel.

备注: This function can be called when the touch sensor controller is NOT enabled (i.e. INIT state).

参数 **chan_handle** -- **[in]** Touch channel handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The touch sensor controller has not disabled

esp_err_t **touch_sensor_reconfig_controller**(*touch_sensor_handle_t* sens_handle, const *touch_sensor_config_t* *sens_cfg)

Re-configure the touch sensor controller.

备注: This function can be called when the touch sensor controller is NOT enabled (i.e. INIT state).

备注: The re-configuration only applies to the touch controller, so it requires the controller handle that allocated from `touch_sensor_new_controller`, meanwhile, it won't affect the touch channels, no matter the channels are allocated or not.

参数

- **sens_handle** -- **[in]** Touch sensor controller handle
- **sens_cfg** -- **[in]** Touch sensor controller configurations

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The touch sensor controller has not disabled

esp_err_t **touch_sensor_reconfig_channel**(*touch_channel_handle_t* chan_handle, const *touch_channel_config_t* *chan_cfg)

Re-configure the touch channel.

备注： This function can be called when the touch sensor controller is NOT enabled (i.e. INIT state).

备注： The re-configuration only applies to a particular touch channel, so it requires the channel handle that allocated from `touch_sensor_new_channel`

参数
- **`chan_handle`** -- **[in]** Touch channel handle
- **`chan_cfg`** -- **[in]** Touch channel configurations

返回
- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The touch sensor controller has not disabled

esp_err_t **`touch_sensor_config_filter`**(*touch_sensor_handle_t* sens_handle, const *touch_sensor_filter_config_t* *filter_cfg)

Configure the touch sensor filter.

备注： This function is allowed to be called after the touch sensor is enabled

参数
- **`sens_handle`** -- **[in]** Touch sensor controller handle
- **`filter_cfg`** -- **[in]** Filter configurations, set NULL to disable the filter

返回
- ESP_OK: Configure the filter success
- ESP_ERR_INVALID_ARG: The sensor handle is NULL

esp_err_t **`touch_sensor_enable`**(*touch_sensor_handle_t* sens_handle)

Enable the touch sensor controller.

备注： This function can be called when the touch sensor controller is NOT enabled (i.e. INIT state). And the touch sensor driver will be in ENABLED state after this function is called successfully.

备注： Enable the touch sensor will power on the registered touch channels

参数 **`sens_handle`** -- **[in]** Touch sensor controller handle

返回
- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The touch sensor controller has already enabled

esp_err_t **`touch_sensor_disable`**(*touch_sensor_handle_t* sens_handle)

Disable the touch sensor controller.

备注： This function can only be called after the touch sensor controller is enabled (i.e. ENABLED state). And the touch sensor driver will be in INIT state after this function is called successfully.

备注： Disable the touch sensor will power off the registered touch channels

参数 **sens_handle** -- **[in]** Touch sensor controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The touch sensor controller has already disabled

esp_err_t **touch_sensor_start_continuous_scanning**(*touch_sensor_handle_t* sens_handle)

Start the scanning of the registered touch channels continuously.

备注： This function can only be called after the touch sensor controller is enabled (i.e. ENABLED state). And the touch sensor driver will be in SCANNING state after this function is called successfully. And it can also be called in ISR/callback context.

备注： The hardware FSM (Finite-State Machine) of touch sensor will be driven by its hardware timer continuously and repeatedly. i.e., the registered channels will be scanned one by one repeatedly.

参数 **sens_handle** -- **[in]** Touch sensor controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The touch sensor controller is not enabled or the continuous scanning has started

esp_err_t **touch_sensor_stop_continuous_scanning**(*touch_sensor_handle_t* sens_handle)

Stop the continuous scanning of the registered touch channels immediately.

备注： This function can only be called after the continuous scanning started (i.e. SCANNING state). And the touch sensor driver will be in ENABLED state after this function is called successfully. And it can also be called in ISR/callback context.

备注： Stop the hardware timer and reset the FSM (Finite-State Machine) of the touch sensor controller

参数 **sens_handle** -- **[in]** Touch sensor controller handle

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer
- ESP_ERR_INVALID_STATE The continuous scanning has stopped

esp_err_t **touch_sensor_trigger_oneshot_scanning**(*touch_sensor_handle_t* sens_handle, int timeout_ms)

Trigger an oneshot scanning for all the registered channels.

备注： This function can only be called after the touch sensor controller is enabled (i.e. ENABLED state). And the touch sensor driver will be in SCANNING state after this function is called successfully, and then switch back to ENABLED state after the scanning is done or timeout.

备注： The block time of this function depends on various factors, In common practice, recommend to set the timeout to several seconds or wait forever, because oneshot scanning can't last for so long.

参数

- **`sens_handle`** -- **[in]** Touch sensor controller handle
- **`timeout_ms`** -- **[in]** Set a positive value or zero means scan timeout in milliseconds Set a negative value means wait forever until finished scanning

返回

- ESP_OK On success
- ESP_ERR_TIMEOUT Timeout to finish the oneshot scanning
- ESP_ERR_INVALID_ARG Invalid argument
- ESP_ERR_INVALID_STATE The touch sensor controller is not enabled or the continuous scanning has started

esp_err_t **`touch_sensor_register_callbacks`**(*touch_sensor_handle_t* sens_handle, const *touch_event_callbacks_t* *callbacks, void *user_ctx)

Register the touch sensor callbacks.

备注： This function can be called when the touch sensor controller is NOT enabled (i.e. INIT state).

参数

- **`sens_handle`** -- **[in]** Touch sensor controller handle
- **`callbacks`** -- **[in]** Callback functions
- **`user_ctx`** -- **[in]** User context that will be passed to the callback functions

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG NULL pointer
- ESP_ERR_INVALID_STATE Touch sensor controller has not disabled

esp_err_t **`touch_channel_config_benchmark`**(*touch_channel_handle_t* chan_handle, const *touch_chan_benchmark_config_t* *benchmark_cfg)

Confiture the touch sensor benchmark for all the registered channels.

备注： This function can be called no matter the touch sensor controller is enabled or not (i.e. ENABLED or SCANNING state). And it can also be called in ISR/callback context.

参数

- **`chan_handle`** -- **[in]** Touch channel handle
- **`benchmark_cfg`** -- **[in]** The benchmark configurations

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG NULL pointer

esp_err_t **`touch_channel_read_data`**(*touch_channel_handle_t* chan_handle, *touch_chan_data_type_t* type, uint32_t *data)

Read the touch channel data according to the types.

备注： This function can be called no matter the touch sensor controller is enabled or not (i.e. ENABLED or SCANNING state). And it can also be called in ISR/callback context.

备注: Specially, `TOUCH_CHAN_DATA_TYPE_PROXIMITY` data type will be read from the cached data in the driver, because the proximity data need to be accumulated in several scan times that specified by *touch_proximity_config_t::scan_times*. For other data types, the data are read from the hardware registers directly (not cached in the driver). The channel data in the register will be updated once the measurement of this channels is done, And keep locked until the next measurement is done.

参数

- **chan_handle** -- **[in]** Touch channel handle
- **type** -- **[in]** Specify the data type to read
- **data** -- **[out]** The data array pointer. If the target supports multiple sample configurations (SOC_TOUCH_SAMPLE_CFG_NUM), the array length should be equal to the frequency hopping number that specified in *touch_sensor_config_t::sample_cfg_num*, otherwise the array length should be 1.

返回

- ESP_OK On success
- ESP_ERR_INVALID_ARG Invalid argument or NULL pointer

esp_err_t **touch_sensor_config_waterproof**(*touch_sensor_handle_t* sens_handle, const *touch_waterproof_config_t* *wp_cfg)

Configure the touch sensor water proof channels.

备注: Once waterproof is enabled, the other touch channels won't be updated unless the shield channels is activated.

参数

- **sens_handle** -- **[in]** Touch sensor controller handle
- **wp_cfg** -- **[in]** Water proof channel configurations, set NULL to disable the water proof function

返回

- ESP_OK: Configure the water proof success
- ESP_ERR_INVALID_ARG: The sensor handle is NULL
- ESP_ERR_INVALID_STATE: The touch sensor is enabled

esp_err_t **touch_sensor_config_proximity_sensing**(*touch_sensor_handle_t* sens_handle, const *touch_proximity_config_t* *prox_cfg)

Configure the touch sensor proximity sensing channels.

参数

- **sens_handle** -- **[in]** Touch sensor controller handle
- **prox_cfg** -- **[in]** Proximity channels configurations, set NULL to disable the proximity sensing

返回

- ESP_OK: Configure the proximity channel success
- ESP_ERR_INVALID_ARG: The sensor handle is NULL
- ESP_ERR_INVALID_STATE: The touch sensor is enabled

esp_err_t **touch_sensor_config_sleep_wakeup**(*touch_sensor_handle_t* sens_handle, const *touch_sleep_config_t* *sleep_cfg)

Configure the touch sensor to wake-up the chip from sleep.

备 注: Call this function to enable/disable the touch sensor wake-up the chip from deep/light sleep To only enable the touch sensor wake-up the chip from light sleep, set the

touch_sleep_config_t::deep_slp_chan to NULL. During the light sleep, all enabled touch channels will keep working, and any one of them can wake-up the chip from light sleep. To enable the touch sensor wake-up the chip from both light and deep sleep, set the *touch_sleep_config_t::deep_slp_chan* to an enabled channel. During the deep sleep, only this specified channels can work and wake-up the chip from the deep sleep, and other enabled channels can only wake-up the chip from light sleep.

参数

- **sens_handle** -- **[in]** Touch sensor controller handle
- **sleep_cfg** -- **[in]** Sleep wake-up configurations, set NULL to disable the touch sensor wake-up the chip from sleep

返回

- ESP_OK: Configure the sleep channel success
- ESP_ERR_INVALID_ARG: The sensor handle is NULL
- ESP_ERR_INVALID_STATE: The touch sensor is enabled

Header File

- components/esp_driver_touch_sens/include/driver/touch_sens_types.h
- This header file can be included with:

```
#include "driver/touch_sens_types.h"
```

- This header file is a part of the API provided by the `esp_driver_touch_sens` component. To declare that your component depends on `esp_driver_touch_sens`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_touch_sens
```

or

```
PRIV_REQUIRES esp_driver_touch_sens
```

Macros

TOUCH_TOTAL_CHAN_NUM

The total channel number of the touch sensor

TOUCH_SAMPLE_CFG_NUM

The supported max sample configuration number

TOUCH_PROXIMITY_CHAN_NUM

The supported proximity channel number in proximity sensing mode

Type Definitions

typedef struct touch_sensor_s ***touch_sensor_handle_t**

The handle of touch sensor controller

typedef struct touch_channel_s ***touch_channel_handle_t**

The handle of touch channel

Header File

- components/esp_driver_touch_sens/hw_ver3/include/driver/touch_version_types.h
- This header file can be included with:

```
#include "driver/touch_version_types.h"
```

- This header file is a part of the API provided by the `esp_driver_touch_sens` component. To declare that your component depends on `esp_driver_touch_sens`, add the following to your CMakeLists.txt:

  ```
  REQUIRES esp_driver_touch_sens
  ```

  or

  ```
  PRIV_REQUIRES esp_driver_touch_sens
  ```

Structures

struct **touch_sensor_sample_config_t**

Sample configurations of the touch sensor.

Public Members

uint32_t **div_num**

Division of the touch output pulse, `touch_out_pulse / div_num = charge_times`

uint32_t **charge_times**

The charge and discharge times of this sample configuration, the read data are positive correlation to the charge_times

uint8_t **rc_filter_res**

The resistance of the RC filter of this sample configuration, range [0, 3], while 0 = 0K, 1 = 1.5K, 2 = 3K, 3 = 4.5K

uint8_t **rc_filter_cap**

The capacitance of the RC filter of this sample configuration, range [0, 127], while 0 = 0pF, 1 = 20fF, ..., 127 = 2.54pF

uint8_t **low_drv**

Low speed touch driver, only effective when high speed driver is disabled

uint8_t **high_drv**

High speed touch driver

uint8_t **bias_volt**

The Internal LDO voltage, which decide the bias voltage of the sample wave, range [0,15]

bool **bypass_shield_output**

Whether to bypass the shield output, enable when the charging/discharging rate greater than 10MHz

struct **touch_sensor_config_t**

Configurations of the touch sensor controller.

Public Members

uint32_t **power_on_wait_us**

The waiting time between the channels power on and able to measure, to ensure the data stability

float **meas_interval_us**

Measurement interval of each channels

uint32_t **max_meas_time_us**

The maximum time of measuring one channel, if the time exceeds this value, the timeout interrupt will be triggered. Set to '0' to ignore the measurement time limitation, otherwise please set a proper time considering the configurations of this sample configurations below.

touch_out_mode_t **output_mode**

Touch channel counting mode of the binarized touch output

uint32_t **sample_cfg_num**

The sample configuration number that used for sampling

touch_sensor_sample_config_t ***sample_cfg**

The array of this sample configuration configurations, the length should be specified in `touch_sensor_config_t::sample_cfg_num`

struct **touch_channel_config_t**

Configurations of the touch sensor channel.

Public Members

uint32_t **active_thresh**[TOUCH_SAMPLE_CFG_NUM]

The active threshold of each sample configuration, while the touch channel smooth value minus benchmark value exceed this threshold, will be regarded as activated

struct **touch_sensor_filter_config_t**

Configurations of the touch sensor filter.

Public Members

touch_benchmark_filter_mode_t **filter_mode**

Benchmark filter mode. IIR filter and Jitter filter can be selected, TOUCH_BM_IIR_FILTER_16 is recommended

uint32_t **jitter_step**

Jitter filter step size, only takes effect when the `filter_mode` is TOUCH_BM_JITTER_FILTER. Range: [0 ~ 15]

int **denoise_lvl**

The denoise level, which determines the noise bouncing range that won't trigger benchmark update. Range: [0 ~ 4]. The greater the denoise_lvl is, more noise resistance will be. Specially, `0` stands for no denoise Typically, recommend to set this field to 1.

struct *touch_sensor_filter_config_t*::[anonymous] **benchmark**

Benchmark configuration.

Benchmark filter

touch_smooth_filter_mode_t **smooth_filter**

Smooth data IIR filter mode

uint32_t **active_hysteresis**

The hysteresis threshold to judge whether the touch channel is active If the channel data exceed the '*touch_channel_config_t::active_thresh* + active_hysteresis' The channel will be activated. If the channel data is below to '*touch_channel_config_t::active_thresh* - active_hysteresis' the channel will be inactivated.

uint32_t **debounce_cnt**

The debounce count of the touch channel. Only when the channel data exceed the *touch_channel_config_t::active_thresh* + active_hysteresis for debounce_cnt times The channel will be activated. And only if the channel data is below to the *touch_channel_config_t::active_thresh* - active_hysteresis for debounce_cnt times, the channel will be inactivated. (The unit of debounce_cnt is the tick of the slow clock source)

struct *touch_sensor_filter_config_t*::[anonymous] **data**

Data configuration.

Channel data filter

struct **touch_sleep_config_t**

Configure the touch sensor sleep function.

Public Members

touch_sleep_wakeup_level_t **slp_wakeup_lvl**

The sleep level that can be woke up by touch sensor.

touch_channel_handle_t **deep_slp_chan**

The touch channel handle that supposed to work in the deep sleep. It can wake up the chip from deep sleep when this channel is activated. Only effective when the *touch_sleep_config_t::slp_wakeup_lvl* is TOUCH_DEEP_SLEEP_WAKEUP

uint32_t **deep_slp_thresh**[TOUCH_SAMPLE_CFG_NUM]

The active threshold of the deep sleep channel during deep sleep, while the sleep channel exceed this threshold, it will be regarded as activated Only effective when the *touch_sleep_config_t::slp_wakeup_lvl* is TOUCH_DEEP_SLEEP_WAKEUP

touch_sensor_config_dslp_t ***deep_slp_sens_cfg**

Specify the touch sensor configuration during the deep sleep. Note that these configurations will no take effect immediately, they will be set automatically while the chip prepare to enter sleep. Set NULL to not change the configurations before entering sleep. The sleep configuration mainly aims at lower down the charging and measuring times, so that to save power consumption during the sleep. Only effective when the *touch_sleep_config_t::slp_wakeup_lvl* is TOUCH_DEEP_SLEEP_WAKEUP

struct **touch_waterproof_config_t**

Configure the touch sensor waterproof function.

Public Members

touch_channel_handle_t **guard_chan**

The guard channel of that used for immersion detect. Set NULL if you don't need the guard channel. Typically, the guard channel is a ring that surrounds the touch panels, it is used to detect the large area that covered by water. While large area of water covers the touch panel, the guard channel will be activated.

touch_channel_handle_t **shield_chan**

The shield channel that used for water droplets shield, can't be NULL. Typically, the shield channel uses grid layout which covers the touch area, it is used to shield the influence of water droplets covering both the touch panel and the shield channel. The shield channel will be paralleled to the current measuring channel (except the guard channel) to reduce the influence of water droplets.

uint32_t **shield_drv**

The shield channel driver, which controls the drive capability of shield channel, range: 0 ~ 7 The larger the parasitic capacitance on the shielding channel, the higher the drive capability needs to be set.

uint32_t **immersion_proof**

Enable to protect the touch sensor pad when immersion detected. It will temporary disable the touch scanning if the guard channel triggered, and enable again if guard channel released. So that to avoid the fake touch when the touch panel is immersed in water.

struct *touch_waterproof_config_t*::[anonymous] **flags**

Flags of the water proof function

struct **touch_proximity_config_t**

Configure the touch sensor proximity function.

Public Members

touch_channel_handle_t **proximity_chan**[TOUCH_PROXIMITY_CHAN_NUM]

The touch channel handles that will be configured as proximity sensing channels

uint32_t **scan_times**

The total scan times of EACH sample configuration, all sample configurations share a same `scan_times`. The measurement result of each scanning will be accumulated together to get the final result.

uint32_t **charge_times**[TOUCH_SAMPLE_CFG_NUM]

The charge times of EACH scanning, different sample configurations can set different `charge_times`. The measurement result of each scanning is positive correlation to the `charge_times`, please set a proper value in case of the final accumulated result overflow.

struct **touch_base_event_data_t**

Base event structure used in touch event queue.

Public Members

touch_channel_handle_t **chan**

the current triggered touch channel handle

int **chan_id**

the current triggered touch channel number

uint32_t **status_mask**

the current channel triggered status. For the bits in the status mask, if the bit is set, the corresponding channel is active if the bit is cleared, the corresponding channel is inactive

struct **touch_event_callbacks_t**

Touch sensor callbacks.

备注： Set NULL for the used callbacks.

Public Members

bool (***on_active**)(*touch_sensor_handle_t* sens_handle, const *touch_active_event_data_t* *event, void *user_ctx)

Touch sensor on active event callback. Callback when any touch channel is activated.

Param sens_handle [in] Touch sensor controller handle, created from `touch_sensor_new_controller()`
Param event [in] Touch sensor active event data
Param user_ctx [in] User registered context, passed from `touch_sensor_register_callbacks()`
Return Whether a high priority task has been waken up by this callback function

bool (***on_inactive**)(*touch_sensor_handle_t* sens_handle, const *touch_inactive_event_data_t* *event, void *user_ctx)

Touch sensor on inactive event callback. Callback when any touch channel is inactivated.

Param sens_handle [in] Touch sensor controller handle, created from `touch_sensor_new_controller()`
Param event [in] Touch sensor inactive event data
Param user_ctx [in] User registered context, passed from `touch_sensor_register_callbacks()`
Return Whether a high priority task has been waken up by this callback function

bool (***on_measure_done**)(*touch_sensor_handle_t* sens_handle, const *touch_meas_done_event_data_t* *event, void *user_ctx)

Touch sensor on measure done event callback. Callback when the measurement of all the sample configurations on the current touch channel is done.

Param sens_handle [in] Touch sensor controller handle, created from `touch_sensor_new_controller()`
Param event [in] Touch sensor measure done event data
Param user_ctx [in] User registered context, passed from `touch_sensor_register_callbacks()`
Return Whether a high priority task has been waken up by this callback function

bool (*`on_scan_done`)(*touch_sensor_handle_t* sens_handle, const *touch_scan_done_event_data_t* *event, void *user_ctx)

Touch sensor on scan done event callback. Callback when finished scanning all the registered touch channels.

Param sens_handle [in] Touch sensor controller handle, created from `touch_sensor_new_controller()`
Param event [in] Touch sensor scan done event data
Param user_ctx [in] User registered context, passed from `touch_sensor_register_callbacks()`
Return Whether a high priority task has been waken up by this callback function

bool (*`on_timeout`)(*touch_sensor_handle_t* sens_handle, const *touch_timeout_event_data_t* *event, void *user_ctx)

Touch sensor on measurement timeout event callback. Callback when measure the current touch channel timeout.

Param sens_handle [in] Touch sensor controller handle, created from `touch_sensor_new_controller()`
Param event [in] Touch sensor timeout event data
Param user_ctx [in] User registered context, passed from `touch_sensor_register_callbacks()`
Return Whether a high priority task has been waken up by this callback function

bool (*`on_proximity_meas_done`)(*touch_sensor_handle_t* sens_handle, const *touch_prox_done_event_data_t* *event, void *user_ctx)

Touch sensor on proximity sensing measurement done event callback. Callback when proximity sensing measurement of the current channel is done.

Param sens_handle [in] Touch sensor controller handle, created from `touch_sensor_new_controller()`
Param event [in] Touch sensor proximity sensing measure done event data
Param user_ctx [in] User registered context, passed from `touch_sensor_register_callbacks()`
Return Whether a high priority task has been waken up by this callback function

struct **`touch_chan_benchmark_config_t`**

Touch sensor benchmark configurations, to set or reset the benchmark of the channel.

Public Members

bool **`do_reset`**

Whether to reset the benchmark to the channel's latest smooth data

Macros

`TOUCH_MIN_CHAN_ID`

This file is only applicable to the touch hardware version3 Version 3 includes ESP32-P4.

The minimum available channel id of the touch pad

`TOUCH_MAX_CHAN_ID`

The maximum available channel id of the touch pad

TOUCH_SENSOR_DEFAULT_BASIC_CONFIG(sample_cfg_number, sample_cfg_array)

Helper macro to the default configurations of the touch sensor controller.

TOUCH_SENSOR_V3_DEFAULT_SAMPLE_CONFIG(_div_num, coarse_freq_tune, fine_freq_tune)

Helper macro to the default sample configurations.

备注： This default configuration uses `sample frequency = clock frequency / 1`

TOUCH_SENSOR_DEFAULT_FILTER_CONFIG()

Type Definitions

typedef *touch_sensor_config_t* **touch_sensor_config_dslp_t**

Touch sensor configuration during the deep sleep.

备注： Currently it is the same as the normal controller configuration. The deep sleep configuration only takes effect when the chip entered sleep, so that to update a more power efficient configuration.

---

typedef *touch_base_event_data_t* **touch_meas_done_event_data_t**

Measure done event data.

备注： Currently same as base event data

---

typedef *touch_base_event_data_t* **touch_scan_done_event_data_t**

Scan done event data.

备注： Currently same as base event data

---

typedef *touch_base_event_data_t* **touch_active_event_data_t**

Active event data.

备注： Currently same as base event data

---

typedef *touch_base_event_data_t* **touch_inactive_event_data_t**

Inactive event data.

备注： Currently same as base event data

---

typedef *touch_base_event_data_t* **touch_prox_done_event_data_t**

Proximity sensing measure done event data.

备注： Currently same as base event data

---

typedef *touch_base_event_data_t* **touch_timeout_event_data_t**

Timeout event data.

备注： Currently same as base event data

Enumerations

enum **touch_chan_data_type_t**

The data type of the touch channel.

Values:

enumerator **TOUCH_CHAN_DATA_TYPE_SMOOTH**

The smooth data of the touch channel

enumerator **TOUCH_CHAN_DATA_TYPE_BENCHMARK**

The benchmark of the touch channel

enumerator **TOUCH_CHAN_DATA_TYPE_PROXIMITY**

The proximity data of the proximity channel

enum **touch_sleep_wakeup_level_t**

The chip sleep level that allows the touch sensor to wake-up.

Values:

enumerator **TOUCH_LIGHT_SLEEP_WAKEUP**

Only enable the touch sensor to wake up the chip from light sleep

enumerator **TOUCH_DEEP_SLEEP_WAKEUP**

Enable the touch sensor to wake up the chip from deep sleep or light sleep

enum **touch_chan_shield_cap_t**

Touch sensor shield channel drive capability level.

Values:

enumerator **TOUCH_SHIELD_CAP_40PF**

The max equivalent capacitance in shield channel is 40pf

enumerator **TOUCH_SHIELD_CAP_80PF**

The max equivalent capacitance in shield channel is 80pf

enumerator **TOUCH_SHIELD_CAP_120PF**

The max equivalent capacitance in shield channel is 120pf

enumerator **TOUCH_SHIELD_CAP_160PF**

The max equivalent capacitance in shield channel is 160pf

enumerator **TOUCH_SHIELD_CAP_200PF**

The max equivalent capacitance in shield channel is 200pf

enumerator **TOUCH_SHIELD_CAP_240PF**

The max equivalent capacitance in shield channel is 240pf

enumerator **TOUCH_SHIELD_CAP_280PF**

The max equivalent capacitance in shield channel is 280pf

enumerator **TOUCH_SHIELD_CAP_320PF**

The max equivalent capacitance in shield channel is 320pf

enum **touch_benchmark_filter_mode_t**

Touch channel Infinite Impulse Response (IIR) filter or Jitter filter for benchmark.

备注： Recommended filter coefficient selection is `IIR_16`.

Values:

enumerator **TOUCH_BM_IIR_FILTER_4**

IIR Filter for benchmark, 1/4 raw_value + 3/4 benchmark

enumerator **TOUCH_BM_IIR_FILTER_8**

IIR Filter for benchmark, 1/8 raw_value + 7/8 benchmark

enumerator **TOUCH_BM_IIR_FILTER_16**

IIR Filter for benchmark, 1/16 raw_value + 15/16 benchmark (typical)

enumerator **TOUCH_BM_IIR_FILTER_32**

IIR Filter for benchmark, 1/32 raw_value + 31/32 benchmark

enumerator **TOUCH_BM_IIR_FILTER_64**

IIR Filter for benchmark, 1/64 raw_value + 63/64 benchmark

enumerator **TOUCH_BM_IIR_FILTER_128**

IIR Filter for benchmark, 1/128 raw_value + 127/128 benchmark

enumerator **TOUCH_BM_JITTER_FILTER**

Jitter Filter for benchmark, raw value +/- jitter_step

enum **touch_smooth_filter_mode_t**

Touch channel Infinite Impulse Response (IIR) filter for smooth data.

Values:

enumerator **TOUCH_SMOOTH_NO_FILTER**

No filter adopted for smooth data, smooth data equals raw data

enumerator **TOUCH_SMOOTH_IIR_FILTER_2**

IIR filter adopted for smooth data, smooth data equals 1/2 raw data + 1/2 last smooth data (typical)

enumerator **TOUCH_SMOOTH_IIR_FILTER_4**

IIR filter adopted for smooth data, smooth data equals 1/4 raw data + 3/4 last smooth data

enumerator **TOUCH_SMOOTH_IIR_FILTER_8**

IIR filter adopted for smooth data, smooth data equals 1/8 raw data + 7/8 last smooth data

enum **touch_intr_event_t**

Interrupt events.

Values:

enumerator **TOUCH_INTR_EVENT_ACTIVE**

Touch channel active event

enumerator **TOUCH_INTR_EVENT_INACTIVE**

Touch channel inactive event

enumerator **TOUCH_INTR_EVENT_MEASURE_DONE**

Touch channel measure done event

enumerator **TOUCH_INTR_EVENT_SCAN_DONE**

All touch channels scan done event

enumerator **TOUCH_INTR_EVENT_TIMEOUT**

Touch channel measurement timeout event

enumerator **TOUCH_INTR_EVENT_PROXIMITY_DONE**

Proximity channel measurement done event

2.5.35 Two-Wire Automotive Interface (TWAI)

Overview

The Two-Wire Automotive Interface (TWAI) is a real-time serial communication protocol suited for automotive and industrial applications. It is compatible with ISO11898-1 Classical frames, thus can support Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID). The ESP32-P4 contains 3 TWAI controller(s) that can be configured to communicate on a TWAI bus via an external transceiver.

警告: The TWAI controller is not compatible with ISO11898-1 FD Format frames, and will interpret such frames as errors.

This programming guide is split into the following sections:

Sections

- *Two-Wire Automotive Interface (TWAI)*

- *Overview*
- *TWAI Protocol Summary*
- *Signals Lines and Transceiver*
- *API Naming Conventions*
- *Driver Configuration*
- *Driver Operation*
- *Examples*
- *API Reference*

TWAI Protocol Summary

The TWAI is a multi-master, multi-cast, asynchronous, serial communication protocol. TWAI also supports error detection and signalling, and inbuilt message prioritization.

Multi-master: Any node on the bus can initiate the transfer of a message.

Multi-cast: When a node transmits a message, all nodes on the bus will receive the message (i.e., broadcast) thus ensuring data consistency across all nodes. However, some nodes can selectively choose which messages to accept via the use of acceptance filtering (multi-cast).

Asynchronous: The bus does not contain a clock signal. All nodes on the bus operate at the same bit rate and synchronize using the edges of the bits transmitted on the bus.

Error Detection and Signaling: Every node constantly monitors the bus. When any node detects an error, it signals the detection by transmitting an error frame. Other nodes will receive the error frame and transmit their own error frames in response. This results in an error detection being propagated to all nodes on the bus.

Message Priorities: Messages contain an ID field. If two or more nodes attempt to transmit simultaneously, the node transmitting the message with the lower ID value will win arbitration of the bus. All other nodes will become receivers ensuring that there is at most one transmitter at any time.

TWAI Messages TWAI Messages are split into Data Frames and Remote Frames. Data Frames are used to deliver a data payload to other nodes, whereas a Remote Frame is used to request a Data Frame from other nodes (other nodes can optionally respond with a Data Frame). Data and Remote Frames have two frame formats known as **Extended Frame** and **Standard Frame** which contain a 29-bit ID and an 11-bit ID respectively. A TWAI message consists of the following fields:

- 29-bit or 11-bit ID: Determines the priority of the message (lower value has higher priority).
- Data Length Code (DLC) between 0 to 8: Indicates the size (in bytes) of the data payload for a Data Frame, or the amount of data to request for a Remote Frame.
- Up to 8 bytes of data for a Data Frame (should match DLC).

Error States and Counters The TWAI protocol implements a feature known as ”fault confinement” where a persistently erroneous node will eventually eliminate itself from the bus. This is implemented by requiring every node to maintain two internal error counters known as the **Transmit Error Counter (TEC)** and the **Receive Error Counter (REC)**. The two error counters are incremented and decremented according to a set of rules (where the counters increase on an error, and decrease on a successful message transmission/reception). The values of the counters are used to determine a node's **error state**, namely **Error Active**, **Error Passive**, and **Bus-Off**.

Error Active: A node is Error Active when **both TEC and REC are less than 128** and indicates that the node is operating normally. Error Active nodes are allowed to participate in bus communications, and will actively signal the detection of any errors by automatically transmitting an **Active Error Flag** over the bus.

Error Passive: A node is Error Passive when **either the TEC or REC becomes greater than or equal to 128**. Error Passive nodes are still able to take part in bus communications, but will instead transmit a **Passive Error Flag** upon detection of an error.

Bus-Off: A node becomes Bus-Off when the **TEC becomes greater than or equal to 256**. A Bus-Off node is unable influence the bus in any manner (essentially disconnected from the bus) thus eliminating itself from the bus. A node will remain in the Bus-Off state until it undergoes bus-off recovery.

Signals Lines and Transceiver

The TWAI controller does not contain a integrated transceiver. Therefore, to connect the TWAI controller to a TWAI bus, **an external transceiver is required**. The type of external transceiver used should depend on the application's physical layer specification (e.g., using SN65HVD23x transceivers for ISO 11898-2 compatibility).

The TWAI controller's interface consists of 4 signal lines known as **TX, RX, BUS-OFF, and CLKOUT**. These four signal lines can be routed through the GPIO Matrix to the ESP32-P4's GPIO pads.

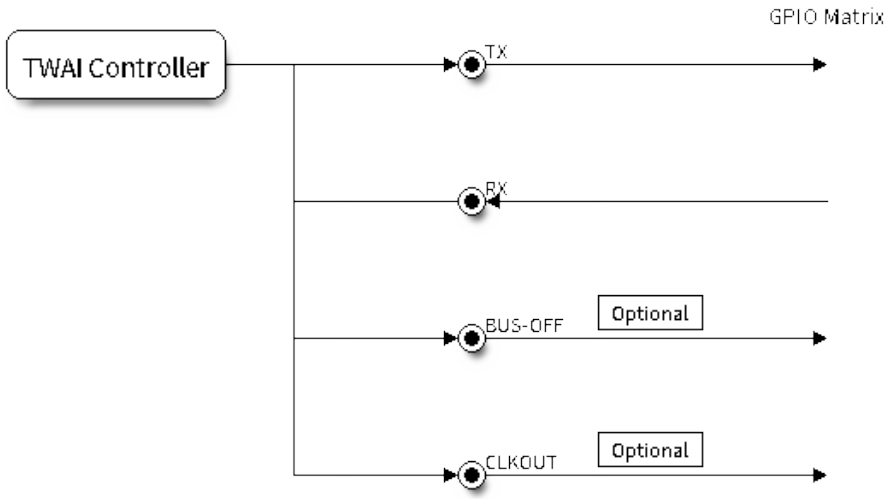


图 39: Signal lines of the TWAI controller

TX and RX: The TX and RX signal lines are required to interface with an external transceiver. Both signal lines represent/interpret a dominant bit as a low logic level (0 V), and a recessive bit as a high logic level (3.3 V).

BUS-OFF: The BUS-OFF signal line is **optional** and is set to a low logic level (0 V) whenever the TWAI controller reaches a bus-off state. The BUS-OFF signal line is set to a high logic level (3.3 V) otherwise.

CLKOUT: The CLKOUT signal line is **optional** and outputs a prescaled version of the controller's source clock.

备注: An external transceiver **must internally loopback the TX to RX** such that a change in logic level to the TX signal line can be observed on the RX line. Failing to do so will cause the TWAI controller to interpret differences in logic levels between the two signal lines as a loss in arbitration or a bit error.

API Naming Conventions

备注: The TWAI driver provides two sets of API. One is handle-free and is widely used in IDF versions earlier than v5.2, but it can only support one TWAI hardware controller. The other set is with handles, and the function name is usually suffixed with "v2", which can support any number of TWAI controllers. These two sets of API can be used at the same time, but it is recommended to use the "v2" version in your new projects.

Driver Configuration

This section covers how to configure the TWAI driver.

Operating Modes The TWAI driver supports the following modes of operations:

Normal Mode: The normal operating mode allows the TWAI controller to take part in bus activities such as transmitting and receiving messages/error frames. Acknowledgement from another node is required when transmitting a message.

No Ack Mode: The No Acknowledgement mode is similar to normal mode, however acknowledgements are not required for a message transmission to be considered successful. This mode is useful when self testing the TWAI controller (loopback of transmissions).

Listen Only Mode: This mode prevents the TWAI controller from influencing the bus. Therefore, transmission of messages/acknowledgement/error frames will be disabled. However the TWAI controller is still able to receive messages but will not acknowledge the message. This mode is suited for bus monitor applications.

Alerts The TWAI driver contains an alert feature that is used to notify the application layer of certain TWAI controller or TWAI bus events. Alerts are selectively enabled when the TWAI driver is installed, but can be reconfigured during runtime by calling `twai_reconfigure_alerts()`. The application can then wait for any enabled alerts to occur by calling `twai_read_alerts()`. The TWAI driver supports the following alerts:

表 5: TWAI Driver Alerts

Alert Flag	Description
TWAI_ALERT_TX_IDLE	No more messages queued for transmission
TWAI_ALERT_TX_SUCCESS	The previous transmission was successful
TWAI_ALERT_RX_DATA	A frame has been received and added to the RX queue
TWAI_ALERT_BELOW_ERR_WARN	Both error counters have dropped below error warning limit
TWAI_ALERT_ERR_ACTIVE	TWAI controller has become error active
TWAI_ALERT_RECOVERY_IN_PROGRESS	TWAI controller is undergoing bus recovery
TWAI_ALERT_BUS_RECOVERED	TWAI controller has successfully completed bus recovery
TWAI_ALERT_ARB_LOST	The previous transmission lost arbitration
TWAI_ALERT_ABOVE_ERR_WARN	One of the error counters have exceeded the error warning limit
TWAI_ALERT_BUS_ERROR	A (Bit, Stuff, CRC, Form, ACK) error has occurred on the bus
TWAI_ALERT_TX_FAILED	The previous transmission has failed
TWAI_ALERT_RX_QUEUE_FULL	The RX queue is full causing a received frame to be lost
TWAI_ALERT_ERR_PASS	TWAI controller has become error passive
TWAI_ALERT_BUS_OFF	Bus-off condition occurred. TWAI controller can no longer influence bus

备注: The TWAI controller's **error warning limit** is used to preemptively warn the application of bus errors before the error passive state is reached. By default, the TWAI driver sets the **error warning limit** to **96**. The TWAI_ALERT_ABOVE_ERR_WARN is raised when the TEC or REC becomes larger then or equal to the error warning limit. The TWAI_ALERT_BELOW_ERR_WARN is raised when both TEC and REC return back to values below **96**.

备注: When enabling alerts, the TWAI_ALERT_AND_LOG flag can be used to cause the TWAI driver to log any raised alerts to UART. However, alert logging is disabled and TWAI_ALERT_AND_LOG if the *CONFIG_TWAI_ISR_IN_IRAM* option is enabled (see *Placing ISR into IRAM*).

备注: The TWAI_ALERT_ALL and TWAI_ALERT_NONE macros can also be used to enable/disable all alerts

during configuration/reconfiguration.

Bit Timing The operating bit rate of the TWAI driver is configured using the *twai_timing_config_t* structure. The period of each bit is made up of multiple **time quanta**, and the period of a **time quantum** is determined by a pre-scaled version of the TWAI controller's source clock. A single bit contains the following segments in the following order:

1. The **Synchronization Segment** consists of a single time quantum
2. **Timing Segment 1** consists of 1 to 16 time quanta before sample point
3. **Timing Segment 2** consists of 1 to 8 time quanta after sample point

The **Baudrate Prescaler** is used to determine the period of each time quantum by dividing the TWAI controller's source clock. On the ESP32-P4, the `brp` can be **any even number from 2 to 32768**. Alternatively, you can decide the resolution of each quantum, by setting *twai_timing_config_t::quanta_resolution_hz* to a non-zero value. In this way, the driver can calculate the underlying `brp` value for you. It is useful when you set different clock sources but want the bitrate to keep the same.

Supported clock source for a TWAI controller is listed in the *twai_clock_source_t* and can be specified in *twai_timing_config_t::clk_src*.

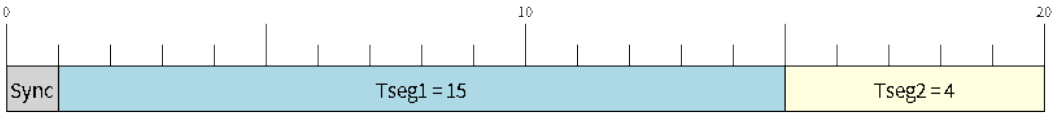


图 40: Bit timing configuration for 500kbit/s given BRP = 8, clock source frequency is 80MHz

The sample point of a bit is located on the intersection of Timing Segment 1 and 2. Enabling **Triple Sampling** causes 3 time quanta to be sampled per bit instead of 1 (extra samples are located at the tail end of Timing Segment 1).

The **Synchronization Jump Width** is used to determine the maximum number of time quanta a single bit time can be lengthened/shortened for synchronization purposes. `sjw` can **range from 1 to 4**.

备注: Multiple combinations of `brp`, `tseg_1`, `tseg_2`, and `sjw` can achieve the same bit rate. Users should tune these values to the physical characteristics of their bus by taking into account factors such as **propagation delay, node information processing time, and phase errors**.

Bit timing **macro initializers** are also available for commonly used bit rates. The following macro initializers are provided by the TWAI driver.

- `TWAI_TIMING_CONFIG_1MBITS`
- `TWAI_TIMING_CONFIG_800KBITS`
- `TWAI_TIMING_CONFIG_500KBITS`
- `TWAI_TIMING_CONFIG_250KBITS`
- `TWAI_TIMING_CONFIG_125KBITS`
- `TWAI_TIMING_CONFIG_100KBITS`
- `TWAI_TIMING_CONFIG_50KBITS`
- `TWAI_TIMING_CONFIG_25KBITS`
- `TWAI_TIMING_CONFIG_20KBITS`
- `TWAI_TIMING_CONFIG_16KBITS`
- `TWAI_TIMING_CONFIG_12_5KBITS`
- `TWAI_TIMING_CONFIG_10KBITS`
- `TWAI_TIMING_CONFIG_5KBITS`

- `TWAI_TIMING_CONFIG_1KBITS`

Acceptance Filter The TWAI controller contains a hardware acceptance filter which can be used to filter messages of a particular ID. A node that filters out a message **does not receive the message, but will still acknowledge it**. Acceptance filters can make a node more efficient by filtering out messages sent over the bus that are irrelevant to the node. The acceptance filter is configured using two 32-bit values within *`twai_filter_config_t`* known as the **acceptance code** and the **acceptance mask**.

The **acceptance code** specifies the bit sequence which a message's ID, RTR, and data bytes must match in order for the message to be received by the TWAI controller. The **acceptance mask** is a bit sequence specifying which bits of the acceptance code can be ignored. This allows for a messages of different IDs to be accepted by a single acceptance code.

The acceptance filter can be used under **Single or Dual Filter Mode**. Single Filter Mode uses the acceptance code and mask to define a single filter. This allows for the first two data bytes of a standard frame to be filtered, or the entirety of an extended frame's 29-bit ID. The following diagram illustrates how the 32-bit acceptance code and mask are interpreted under Single Filter Mode (Note: The yellow and blue fields represent standard and extended frame formats respectively).

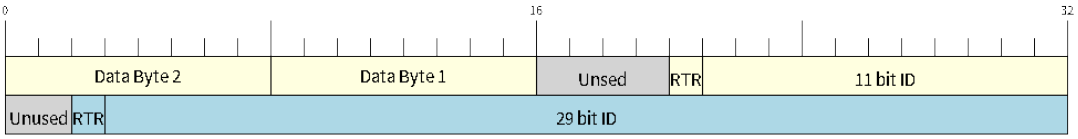


图 41: Bit layout of single filter mode (Right side MSBit)

Dual Filter Mode uses the acceptance code and mask to define two separate filters allowing for increased flexibility of ID's to accept, but does not allow for all 29-bits of an extended ID to be filtered. The following diagram illustrates how the 32-bit acceptance code and mask are interpreted under **Dual Filter Mode** (Note: The yellow and blue fields represent standard and extended frame formats respectively).

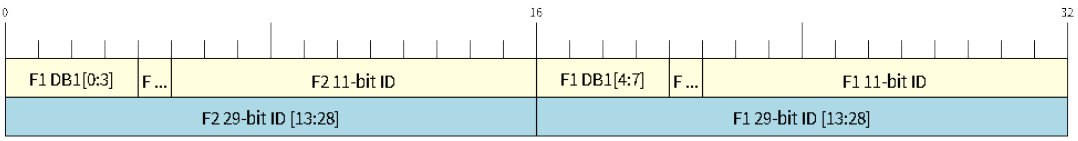


图 42: Bit layout of dual filter mode (Right side MSBit)

Disabling TX Queue The TX queue can be disabled during configuration by setting the `tx_queue_len` member of *`twai_general_config_t`* to `0`. This allows applications that do not require message transmission to save a small amount of memory when using the TWAI driver.

Placing ISR into IRAM The TWAI driver's ISR (Interrupt Service Routine) can be placed into IRAM so that the ISR can still run whilst the cache is disabled. Placing the ISR into IRAM may be necessary to maintain the TWAI driver's functionality during lengthy cache disabling operations (such as SPI Flash writes, OTA updates etc). Whilst the cache is disabled, the ISR continues to:

- Read received messages from the RX buffer and place them into the driver's RX queue.
- Load messages pending transmission from the driver's TX queue and write them into the TX buffer.

To place the TWAI driver's ISR, users must do the following:

- Enable the *CONFIG_TWAI_ISR_IN_IRAM* option using `idf.py menuconfig`.

- When calling *twai_driver_install()*, the intr_flags member of *twai_general_config_t* should set the *ESP_INTR_FLAG_IRAM* set.

备注： When the *CONFIG_TWAI_ISR_IN_IRAM* option is enabled, the TWAI driver will no longer log any alerts (i.e., the TWAI_ALERT_AND_LOG flag will not have any effect).

Driver Operation

The TWAI driver is designed with distinct states and strict rules regarding the functions or conditions that trigger a state transition. The following diagram illustrates the various states and their transitions.

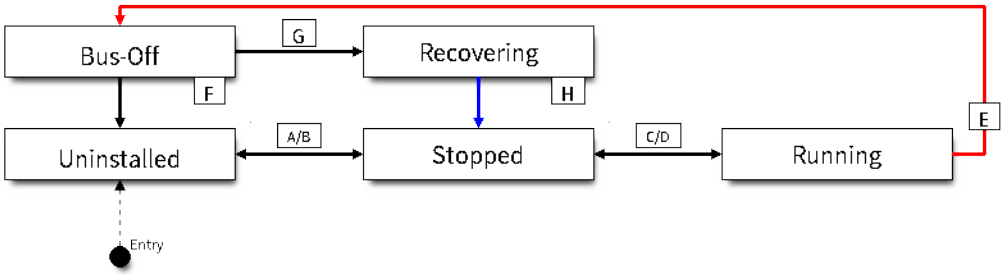


图 43: State transition diagram of the TWAI driver (see table below)

Label	Transition	Action/Condition
A	Uninstalled > Stopped	*twai_driver_install()*
B	Stopped > Uninstalled	*twai_driver_uninstall()*
C	Stopped > Running	*twai_start()*
D	Running > Stopped	*twai_stop()*
E	Running > Bus-Off	Transmit Error Counter >= 256
F	Bus-Off > Uninstalled	*twai_driver_uninstall()*
G	Bus-Off > Recovering	*twai_initiate_recovery()*
H	Recovering > Stopped	128 occurrences of 11 consecutive recessive bits.

Driver States **Uninstalled**: In the uninstalled state, no memory is allocated for the driver and the TWAI controller is powered OFF.

Stopped: In this state, the TWAI controller is powered ON and the TWAI driver has been installed. However the TWAI controller is unable to take part in any bus activities such as transmitting, receiving, or acknowledging messages.

Running: In the running state, the TWAI controller is able to take part in bus activities. Therefore messages can be transmitted/received/acknowledged. Furthermore, the TWAI controller is able to transmit error frames upon detection of errors on the bus.

Bus-Off: The bus-off state is automatically entered when the TWAI controller's Transmit Error Counter becomes greater than or equal to 256. The bus-off state indicates the occurrence of severe errors on the bus or in the TWAI controller. Whilst in the bus-off state, the TWAI controller is unable to take part in any bus activities. To exit the bus-off state, the TWAI controller must undergo the bus recovery process.

Recovering: The recovering state is entered when the TWAI controller undergoes bus recovery. The TWAI controller/TWAI driver remains in the recovering state until the 128 occurrences of 11 consecutive recessive bits is observed on the bus.

Message Fields and Flags The TWAI driver distinguishes different types of messages by using the various bit field members of the *twai_message_t* structure. These bit field members determine whether a message is in standard or extended format, a remote frame, and the type of transmission to use when transmitting such a message.

These bit field members can also be toggled using the `flags` member of *twai_message_t* and the following message flags:

Message Flag	Description
`TWAI_MSG_FLAG_EXTD`	Message is in Extended Frame Format (29bit ID)
`TWAI_MSG_FLAG_RTR`	Message is a Remote Frame (Remote Transmission Request)
`TWAI_MSG_FLAG_SS`	Transmit message using Single Shot Transmission (Message will not be re-transmitted upon error or loss of arbitration). Unused for received message.
`TWAI_MSG_FLAG_SELF`	Transmit message using Self Reception Request (Transmitted message will also received by the same node). Unused for received message.
`TWAI_MSG_FLAG_DLC_NON_COMP`	Message's Data length code is larger than 8. This will break compliance with TWAI
`TWAI_MSG_FLAG_NONE`	Clears all bit fields. Equivalent to a Standard Frame Format (11bit ID) Data Frame.

Examples

Configuration & Installation The following code snippet demonstrates how to configure, install, and start the TWAI driver via the use of the various configuration structures, macro initializers, the *twai_driver_install()* function, and the *twai_start()* function.

```
#include "driver/gpio.h"
#include "driver/twai.h"

void app_main()
{
    //Initialize configuration structures using macro initializers
    twai_general_config_t g_config = TWAI_GENERAL_CONFIG_DEFAULT(GPIO_NUM_21, GPIO_
↪NUM_22, TWAI_MODE_NORMAL);
    twai_timing_config_t t_config = TWAI_TIMING_CONFIG_500KBITS();
    twai_filter_config_t f_config = TWAI_FILTER_CONFIG_ACCEPT_ALL();

    //Install TWAI driver
    if (twai_driver_install(&g_config, &t_config, &f_config) == ESP_OK) {
        printf("Driver installed\n");
    } else {
        printf("Failed to install driver\n");
        return;
    }

    //Start TWAI driver
    if (twai_start() == ESP_OK) {
        printf("Driver started\n");
    } else {
        printf("Failed to start driver\n");
        return;
    }

    ...

}
```

The usage of macro initializers is not mandatory and each of the configuration structures can be manually.

Install Multiple TWAI Instances The following code snippet demonstrates how to install multiple TWAI instances via the use of the *twai_driver_install_v2()* function.

```
#include "driver/gpio.h"
#include "driver/twai.h"

void app_main()
{
    twai_handle_t twai_bus_0;
    twai_handle_t twai_bus_1;
    //Initialize configuration structures using macro initializers
    twai_general_config_t g_config = TWAI_GENERAL_CONFIG_DEFAULT(GPIO_NUM_0, GPIO_
↪NUM_1, TWAI_MODE_NORMAL);
    twai_timing_config_t t_config = TWAI_TIMING_CONFIG_500KBITS();
    twai_filter_config_t f_config = TWAI_FILTER_CONFIG_ACCEPT_ALL();

    //Install driver for TWAI bus 0
    g_config.controller_id = 0;
    if (twai_driver_install_v2(&g_config, &t_config, &f_config, &twai_bus_0) ==␣
↪ESP_OK) {
        printf("Driver installed\n");
    } else {
        printf("Failed to install driver\n");
        return;
    }
    //Start TWAI driver
    if (twai_start_v2(twai_bus_0) == ESP_OK) {
        printf("Driver started\n");
    } else {
        printf("Failed to start driver\n");
        return;
    }

    //Install driver for TWAI bus 1
    g_config.controller_id = 1;
    g_config.tx_io = GPIO_NUM_2;
    g_config.rx_io = GPIO_NUM_3;
    if (twai_driver_install_v2(&g_config, &t_config, &f_config, &twai_bus_1) ==␣
↪ESP_OK) {
        printf("Driver installed\n");
    } else {
        printf("Failed to install driver\n");
        return;
    }
    //Start TWAI driver
    if (twai_start_v2(twai_bus_1) == ESP_OK) {
        printf("Driver started\n");
    } else {
        printf("Failed to start driver\n");
        return;
    }

    //Other Driver operations must use version 2 API as well
    ...

}
```

Message Transmission The following code snippet demonstrates how to transmit a message via the usage of the *twai_message_t* type and *twai_transmit()* function.

```
#include "driver/twai.h"

...

// Configure message to transmit
twai_message_t message = {
    // Message type and format settings
    .extd = 1,              // Standard vs extended format
    .rtr = 0,               // Data vs RTR frame
    .ss = 0,                // Whether the message is single shot (i.e., does not repeat on error)
    .self = 0,              // Whether the message is a self reception request (loopback)
    .dlc_non_comp = 0,      // DLC is less than 8
    // Message ID and payload
    .identifier = 0xAAAA,
    .data_length_code = 4,
    .data = {0, 1, 2, 3},
};

//Queue message for transmission
if (twai_transmit(&message, pdMS_TO_TICKS(1000)) == ESP_OK) {
    printf("Message queued for transmission\n");
} else {
    printf("Failed to queue message for transmission\n");
}
```

Message Reception The following code snippet demonstrates how to receive a message via the usage of the *twai_message_t* type and *twai_receive()* function.

```
#include "driver/twai.h"

...

//Wait for message to be received
twai_message_t message;
if (twai_receive(&message, pdMS_TO_TICKS(10000)) == ESP_OK) {
    printf("Message received\n");
} else {
    printf("Failed to receive message\n");
    return;
}

//Process received message
if (message.extd) {
    printf("Message is in Extended Format\n");
} else {
    printf("Message is in Standard Format\n");
}
printf("ID is %d\n", message.identifier);
if (!(message.rtr)) {
    for (int i = 0; i < message.data_length_code; i++) {
        printf("Data byte %d = %d\n", i, message.data[i]);
    }
}
```

Reconfiguring and Reading Alerts The following code snippet demonstrates how to reconfigure and read TWAI driver alerts via the use of the *twai_reconfigure_alerts()* and *twai_read_alerts()* functions.

```
#include "driver/twai.h"

...

//Reconfigure alerts to detect Error Passive and Bus-Off error states
uint32_t alerts_to_enable = TWAI_ALERT_ERR_PASS | TWAI_ALERT_BUS_OFF;
if (twai_reconfigure_alerts(alerts_to_enable, NULL) == ESP_OK) {
    printf("Alerts reconfigured\n");
} else {
    printf("Failed to reconfigure alerts");
}

//Block indefinitely until an alert occurs
uint32_t alerts_triggered;
twai_read_alerts(&alerts_triggered, portMAX_DELAY);
```

Stop and Uninstall The following code demonstrates how to stop and uninstall the TWAI driver via the use of the *twai_stop()* and *twai_driver_uninstall()* functions.

```
#include "driver/twai.h"

...

//Stop the TWAI driver
if (twai_stop() == ESP_OK) {
    printf("Driver stopped\n");
} else {
    printf("Failed to stop driver\n");
    return;
}

//Uninstall the TWAI driver
if (twai_driver_uninstall() == ESP_OK) {
    printf("Driver uninstalled\n");
} else {
    printf("Failed to uninstall driver\n");
    return;
}
```

Multiple ID Filter Configuration The acceptance mask in *twai_filter_config_t* can be configured such that two or more IDs are accepted for a single filter. For a particular filter to accept multiple IDs, the conflicting bit positions amongst the IDs must be set in the acceptance mask. The acceptance code can be set to any one of the IDs.

The following example shows how the calculate the acceptance mask given multiple IDs:

```
ID1 =  11'b101 1010 0000
ID2 =  11'b101 1010 0001
ID3 =  11'b101 1010 0100
ID4 =  11'b101 1010 1000
//Acceptance Mask
MASK = 11'b000 0000 1101
```

Application Examples **Network Example:** The TWAI Network example demonstrates communication between two ESP32-P4s using the TWAI driver API. One TWAI node acts as a network master that initiates and ceases the transfer of a data from another node acting as a network slave. The example can be found via peripherals/twai/twai_network.

Alert and Recovery Example: This example demonstrates how to use the TWAI driver's alert and bus-off recovery API. The example purposely introduces errors on the bus to put the TWAI controller into the Bus-Off state.

An alert is used to detect the Bus-Off state and trigger the bus recovery process. The example can be found via peripherals/twai/twai_alert_and_recovery.

Self Test Example: This example uses the No Acknowledge Mode and Self Reception Request to cause the TWAI controller to send and simultaneously receive a series of messages. This example can be used to verify if the connections between the TWAI controller and the external transceiver are working correctly. The example can be found via peripherals/twai/twai_self_test.

API Reference

Header File

- components/hal/include/hal/twai_types.h
- This header file can be included with:

```
#include "hal/twai_types.h"
```

Structures

struct **twai_message_t**

Structure to store a TWAI message.

备注： The flags member is deprecated

Public Members

uint32_t **extd**

Extended Frame Format (29bit ID)

uint32_t **rtr**

Message is a Remote Frame

uint32_t **ss**

Transmit as a Single Shot Transmission. Unused for received.

uint32_t **self**

Transmit as a Self Reception Request. Unused for received.

uint32_t **dlc_non_comp**

Message's Data length code is larger than 8. This will break compliance with ISO 11898-1

uint32_t **reserved**

Reserved bits

uint32_t **flags**

Deprecated: Alternate way to set bits using message flags

uint32_t **identifier**

11 or 29 bit identifier

uint8_t **data_length_code**

Data length code

uint8_t **data**[TWAI_FRAME_MAX_DLC]

Data bytes (not relevant in RTR frame)

struct **twai_timing_config_t**

Structure for bit timing configuration of the TWAI driver.

备注: Macro initializers are available for this structure

Public Members

twai_clock_source_t **clk_src**

Clock source, set to 0 or TWAI_CLK_SRC_DEFAULT if you want a default clock source

uint32_t **quanta_resolution_hz**

The resolution of one timing quanta, in Hz. Note: the value of `brp` will reflected by this field if it's non-zero, otherwise, `brp` needs to be set manually

uint32_t **brp**

Baudrate prescale (i.e., clock divider). Any even number from 2 to 128 for ESP32, 2 to 32768 for non-ESP32 chip. Note: For ESP32 ECO 2 or later, multiples of 4 from 132 to 256 are also supported

uint8_t **tseg_1**

Timing segment 1 (Number of time quanta, between 1 to 16)

uint8_t **tseg_2**

Timing segment 2 (Number of time quanta, 1 to 8)

uint8_t **sjw**

Synchronization Jump Width (Max time quanta jump for synchronize from 1 to 4)

bool **triple_sampling**

Enables triple sampling when the TWAI controller samples a bit

struct **twai_filter_config_t**

Structure for acceptance filter configuration of the TWAI driver (see documentation)

备注: Macro initializers are available for this structure

Public Members

uint32_t **acceptance_code**

32-bit acceptance code

uint32_t **acceptance_mask**

32-bit acceptance mask

bool **single_filter**

Use Single Filter Mode (see documentation)

Macros

TWAI_EXTD_ID_MASK

TWAI Constants.

Bit mask for 29 bit Extended Frame Format ID

TWAI_STD_ID_MASK

Bit mask for 11 bit Standard Frame Format ID

TWAI_FRAME_MAX_DLC

Max data bytes allowed in TWAI

TWAI_FRAME_EXTD_ID_LEN_BYTES

EFF ID requires 4 bytes (29bit)

TWAI_FRAME_STD_ID_LEN_BYTES

SFF ID requires 2 bytes (11bit)

TWAI_ERR_PASS_THRESH

Error counter threshold for error passive

Type Definitions

typedef *soc_periph_twai_clk_src_t* **twai_clock_source_t**

RMT group clock source.

> **备注:** User should select the clock source based on the power and resolution requirement

Enumerations

enum **twai_mode_t**

TWAI Controller operating modes.

Values:

enumerator **TWAI_MODE_NORMAL**

Normal operating mode where TWAI controller can send/receive/acknowledge messages

enumerator **TWAI_MODE_NO_ACK**

Transmission does not require acknowledgment. Use this mode for self testing

enumerator **TWAI_MODE_LISTEN_ONLY**

The TWAI controller will not influence the bus (No transmissions or acknowledgments) but can receive messages

Header File

- components/driver/twai/include/driver/twai.h
- This header file can be included with:

```
#include "driver/twai.h"
```

- This header file is a part of the API provided by the `driver` component. To declare that your component depends on `driver`, add the following to your CMakeLists.txt:

```
REQUIRES driver
```

or

```
PRIV_REQUIRES driver
```

Functions

esp_err_t **twai_driver_install** (const *twai_general_config_t* *g_config, const *twai_timing_config_t* *t_config, const *twai_filter_config_t* *f_config)

Install TWAI driver.

This function installs the TWAI driver using three configuration structures. The required memory is allocated and the TWAI driver is placed in the stopped state after running this function.

备注: Macro initializers are available for the configuration structures (see documentation)

备注: To reinstall the TWAI driver, call twai_driver_uninstall() first

参数

- **g_config** -- **[in]** General configuration structure
- **t_config** -- **[in]** Timing configuration structure
- **f_config** -- **[in]** Filter configuration structure

返回

- ESP_OK: Successfully installed TWAI driver
- ESP_ERR_INVALID_ARG: Arguments are invalid, e.g. invalid clock source, invalid quanta resolution
- ESP_ERR_NO_MEM: Insufficient memory
- ESP_ERR_INVALID_STATE: Driver is already installed

esp_err_t **twai_driver_install_v2** (const *twai_general_config_t* *g_config, const *twai_timing_config_t* *t_config, const *twai_filter_config_t* *f_config, *twai_handle_t* *ret_twai)

Install TWAI driver and return a handle.

备注: This is an advanced version of `twai_driver_install` that can return a driver handle, so that it allows you to install multiple TWAI drivers. Don't forget to set the proper controller_id in the *twai_general_config_t* Please refer to the documentation of `twai_driver_install` for more details.

参数

- **g_config** -- **[in]** General configuration structure
- **t_config** -- **[in]** Timing configuration structure
- **f_config** -- **[in]** Filter configuration structure
- **ret_twai** -- **[out]** Pointer to a new created TWAI handle

返回

- ESP_OK: Successfully installed TWAI driver
- ESP_ERR_INVALID_ARG: Arguments are invalid, e.g. invalid clock source, invalid quanta resolution, invalid controller ID
- ESP_ERR_NO_MEM: Insufficient memory
- ESP_ERR_INVALID_STATE: Driver is already installed

esp_err_t **twai_driver_uninstall**(void)

Uninstall the TWAI driver.

This function uninstalls the TWAI driver, freeing the memory utilized by the driver. This function can only be called when the driver is in the stopped state or the bus-off state.

警告： The application must ensure that no tasks are blocked on TX/RX queues or alerts when this function is called.

返回

- ESP_OK: Successfully uninstalled TWAI driver
- ESP_ERR_INVALID_STATE: Driver is not in stopped/bus-off state, or is not installed

esp_err_t **twai_driver_uninstall_v2**(*twai_handle_t* handle)

Uninstall the TWAI driver with a given handle.

备注： This is an advanced version of `twai_driver_uninstall` that can uninstall a TWAI driver with a given handle. Please refer to the documentation of `twai_driver_uninstall` for more details.

参数 **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`

返回

- ESP_OK: Successfully uninstalled TWAI driver
- ESP_ERR_INVALID_STATE: Driver is not in stopped/bus-off state, or is not installed

esp_err_t **twai_start**(void)

Start the TWAI driver.

This function starts the TWAI driver, putting the TWAI driver into the running state. This allows the TWAI driver to participate in TWAI bus activities such as transmitting/receiving messages. The TX and RX queue are reset in this function, clearing any messages that are unread or pending transmission. This function can only be called when the TWAI driver is in the stopped state.

返回

- ESP_OK: TWAI driver is now running
- ESP_ERR_INVALID_STATE: Driver is not in stopped state, or is not installed

esp_err_t **twai_start_v2**(*twai_handle_t* handle)

Start the TWAI driver with a given handle.

备注： This is an advanced version of `twai_start` that can start a TWAI driver with a given handle. Please refer to the documentation of `twai_start` for more details.

参数 **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`

返回

- ESP_OK: TWAI driver is now running
- ESP_ERR_INVALID_STATE: Driver is not in stopped state, or is not installed

esp_err_t **twai_stop**(void)

Stop the TWAI driver.

This function stops the TWAI driver, preventing any further message from being transmitted or received until twai_start() is called. Any messages in the TX queue are cleared. Any messages in the RX queue should be read by the application after this function is called. This function can only be called when the TWAI driver is in the running state.

警告： A message currently being transmitted/received on the TWAI bus will be ceased immediately. This may lead to other TWAI nodes interpreting the unfinished message as an error.

返回

- ESP_OK: TWAI driver is now Stopped
- ESP_ERR_INVALID_STATE: Driver is not in running state, or is not installed

esp_err_t **twai_stop_v2**(*twai_handle_t* handle)

Stop the TWAI driver with a given handle.

备注： This is an advanced version of `twai_stop` that can stop a TWAI driver with a given handle. Please refer to the documentation of `twai_stop` for more details.

参数 **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`

返回

- ESP_OK: TWAI driver is now Stopped
- ESP_ERR_INVALID_STATE: Driver is not in running state, or is not installed

esp_err_t **twai_transmit**(const *twai_message_t* *message, TickType_t ticks_to_wait)

Transmit a TWAI message.

This function queues a TWAI message for transmission. Transmission will start immediately if no other messages are queued for transmission. If the TX queue is full, this function will block until more space becomes available or until it times out. If the TX queue is disabled (TX queue length = 0 in configuration), this function will return immediately if another message is undergoing transmission. This function can only be called when the TWAI driver is in the running state and cannot be called under Listen Only Mode.

备注： This function does not guarantee that the transmission is successful. The TX_SUCCESS/TX_FAILED alert can be enabled to alert the application upon the success/failure of a transmission.

备注： The TX_IDLE alert can be used to alert the application when no other messages are awaiting transmission.

参数

- **message** -- **[in]** Message to transmit
- **ticks_to_wait** -- **[in]** Number of FreeRTOS ticks to block on the TX queue

返回

- ESP_OK: Transmission successfully queued/initiated
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_TIMEOUT: Timed out waiting for space on TX queue
- ESP_FAIL: TX queue is disabled and another message is currently transmitting
- ESP_ERR_INVALID_STATE: TWAI driver is not in running state, or is not installed
- ESP_ERR_NOT_SUPPORTED: Listen Only Mode does not support transmissions

esp_err_t **twai_transmit_v2** (*twai_handle_t* handle, const *twai_message_t* *message, TickType_t ticks_to_wait)

Transmit a TWAI message via a given handle.

备注： This is an advanced version of `twai_transmit` that can transmit a TWAI message with a given handle. Please refer to the documentation of `twai_transmit` for more details.

参数

- **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`
- **message** -- **[in]** Message to transmit
- **ticks_to_wait** -- **[in]** Number of FreeRTOS ticks to block on the TX queue

返回

- ESP_OK: Transmission successfully queued/initiated
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_TIMEOUT: Timed out waiting for space on TX queue
- ESP_FAIL: TX queue is disabled and another message is currently transmitting
- ESP_ERR_INVALID_STATE: TWAI driver is not in running state, or is not installed
- ESP_ERR_NOT_SUPPORTED: Listen Only Mode does not support transmissions

esp_err_t **twai_receive** (*twai_message_t* *message, TickType_t ticks_to_wait)

Receive a TWAI message.

This function receives a message from the RX queue. The flags field of the message structure will indicate the type of message received. This function will block if there are no messages in the RX queue

警告： The flags field of the received message should be checked to determine if the received message contains any data bytes.

参数

- **message** -- **[out]** Received message
- **ticks_to_wait** -- **[in]** Number of FreeRTOS ticks to block on RX queue

返回

- ESP_OK: Message successfully received from RX queue
- ESP_ERR_TIMEOUT: Timed out waiting for message
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_receive_v2** (*twai_handle_t* handle, *twai_message_t* *message, TickType_t ticks_to_wait)

Receive a TWAI message via a given handle.

备注： This is an advanced version of `twai_receive` that can receive a TWAI message with a given handle. Please refer to the documentation of `twai_receive` for more details.

参数

- **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`
- **message** -- **[out]** Received message
- **ticks_to_wait** -- **[in]** Number of FreeRTOS ticks to block on RX queue

返回

- ESP_OK: Message successfully received from RX queue
- ESP_ERR_TIMEOUT: Timed out waiting for message
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_read_alerts**(uint32_t *alerts, TickType_t ticks_to_wait)

Read TWAI driver alerts.

This function will read the alerts raised by the TWAI driver. If no alert has been issued when this function is called, this function will block until an alert occurs or until it timeouts.

备注： Multiple alerts can be raised simultaneously. The application should check for all alerts that have been enabled.

参数

- **alerts** -- **[out]** Bit field of raised alerts (see documentation for alert flags)
- **ticks_to_wait** -- **[in]** Number of FreeRTOS ticks to block for alert

返回

- ESP_OK: Alerts read
- ESP_ERR_TIMEOUT: Timed out waiting for alerts
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_read_alerts_v2**(*twai_handle_t* handle, uint32_t *alerts, TickType_t ticks_to_wait)

Read TWAI driver alerts with a given handle.

备注： This is an advanced version of `twai_read_alerts` that can read TWAI driver alerts with a given handle. Please refer to the documentation of `twai_read_alerts` for more details.

参数

- **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`
- **alerts** -- **[out]** Bit field of raised alerts (see documentation for alert flags)
- **ticks_to_wait** -- **[in]** Number of FreeRTOS ticks to block for alert

返回

- ESP_OK: Alerts read
- ESP_ERR_TIMEOUT: Timed out waiting for alerts
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_reconfigure_alerts**(uint32_t alerts_enabled, uint32_t *current_alerts)

Reconfigure which alerts are enabled.

This function reconfigures which alerts are enabled. If there are alerts which have not been read whilst reconfiguring, this function can read those alerts.

参数

- **alerts_enabled** -- **[in]** Bit field of alerts to enable (see documentation for alert flags)
- **current_alerts** -- **[out]** Bit field of currently raised alerts. Set to NULL if unused

返回

- ESP_OK: Alerts reconfigured
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_reconfigure_alerts_v2**(*twai_handle_t* handle, uint32_t alerts_enabled, uint32_t *current_alerts)

Reconfigure which alerts are enabled, with a given handle.

备注： This is an advanced version of `twai_reconfigure_alerts` that can reconfigure which alerts are enabled with a given handle. Please refer to the documentation of `twai_reconfigure_alerts` for more details.

参数

- **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`
- **alerts_enabled** -- **[in]** Bit field of alerts to enable (see documentation for alert flags)
- **current_alerts** -- **[out]** Bit field of currently raised alerts. Set to NULL if unused

返回

- ESP_OK: Alerts reconfigured
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_initiate_recovery**(void)

Start the bus recovery process.

This function initiates the bus recovery process when the TWAI driver is in the bus-off state. Once initiated, the TWAI driver will enter the recovering state and wait for 128 occurrences of the bus-free signal on the TWAI bus before returning to the stopped state. This function will reset the TX queue, clearing any messages pending transmission.

备注: The BUS_RECOVERED alert can be enabled to alert the application when the bus recovery process completes.

返回

- ESP_OK: Bus recovery started
- ESP_ERR_INVALID_STATE: TWAI driver is not in the bus-off state, or is not installed

esp_err_t **twai_initiate_recovery_v2**(*twai_handle_t* handle)

Start the bus recovery process with a given handle.

备注: This is an advanced version of `twai_initiate_recovery` that can start the bus recovery process with a given handle. Please refer to the documentation of `twai_initiate_recovery` for more details.

参数 **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`

返回

- ESP_OK: Bus recovery started
- ESP_ERR_INVALID_STATE: TWAI driver is not in the bus-off state, or is not installed

esp_err_t **twai_get_status_info**(*twai_status_info_t* *status_info)

Get current status information of the TWAI driver.

参数 **status_info** -- **[out]** Status information

返回

- ESP_OK: Status information retrieved
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_get_status_info_v2**(*twai_handle_t* handle, *twai_status_info_t* *status_info)

Get current status information of a given TWAI driver handle.

备注: This is an advanced version of `twai_get_status_info` that can get current status information of a given TWAI driver handle. Please refer to the documentation of `twai_get_status_info` for more details.

参数

- **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`
- **status_info** -- **[out]** Status information

返回

- ESP_OK: Status information retrieved
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_clear_transmit_queue**(void)

Clear the transmit queue.

This function will clear the transmit queue of all messages.

备注: The transmit queue is automatically cleared when twai_stop() or twai_initiate_recovery() is called.

返回

- ESP_OK: Transmit queue cleared
- ESP_ERR_INVALID_STATE: TWAI driver is not installed or TX queue is disabled

esp_err_t **twai_clear_transmit_queue_v2**(*twai_handle_t* handle)

Clear the transmit queue of a given TWAI driver handle.

备注: This is an advanced version of `twai_clear_transmit_queue` that can clear the transmit queue of a given TWAI driver handle. Please refer to the documentation of `twai_clear_transmit_queue` for more details.

参数 **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`

返回

- ESP_OK: Transmit queue cleared
- ESP_ERR_INVALID_STATE: TWAI driver is not installed or TX queue is disabled

esp_err_t **twai_clear_receive_queue**(void)

Clear the receive queue.

This function will clear the receive queue of all messages.

备注: The receive queue is automatically cleared when twai_start() is called.

返回

- ESP_OK: Transmit queue cleared
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t **twai_clear_receive_queue_v2**(*twai_handle_t* handle)

Clear the receive queue of a given TWAI driver handle.

备注: This is an advanced version of `twai_clear_receive_queue` that can clear the receive queue of a given TWAI driver handle. Please refer to the documentation of `twai_clear_receive_queue` for more details.

参数 **handle** -- **[in]** TWAI driver handle returned by `twai_driver_install_v2`

返回

- ESP_OK: Transmit queue cleared
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

Structures

struct **twai_general_config_t**

Structure for general configuration of the TWAI driver.

备注: Macro initializers are available for this structure

Public Members

int **controller_id**

TWAI controller ID, index from 0. If you want to install TWAI driver with a non-zero controller_id, please use `twai_driver_install_v2`

twai_mode_t **mode**

Mode of TWAI controller

gpio_num_t **tx_io**

Transmit GPIO number

gpio_num_t **rx_io**

Receive GPIO number

gpio_num_t **clkout_io**

CLKOUT GPIO number (optional, set to -1 if unused)

gpio_num_t **bus_off_io**

Bus off indicator GPIO number (optional, set to -1 if unused)

uint32_t **tx_queue_len**

Number of messages TX queue can hold (set to 0 to disable TX Queue)

uint32_t **rx_queue_len**

Number of messages RX queue can hold

uint32_t **alerts_enabled**

Bit field of alerts to enable (see documentation)

uint32_t **clkout_divider**

CLKOUT divider. Can be 1 or any even number from 2 to 14 (optional, set to 0 if unused)

int **intr_flags**

Interrupt flags to set the priority of the driver's ISR. Note that to use the ESP_INTR_FLAG_IRAM, the CONFIG_TWAI_ISR_IN_IRAM option should be enabled first.

struct **twai_status_info_t**

Structure to store status information of TWAI driver.

Public Members

twai_state_t **state**

Current state of TWAI controller (Stopped/Running/Bus-Off/Recovery)

uint32_t **msgs_to_tx**

Number of messages queued for transmission or awaiting transmission completion

uint32_t **msgs_to_rx**

Number of messages in RX queue waiting to be read

uint32_t **tx_error_counter**

Current value of Transmit Error Counter

uint32_t **rx_error_counter**

Current value of Receive Error Counter

uint32_t **tx_failed_count**

Number of messages that failed transmissions

uint32_t **rx_missed_count**

Number of messages that were lost due to a full RX queue (or errata workaround if enabled)

uint32_t **rx_overrun_count**

Number of messages that were lost due to a RX FIFO overrun

uint32_t **arb_lost_count**

Number of instances arbitration was lost

uint32_t **bus_error_count**

Number of instances a bus error has occurred

Macros

TWAI_IO_UNUSED

Marks GPIO as unused in TWAI configuration

Type Definitions

typedef struct twai_obj_t ***twai_handle_t**

TWAI controller handle.

Enumerations

enum **twai_state_t**

TWAI driver states.

Values:

enumerator **TWAI_STATE_STOPPED**

Stopped state. The TWAI controller will not participate in any TWAI bus activities

enumerator **TWAI_STATE_RUNNING**

Running state. The TWAI controller can transmit and receive messages

enumerator **TWAI_STATE_BUS_OFF**

Bus-off state. The TWAI controller cannot participate in bus activities until it has recovered

enumerator **TWAI_STATE_RECOVERING**

Recovering state. The TWAI controller is undergoing bus recovery

2.5.36 通用异步接收器/发送器 (UART)

简介

通用异步接收器/发送器 (UART) 属于一种硬件功能，通过使用 RS232、RS422、RS485 等常见异步串行通信接口来处理通信时序要求和数据帧。UART 是实现不同设备之间全双工或半双工数据交换的一种常用且经济的方式。

ESP32-P4 芯片有 5 个 UART 控制器（也称为端口），每个控制器都有一组相同的寄存器以简化编程并提高灵活性。

每个 UART 控制器可以独立配置波特率、数据位长度、位顺序、停止位位数、奇偶校验位等参数。所有具备完整功能的 UART 控制器都能与不同制造商的 UART 设备兼容，并且支持红外数据协会 (IrDA) 定义的标准协议。

此外，ESP32-P4 芯片还有一个满足低功耗需求的 LP UART 控制器。LP UART 是原 UART 的功能剪裁版本。它只支持基础 UART 功能，不支持 IrDA 或 RS485 协议，并且只有一块较小的 RAM 存储空间。想要全面了解的 UART 及 LP UART 功能区别，请参考 **ESP32-P4 技术参考手册** > UART 控制器 (UART) > 主要特性 [PDF]。

功能概述

下文介绍了如何使用 UART 驱动程序的函数和数据类型在 ESP32-P4 和其他 UART 设备之间建立通信。基本编程流程分为以下几个步骤：

1. 设置通信参数 - 设置波特率、数据位、停止位等
2. 设置通信管脚 - 分配连接设备的管脚
3. 安装驱动程序 - 为 UART 驱动程序分配 ESP32-P4 资源
4. 运行 *UART* 通信 - 发送/接收数据
5. 使用中断 - 触发特定通信事件的中断
6. 删除驱动程序 - 如无需 UART 通信，则释放已分配的资源

步骤 1 到 3 为配置阶段，步骤 4 为 UART 运行阶段，步骤 5 和 6 为可选步骤。

此外，LP UART 控制器的编程需要注意使用主核驱动 *LP UART* 控制器。

UART 驱动程序函数通过 `uart_port_t` 识别不同的 UART 控制器。调用以下所有函数均需此标识。

设置通信参数 UART 通信参数可以在一个步骤中完成全部配置，也可以在多个步骤中单独配置。

一次性配置所有参数 调用函数*uart_param_config()* 并向其传递*uart_config_t* 结构体，*uart_config_t* 结构体应包含所有必要的参数。请参考以下示例。

```
const uart_port_t uart_num = UART_NUM_1;
uart_config_t uart_config = {
    .baud_rate = 115200,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_CTS_RTS,
    .rx_flow_ctrl_thresh = 122,
};
// Configure UART parameters
ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
```

了解配置硬件流控模式的更多信息，请参考 peripherals/uart/uart_echo。

此外，置位*uart_config_t::backup_before_sleep* 会使能在进入睡眠模式前备份 UART 配置寄存器并在退出睡眠后恢复这些寄存器。这个功能使 UART 能够在系统唤醒后继续正常工作，即使其电源域在睡眠过程中被完全关闭。此选项需要用户在功耗和内存使用之间取得平衡。如果功耗不是一个问题，可以禁用这个选项来节省内存。

分步依次配置每个参数 调用下表中的专用函数，能够单独配置特定参数。如需重新配置某个参数，也可使用这些函数。

表 6: 单独配置特定参数的函数

配置参数	函数
波特率	*uart_set_baudrate()*
传输位	调用*uart_set_word_length()* 设置*uart_word_length_t*
奇偶控制	调用*uart_parity_t* 设置*uart_set_parity()*
停止位	调用*uart_set_stop_bits()* 设置*uart_stop_bits_t*
硬件流控模式	调用*uart_set_hw_flow_ctrl()* 设置*uart_hw_flowcontrol_t*
通信模式	调用*uart_set_mode()* 设置*uart_mode_t*

表中每个函数都可使用 _get_ 对应项来查看当前设置值。例如，查看当前波特率值，请调用*uart_get_baudrate()*。

设置通信管脚 通信参数设置完成后，可以配置其他 UART 设备连接的 GPIO 管脚。调用函数*uart_set_pin()*，指定配置 Tx、Rx、RTS 和 CTS 信号的 GPIO 管脚编号。如要为特定信号保留当前分配的管脚编号，可传递宏*UART_PIN_NO_CHANGE*。

请为不使用的管脚都指定为宏*UART_PIN_NO_CHANGE*。

```
// Set UART pins(TX: IO4, RX: IO5, RTS: IO18, CTS: IO19)
ESP_ERROR_CHECK(uart_set_pin(UART_NUM_1, 4, 5, 18, 19));
```

安装驱动程序 通信管脚设置完成后，请调用*uart_driver_install()* 安装驱动程序并指定以下参数：

- UART 控制器编号
- Tx 环形缓冲区的大小
- Rx 环形缓冲区的大小
- 指向事件队列句柄的指针
- 事件队列大小
- 分配中断的标志

该函数将为 UART 驱动程序分配所需的内部资源。

```
// Setup UART buffered IO with event queue
const int uart_buffer_size = (1024 * 2);
QueueHandle_t uart_queue;
// Install UART driver using an event queue here
ESP_ERROR_CHECK(uart_driver_install(UART_NUM_1, uart_buffer_size, \
                                        uart_buffer_size, 10, &uart_queue, 0));
```

此步骤完成后，可连接外部 UART 设备检查通信。

运行 UART 通信 串行通信由每个 UART 控制器的有限状态机 (FSM) 控制。

发送数据的过程分为以下步骤：

1. 将数据写入 Tx FIFO 缓冲区
2. FSM 序列化数据
3. FSM 发送数据

接收数据的过程类似，只是步骤相反：

1. FSM 处理且并行化传入的串行流
2. FSM 将数据写入 Rx FIFO 缓冲区
3. 从 Rx FIFO 缓冲区读取数据

因此，应用程序仅会通过*uart_write_bytes()* 和*uart_read_bytes()* 从特定缓冲区写入或读取数据，其余工作由 FSM 完成。

发送数据 发送数据准备好后，调用函数*uart_write_bytes()*，并向其传递数据缓冲区的地址和数据长度。该函数会立即或在有足够可用空间时将数据复制到 Tx 环形缓冲区，随后退出。当 Tx FIFO 缓冲区中有可用空间时，中断服务例程 (ISR) 会在后台将数据从 Tx 环形缓冲区移动到 Tx FIFO 缓冲区。调用函数请参考以下代码。

```
// Write data to UART.
char* test_str = "This is a test string.\n";
uart_write_bytes(uart_num, (const char*)test_str, strlen(test_str));
```

函数*uart_write_bytes_with_break()* 与*uart_write_bytes()* 类似，但在传输结束时会添加串行中断信号。“串行中断信号”意味着 Tx 线保持低电平的时间长于一个数据帧。

```
// Write data to UART, end with a break signal.
uart_write_bytes_with_break(uart_num, "test break\n",strlen("test break\n"), 100);
```

能够将数据写入 Tx FIFO 缓冲区的另一函数是*uart_tx_chars()*。与*uart_write_bytes()* 不同，此函数在没有可用空间之前不会阻塞。相反，它将写入所有可以立即放入硬件 Tx FIFO 的数据，然后返回写入的字节数。

“配套”函数*uart_wait_tx_done()* 用于监听 Tx FIFO 缓冲区的状态，并在缓冲区为空时返回。

```
// Wait for packet to be sent
const uart_port_t uart_num = UART_NUM_1;
ESP_ERROR_CHECK(uart_wait_tx_done(uart_num, 100)); // wait timeout is 100 RTOS␣
↪ticks (TickType_t)
```

接收数据 一旦 UART 接收了数据，并将其保存在 Rx FIFO 缓冲区中，就需要使用函数*uart_read_bytes()* 检索数据。读取数据之前，调用*uart_get_buffered_data_len()* 能够查看 Rx FIFO 缓冲区中可用的字节数。请参考以下示例。

```
// Read data from UART.
const uart_port_t uart_num = UART_NUM_1;
uint8_t data[128];
```

(下页继续)

(续上页)

```
int length = 0;
ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
length = uart_read_bytes(uart_num, data, length, 100);
```

如果不再需要 Rx FIFO 缓冲区中的数据，可以调用*uart_flush()* 清空缓冲区。

软件流控 如果硬件流控处于禁用状态，可使用函数*uart_set_rts()* 和*uart_set_dtr()* 分别手动设置 RTS 和 DTR 信号电平。

通信方式选择 UART 控制器支持多种通信模式，使用函数*uart_set_mode()* 可以选择模式。选择特定模式后，UART 驱动程序将处理已连接 UART 设备的相应行为。例如，使用 RTS 线控制 RS485 驱动芯片，能够实现半双工 RS485 通信。

```
// Setup UART in rs485 half duplex mode
ESP_ERROR_CHECK(uart_set_mode(uart_num, UART_MODE_RS485_HALF_DUPLEX));
```

使用中断 根据特定的 UART 状态或检测到的错误，可以生成许多不同的中断。**ESP32-P4 技术参考手册** > UART 控制器 (UART) > UART 中断和 UHCI 中断 [PDF] 中提供了可用中断的完整列表。调用*uart_enable_intr_mask()* 或*uart_disable_intr_mask()* 能够分别启用或禁用特定中断。

UART 驱动提供了一种便利的方法来处理特定的中断，即将中断包装成相应的事件。这些事件定义在*uart_event_type_t* 中，FreeRTOS 队列功能可将这些事件报告给用户应用程序。

要接收已发生的事件，请调用*uart_driver_install()* 函数并获取返回的事件队列句柄，可参考上述*示例代码*。

UART 驱动可处理的事件包括：

- **FIFO 空间溢出** (*UART_FIFO_OVF*)：当接收到的数据超过 FIFO 的存储能力时，Rx FIFO 会触发中断。
  - （可选）配置 FIFO 阈值：在结构体*uart_intr_config_t* 中输入阈值，然后调用*uart_intr_config()* 使能配置。这有助于驱动及时处理 RX FIFO 中的数据，避免 FIFO 溢出。
  - 启用中断：调用函数*uart_enable_rx_intr()*
  - 禁用中断：调用函数*uart_disable_rx_intr()*

```
const uart_port_t uart_num = UART_NUM_1;
// Configure a UART interrupt threshold and timeout
uart_intr_config_t uart_intr = {
    .intr_enable_mask = UART_INTR_RXFIFO_FULL | UART_INTR_RXFIFO_TOUT,
    .rxfifo_full_thresh = 100,
    .rx_timeout_thresh = 10,
};
ESP_ERROR_CHECK(uart_intr_config(uart_num, &uart_intr));

// Enable UART RX FIFO full threshold and timeout interrupts
ESP_ERROR_CHECK(uart_enable_rx_intr(uart_num));
```

- **模式检测** (*UART_PATTERN_DET*)：在检测到重复接收/发送同一字符的“模式”时触发中断，例如，模式检测可用于检测命令字符串末尾是否存在特定数量的相同字符（“模式”）。可以调用以下函数：
  - 配置并启用此中断：调用*uart_enable_pattern_det_baud_intr()*
  - 禁用中断：调用*uart_disable_pattern_det_intr()*

```
//Set UART pattern detect function
uart_enable_pattern_det_baud_intr(EX_UART_NUM, '+', PATTERN_CHR_NUM, 9, 0, 0);
```

- **其他事件**：UART 驱动可处理的其他事件包括数据接收 (*UART_DATA*)、环形缓冲区已满 (*UART_BUFFER_FULL*)、在停止位后检测到 NULL (*UART_BREAK*)、奇偶校验错误 (*UART_PARITY_ERR*)、以及帧错误 (*UART_FRAME_ERR*)。

括号中的字符串为相应的事件名称。请参考 peripherals/uart/uart_events 中处理 UART 事件的示例。

删除驱动程序 如不再需要与 *uart_driver_install()* 建立通信，则可调用 *uart_driver_delete()* 删除驱动程序，释放已分配的资源。

宏指令 API 还定义了一些宏指令。例如，*UART_HW_FIFO_LEN* 定义了硬件 FIFO 缓冲区的长度，*UART_BITRATE_MAX* 定义了 UART 控制器支持的最大波特率。

使用主核驱动 LP UART 控制器 UART 驱动程序还适配了在 Active 模式下对 LP UART 控制器的驱动。LP UART 的配置流程和普通 UART 没有本质上的差别，除了有以下几点需要注意：

- LP UART 控制器的端口号为 `LP_UART_NUM_0`。
- LP UART 控制器的可选时钟源可以在 *lp_uart_sclk_t* 中找到。
- LP UART 控制器的硬件 FIFO 大小要远小于普通 UART 控制器的硬件 FIFO 大小，其值为 *SOC_LP_UART_FIFO_LEN*。
- LP UART 控制器的 GPIO 引脚只能从 LP GPIO 引脚中选择。

RS485 特定通信模式简介

备注： 下文将使用 `[UART_REGISTER_NAME].[UART_FIELD_BIT]` 指代 UART 寄存器字段/位。了解特定模式位的更多信息，请参考 **ESP32-P4 技术参考手册** > UART 控制器 (UART) > 寄存器摘要 [PDF]。请搜索寄存器名称导航至寄存器描述，找到相应字段/位。

- `UART_RS485_CONF_REG.UART_RS485_EN`：设置此位将启用 RS485 通信模式支持。
- `UART_RS485_CONF_REG.UART_RS485TX_RX_EN`：设置此位，发送器的输出信号将环回到接收器的输入信号。
- `UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN`：设置此位，如果接收器繁忙，发送器仍将发送数据（由硬件自动解决冲突）。

ESP32-P4 的 RS485 UART 硬件能够检测数据报传输期间的信号冲突，并在启用此中断时生成中断 `UART_RS485_CLASH_INT`。术语冲突表示发送的数据报与另一端接收到的数据报不同。数据冲突通常与总线上其他活跃设备的存在有关，或者是由于总线错误而出现。

冲突检测功能允许在激活和触发中断时处理冲突。中断 `UART_RS485_FRM_ERR_INT` 和 `UART_RS485_PARITY_ERR_INT` 可与冲突检测功能一起使用，在 RS485 模式下分别控制帧错误和奇偶校验位错误。UART 驱动程序支持此功能，通过选择 *UART_MODE_RS485_APP_CTRL* 模式可以使用（参考函数 *uart_set_mode()*）。

冲突检测功能可与电路 A 和电路 C 一起使用（参考章节 *接口连接选项*）。在使用电路 A 或 B 时，连接到总线驱动 DE 管脚的 RTS 管脚应由用户应用程序控制。调用函数 *uart_get_collision_flag()* 能够查看是否触发冲突检测标志。

ESP32-P4 UART 控制器本身不支持半双工通信，因其无法自动控制连接到 RS485 总线驱动 RE/DE 输入的 RTS 管脚。然而，半双工通信能够通过 UART 驱动程序对 RTS 管脚的软件控制来实现，调用 *uart_set_mode()* 并选择 *UART_MODE_RS485_HALF_DUPLEX* 模式能够启用这一功能。

主机开始向 Tx FIFO 缓冲区写入数据时，UART 驱动程序会自动置位 RTS 管脚（逻辑 1）；最后一位数据传输完成后，驱动程序就会取消置位 RTS 管脚（逻辑 0）。要使用此模式，软件必须禁用硬件流控功能。此模式适用于下文所有已用电路。

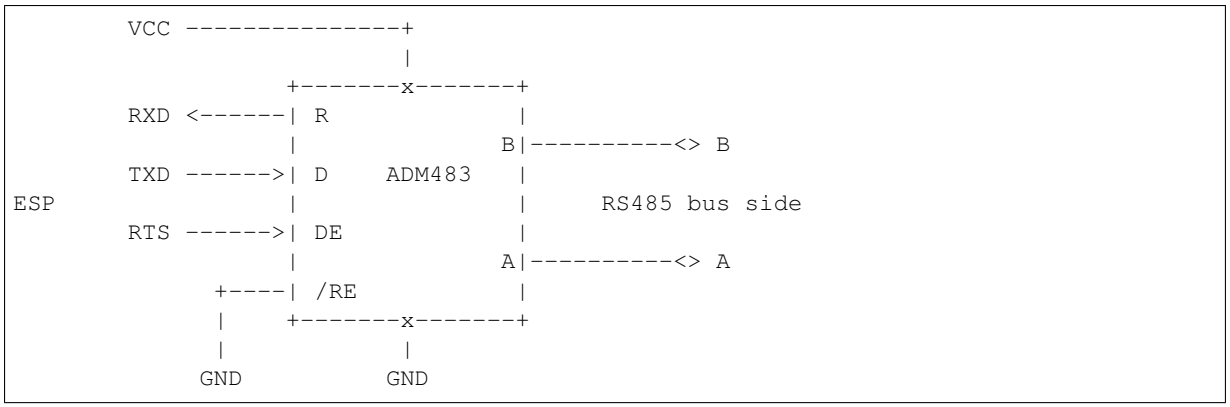
接口连接选项 本节提供了示例原理图来介绍 ESP32-P4 RS485 接口连接的基本内容。

备注：

- **下列原理图不一定包含所有必要元素。**

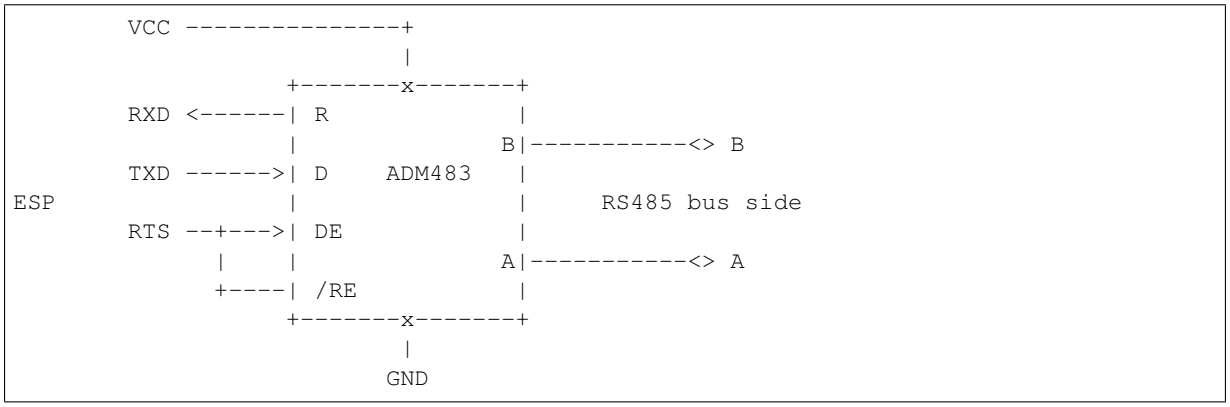
- **模拟设备** ADM483 和 ADM2483 是 RS485 收发器的常见示例，**也可使用其他类似的收发器**。

电路 A：冲突检测电路



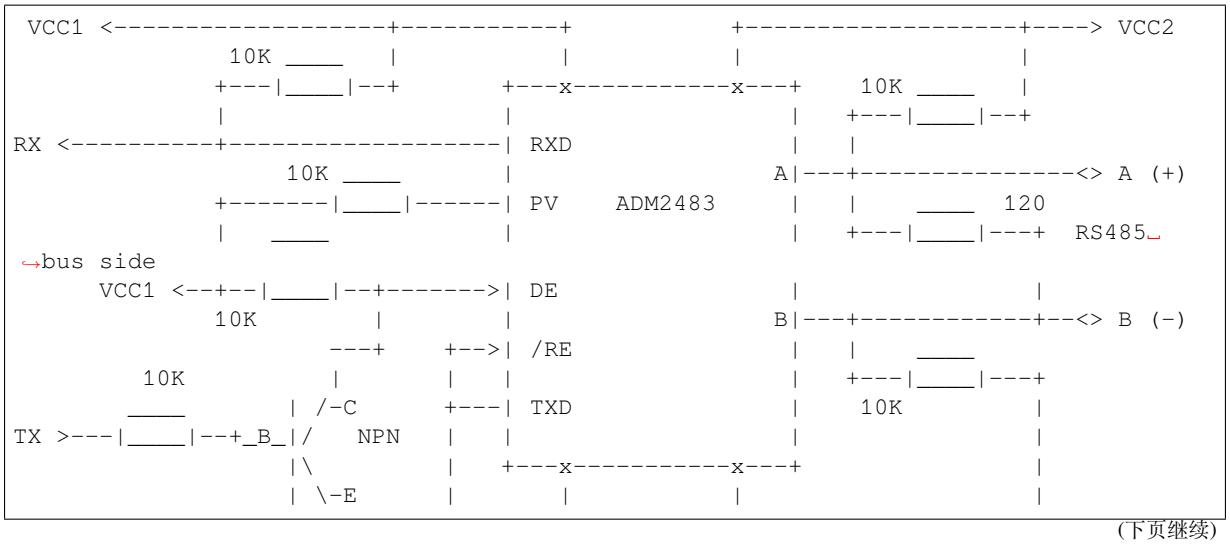
推荐这一电路，因为该电路较为简单，同时能够检测冲突。持续启用线路驱动中的接收器时，UART 将会监控 RS485 总线。启用 `UART_RS485_CONF_REG.UART_RS485TX_RX_EN` 位时，UART 外设会执行回波抑制。

电路 B：无冲突检测的手动切换发射器/接收器



该电路无法检测冲突。置位 `UART_RS485_CONF_REG.UART_RS485TX_RX_EN` 位时，电路将抑制硬件收到的空字节。这种情况下 `UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN` 位不适用。

电路 C：自动切换发射器/接收器



(下页继续)

(续上页)

```
                    |        |        |            |                    |
                   GND1     GND1     GND1         GND2                 GND2
```

这种电气隔离电路不需要用软件应用程序或驱动程序控制 RTS 管脚，因为电路能够自动控制收发器方向。但是在传输过程中，需要将 `UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN` 设置为 1 并将 `UART_RS485_CONF_REG.UART_RS485TX_RX_EN` 设置为 0 来抑制空字节。此设置可以在任何 RS485 UART 模式下工作，包括*UART_MODE_UART*。

应用示例

下表列出了目录 peripherals/uart/ 下可用的代码示例。

代码示例	描述
peripherals/uart/uart_echo	配置 UART 设置、安装 UART 驱动程序以及通过 UART1 接口读取/写入。
peripherals/uart/uart_events	报告各种通信事件，使用模式检测中断。
peripherals/uart/uart_async_rxtxtasks	通过同一 UART 在两个独立的 FreeRTOS 任务中发送和接收数据。
peripherals/uart/uart_select	针对 UART 文件描述符使用同步 I/O 多路复用。
peripherals/uart/uart_echo_rs485	设置 UART 驱动程序以半双工模式通过 RS485 接口进行通信。此示例与 peripherals/uart/uart_echo 类似，但允许通过连接到 ESP32-P4 管脚的 RS485 接口芯片进行通信。
peripherals/uart/nmea0183_parser	解析通过 UART 外设从 GPS 收到的 NMEA0183 语句来获取 GPS 信息。

API 参考

Header File

- components/esp_driver_uart/include/driver/uart.h
- This header file can be included with:

```
#include "driver/uart.h"
```

- This header file is a part of the API provided by the `esp_driver_uart` component. To declare that your component depends on `esp_driver_uart`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_uart
```

or

```
PRIV_REQUIRES esp_driver_uart
```

Functions

esp_err_t **uart_driver_install**(*uart_port_t* uart_num, int rx_buffer_size, int tx_buffer_size, int queue_size, *QueueHandle_t* *uart_queue, int intr_alloc_flags)

Install UART driver and set the UART to the default configuration.

UART ISR handler will be attached to the same CPU core that this function is running on.

备注： Rx_buffer_size should be greater than UART_HW_FIFO_LEN(uart_num). Tx_buffer_size should be either zero or greater than UART_HW_FIFO_LEN(uart_num).

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

- **rx_buffer_size** -- UART RX ring buffer size.
- **tx_buffer_size** -- UART TX ring buffer size. If set to zero, driver will not use TX buffer, TX function will block task until all data have been sent out.
- **queue_size** -- UART event queue size/depth.
- **uart_queue** -- UART event queue handle (out param). On success, a new queue handle is written here to provide access to UART events. If set to NULL, driver will not use an event queue.
- **intr_alloc_flags** -- Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info. Do not set ESP_INTR_FLAG_IRAM here (the driver's ISR handler is not located in IRAM)

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_driver_delete**(*uart_port_t* uart_num)

Uninstall UART driver.

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- ESP_OK Success
- ESP_FAIL Parameter error

bool **uart_is_driver_installed**(*uart_port_t* uart_num)

Checks whether the driver is installed or not.

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- true driver is installed
- false driver is not installed

esp_err_t **uart_set_word_length**(*uart_port_t* uart_num, *uart_word_length_t* data_bit)

Set UART data bits.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **data_bit** -- UART data bits

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_get_word_length**(*uart_port_t* uart_num, *uart_word_length_t* *data_bit)

Get the UART data bit configuration.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **data_bit** -- Pointer to accept value of UART data bits.

返回

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*data_bit)

esp_err_t **uart_set_stop_bits**(*uart_port_t* uart_num, *uart_stop_bits_t* stop_bits)

Set UART stop bits.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **stop_bits** -- UART stop bits

返回

- ESP_OK Success
- ESP_FAIL Fail

esp_err_t **uart_get_stop_bits**(*uart_port_t* uart_num, *uart_stop_bits_t* *stop_bits)

Get the UART stop bit configuration.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **stop_bits** -- Pointer to accept value of UART stop bits.

返回

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*stop_bit)

esp_err_t **uart_set_parity**(*uart_port_t* uart_num, *uart_parity_t* parity_mode)

Set UART parity mode.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **parity_mode** -- the enum of uart parity configuration

返回

- ESP_FAIL Parameter error
- ESP_OK Success

esp_err_t **uart_get_parity**(*uart_port_t* uart_num, *uart_parity_t* *parity_mode)

Get the UART parity mode configuration.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **parity_mode** -- Pointer to accept value of UART parity mode.

返回

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*parity_mode)

esp_err_t **uart_get_sclk_freq**(*uart_sclk_t* sclk, uint32_t *out_freq_hz)

Get the frequency of a clock source for the HP UART port.

参数

- **sclk** -- Clock source
- **out_freq_hz** -- **[out]** Output of frequency, in Hz

返回

- ESP_ERR_INVALID_ARG: if the clock source is not supported
- otherwise ESP_OK

esp_err_t **uart_set_baudrate**(*uart_port_t* uart_num, uint32_t baudrate)

Set UART baud rate.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **baudrate** -- UART baud rate.

返回

- ESP_FAIL Parameter error
- ESP_OK Success

esp_err_t **uart_get_baudrate**(*uart_port_t* uart_num, uint32_t *baudrate)

Get the UART baud rate configuration.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **baudrate** -- Pointer to accept value of UART baud rate

返回

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*baudrate)

esp_err_t **uart_set_line_inverse**(*uart_port_t* uart_num, uint32_t inverse_mask)

Set UART line inverse mode.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

- **inverse_mask** -- Choose the wires that need to be inverted. Using the ORred mask of `uart_signal_inv_t`

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_set_hw_flow_ctrl**(*uart_port_t* uart_num, *uart_hw_flowcontrol_t* flow_ctrl, uint8_t rx_thresh)

Set hardware flow control.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **flow_ctrl** -- Hardware flow control mode
- **rx_thresh** -- Threshold of Hardware RX flow control (0 ~ UART_HW_FIFO_LEN(uart_num)). Only when UART_HW_FLOWCTRL_RTS is set, will the rx_thresh value be set.

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_set_sw_flow_ctrl**(*uart_port_t* uart_num, bool enable, uint8_t rx_thresh_xon, uint8_t rx_thresh_xoff)

Set software flow control.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1)
- **enable** -- switch on or off
- **rx_thresh_xon** -- low water mark
- **rx_thresh_xoff** -- high water mark

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_get_hw_flow_ctrl**(*uart_port_t* uart_num, *uart_hw_flowcontrol_t* *flow_ctrl)

Get the UART hardware flow control configuration.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **flow_ctrl** -- Option for different flow control mode.

返回

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*flow_ctrl)

esp_err_t **uart_clear_intr_status**(*uart_port_t* uart_num, uint32_t clr_mask)

Clear UART interrupt status.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **clr_mask** -- Bit mask of the interrupt status to be cleared.

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_enable_intr_mask**(*uart_port_t* uart_num, uint32_t enable_mask)

Set UART interrupt enable.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **enable_mask** -- Bit mask of the enable bits.

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_disable_intr_mask**(*uart_port_t* uart_num, uint32_t disable_mask)

Clear UART interrupt enable bits.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **disable_mask** -- Bit mask of the disable bits.

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_enable_rx_intr**(*uart_port_t* uart_num)

Enable UART RX interrupt (RX_FULL & RX_TIMEOUT INTERRUPT)

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_disable_rx_intr**(*uart_port_t* uart_num)

Disable UART RX interrupt (RX_FULL & RX_TIMEOUT INTERRUPT)

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_disable_tx_intr**(*uart_port_t* uart_num)

Disable UART TX interrupt (TX_FULL & TX_TIMEOUT INTERRUPT)

参数 **uart_num** -- UART port number

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_enable_tx_intr**(*uart_port_t* uart_num, int enable, int thresh)

Enable UART TX interrupt (TX_FULL & TX_TIMEOUT INTERRUPT)

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **enable** -- 1: enable; 0: disable
- **thresh** -- Threshold of TX interrupt, 0 ~ UART_HW_FIFO_LEN(uart_num)

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_set_pin**(*uart_port_t* uart_num, int tx_io_num, int rx_io_num, int rts_io_num, int cts_io_num)

Assign signals of a UART peripheral to GPIO pins.

备注: If the GPIO number configured for a UART signal matches one of the IOMUX signals for that GPIO, the signal will be connected directly via the IOMUX. Otherwise the GPIO and signal will be connected via the GPIO Matrix. For example, if on an ESP32 the call `uart_set_pin(0, 1, 3, -1, -1)` is performed, as GPIO1 is UART0's default TX pin and GPIO3 is UART0's default RX pin, both will be connected to respectively U0TXD and U0RXD through the IOMUX, totally bypassing the GPIO matrix. The check is performed on a per-pin basis. Thus, it is possible to have RX pin binded to a GPIO through the GPIO matrix, whereas TX is binded to its GPIO through the IOMUX.

备注: Internal signal can be output to multiple GPIO pads. Only one GPIO pad can connect with input signal.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **tx_io_num** -- UART TX pin GPIO number.
- **rx_io_num** -- UART RX pin GPIO number.
- **rts_io_num** -- UART RTS pin GPIO number.
- **cts_io_num** -- UART CTS pin GPIO number.

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_set_rts**(*uart_port_t* uart_num, int level)

Manually set the UART RTS pin level.

备注: UART must be configured with hardware flow control disabled.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **level** -- 1: RTS output low (active); 0: RTS output high (block)

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_set_dtr**(*uart_port_t* uart_num, int level)

Manually set the UART DTR pin level.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **level** -- 1: DTR output low; 0: DTR output high

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_set_tx_idle_num**(*uart_port_t* uart_num, uint16_t idle_num)

Set UART idle interval after tx FIFO is empty.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **idle_num** -- idle interval after tx FIFO is empty(unit: the time it takes to send one bit under current baudrate)

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_param_config**(*uart_port_t* uart_num, const *uart_config_t* *uart_config)

Set UART configuration parameters.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **uart_config** -- UART parameter settings

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_intr_config**(*uart_port_t* uart_num, const *uart_intr_config_t* *intr_conf)

Configure UART interrupts.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **intr_conf** -- UART interrupt settings

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_wait_tx_done**(*uart_port_t* uart_num, TickType_t ticks_to_wait)

Wait until UART TX FIFO is empty.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **ticks_to_wait** -- Timeout, count in RTOS ticks

返回

- ESP_OK Success
- ESP_FAIL Parameter error
- ESP_ERR_TIMEOUT Timeout

int **uart_tx_chars**(*uart_port_t* uart_num, const char *buffer, uint32_t len)

Send data to the UART port from a given buffer and length.

This function will not wait for enough space in TX FIFO. It will just fill the available TX FIFO and return when the FIFO is full.

备注: This function should only be used when UART TX buffer is not enabled.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **buffer** -- data buffer address
- **len** -- data length to send

返回

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

int **uart_write_bytes**(*uart_port_t* uart_num, const void *src, size_t size)

Send data to the UART port from a given buffer and length,.

If the UART driver's parameter 'tx_buffer_size' is set to zero: This function will not return until all the data have been sent out, or at least pushed into TX FIFO.

Otherwise, if the 'tx_buffer_size' > 0, this function will return after copying all the data to tx ring buffer, UART ISR will then move data from the ring buffer to TX FIFO gradually.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **src** -- data buffer address
- **size** -- data length to send

返回

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

int **uart_write_bytes_with_break**(*uart_port_t* uart_num, const void *src, size_t size, int brk_len)

Send data to the UART port from a given buffer and length,.

If the UART driver's parameter 'tx_buffer_size' is set to zero: This function will not return until all the data and the break signal have been sent out. After all data is sent out, send a break signal.

Otherwise, if the 'tx_buffer_size' > 0, this function will return after copying all the data to tx ring buffer, UART ISR will then move data from the ring buffer to TX FIFO gradually. After all data sent out, send a break signal.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **src** -- data buffer address
- **size** -- data length to send

- **brk_len** -- break signal duration(unit: the time it takes to send one bit at current baudrate)

返回

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

int **uart_read_bytes**(*uart_port_t* uart_num, void *buf, uint32_t length, TickType_t ticks_to_wait)

UART read bytes from UART buffer.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **buf** -- pointer to the buffer.
- **length** -- data length
- **ticks_to_wait** -- sTimeout, count in RTOS ticks

返回

- (-1) Error
- OTHERS (>=0) The number of bytes read from UART buffer

esp_err_t **uart_flush**(*uart_port_t* uart_num)

Alias of uart_flush_input. UART ring buffer flush. This will discard all data in the UART RX buffer.

备注: Instead of waiting the data sent out, this function will clear UART rx buffer. In order to send all the data in tx FIFO, we can use uart_wait_tx_done function.

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_flush_input**(*uart_port_t* uart_num)

Clear input buffer, discard all the data is in the ring-buffer.

备注: In order to send all the data in tx FIFO, we can use uart_wait_tx_done function.

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_get_buffered_data_len**(*uart_port_t* uart_num, size_t *size)

UART get RX ring buffer cached data length.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **size** -- Pointer of size_t to accept cached data length

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_get_tx_buffer_free_size**(*uart_port_t* uart_num, size_t *size)

UART get TX ring buffer free space size.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **size** -- Pointer of size_t to accept the free space size

返回

- ESP_OK Success

- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **uart_disable_pattern_det_intr**(*uart_port_t* uart_num)

UART disable pattern detect function. Designed for applications like 'AT commands'. When the hardware detects a series of one same character, the interrupt will be triggered.

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- ESP_OK Success
- ESP_FAIL Parameter error

esp_err_t **uart_enable_pattern_det_baud_intr**(*uart_port_t* uart_num, char pattern_chr, uint8_t chr_num, int chr_tout, int post_idle, int pre_idle)

UART enable pattern detect function. Designed for applications like 'AT commands'. When the hardware detect a series of one same character, the interrupt will be triggered.

参数

- **uart_num** -- UART port number.
- **pattern_chr** -- character of the pattern.
- **chr_num** -- number of the character, 8bit value.
- **chr_tout** -- timeout of the interval between each pattern characters, 16bit value, unit is the baud-rate cycle you configured. When the duration is more than this value, it will not take this data as at_cmd char.
- **post_idle** -- idle time after the last pattern character, 16bit value, unit is the baud-rate cycle you configured. When the duration is less than this value, it will not take the previous data as the last at_cmd char
- **pre_idle** -- idle time before the first pattern character, 16bit value, unit is the baud-rate cycle you configured. When the duration is less than this value, it will not take this data as the first at_cmd char.

返回

- ESP_OK Success
- ESP_FAIL Parameter error

int **uart_pattern_pop_pos**(*uart_port_t* uart_num)

Return the nearest detected pattern position in buffer. The positions of the detected pattern are saved in a queue, this function will dequeue the first pattern position and move the pointer to next pattern position.

The following APIs will modify the pattern position info: uart_flush_input, uart_read_bytes, uart_driver_delete, uart_pop_pattern_pos It is the application's responsibility to ensure atomic access to the pattern queue and the rx data buffer when using pattern detect feature.

备注: If the RX buffer is full and flow control is not enabled, the detected pattern may not be found in the rx buffer due to overflow.

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- (-1) No pattern found for current index or parameter error
- others the pattern position in rx buffer.

int **uart_pattern_get_pos**(*uart_port_t* uart_num)

Return the nearest detected pattern position in buffer. The positions of the detected pattern are saved in a queue, This function do nothing to the queue.

The following APIs will modify the pattern position info: uart_flush_input, uart_read_bytes, uart_driver_delete, uart_pop_pattern_pos It is the application's responsibility to ensure atomic access to the pattern queue and the rx data buffer when using pattern detect feature.

备注： If the RX buffer is full and flow control is not enabled, the detected pattern may not be found in the rx buffer due to overflow.

参数 **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).

返回

- (-1) No pattern found for current index or parameter error
- others the pattern position in rx buffer.

esp_err_t **uart_pattern_queue_reset**(*uart_port_t* uart_num, int queue_length)

Allocate a new memory with the given length to save record the detected pattern position in rx buffer.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1).
- **queue_length** -- Max queue length for the detected pattern. If the queue length is not large enough, some pattern positions might be lost. Set this value to the maximum number of patterns that could be saved in data buffer at the same time.

返回

- ESP_ERR_NO_MEM No enough memory
- ESP_ERR_INVALID_STATE Driver not installed
- ESP_FAIL Parameter error
- ESP_OK Success

esp_err_t **uart_set_mode**(*uart_port_t* uart_num, *uart_mode_t* mode)

UART set communication mode.

备注： This function must be executed after uart_driver_install(), when the driver object is initialized.

参数

- **uart_num** -- Uart number to configure, the max port number is (UART_NUM_MAX -1).
- **mode** -- UART UART mode to set

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **uart_set_rx_full_threshold**(*uart_port_t* uart_num, int threshold)

Set uart threshold value for RX fifo full.

备注： If application is using higher baudrate and it is observed that bytes in hardware RX fifo are overwritten then this threshold can be reduced

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1)
- **threshold** -- Threshold value above which RX fifo full interrupt is generated

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver is not installed

esp_err_t **uart_set_tx_empty_threshold**(*uart_port_t* uart_num, int threshold)

Set uart threshold values for TX fifo empty.

参数

- **uart_num** -- UART port number, the max port number is (UART_NUM_MAX -1)

- **threshold** -- Threshold value below which TX fifo empty interrupt is generated

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver is not installed

esp_err_t **uart_set_rx_timeout**(*uart_port_t* uart_num, const uint8_t tout_thresh)

UART set threshold timeout for TOUT feature.

参数

- **uart_num** -- Uart number to configure, the max port number is (UART_NUM_MAX -1).
- **tout_thresh** -- This parameter defines timeout threshold in uart symbol periods. The maximum value of threshold is 126. tout_thresh = 1, defines TOUT interrupt timeout equal to transmission time of one symbol (~11 bit) on current baudrate. If the time is expired the UART_RXFIFO_TOUT_INT interrupt is triggered. If tout_thresh == 0, the TOUT feature is disabled.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver is not installed

esp_err_t **uart_get_collision_flag**(*uart_port_t* uart_num, bool *collision_flag)

Returns collision detection flag for RS485 mode Function returns the collision detection flag into variable pointed by collision_flag. *collision_flag = true, if collision detected else it is equal to false. This function should be executed when actual transmission is completed (after uart_write_bytes()).

参数

- **uart_num** -- Uart number to configure the max port number is (UART_NUM_MAX -1).
- **collision_flag** -- Pointer to variable of type bool to return collision flag.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **uart_set_wakeup_threshold**(*uart_port_t* uart_num, int wakeup_threshold)

Set the number of RX pin signal edges for light sleep wakeup.

UART can be used to wake up the system from light sleep. This feature works by counting the number of positive edges on RX pin and comparing the count to the threshold. When the count exceeds the threshold, system is woken up from light sleep. This function allows setting the threshold value.

Stop bit and parity bits (if enabled) also contribute to the number of edges. For example, letter 'a' with ASCII code 97 is encoded as 0100001101 on the wire (with 8n1 configuration), start and stop bits included. This sequence has 3 positive edges (transitions from 0 to 1). Therefore, to wake up the system when 'a' is sent, set wakeup_threshold=3.

The character that triggers wakeup is not received by UART (i.e. it can not be obtained from UART FIFO). Depending on the baud rate, a few characters after that will also not be received. Note that when the chip enters and exits light sleep mode, APB frequency will be changing. To ensure that UART has correct Baud rate all the time, it is necessary to select a source clock which has a fixed frequency and remains active during sleep. For the supported clock sources of the chips, please refer to `uart_sclk_t` or `soc_periph_uart_clk_src_legacy_t`

备注: in ESP32, the wakeup signal can only be input via IO_MUX (i.e. GPIO3 should be configured as function_1 to wake up UART0, GPIO9 should be configured as function_5 to wake up UART1), UART2 does not support light sleep wakeup feature.

参数

- **uart_num** -- UART number, the max port number is (UART_NUM_MAX -1).

- **wakeup_threshold** -- number of RX edges for light sleep wakeup, value is 3 .. 0x3ff.

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if uart_num is incorrect or wakeup_threshold is outside of [3, 0x3ff] range.

esp_err_t **uart_get_wakeup_threshold**(*uart_port_t* uart_num, int *out_wakeup_threshold)

Get the number of RX pin signal edges for light sleep wakeup.

See description of uart_set_wakeup_threshold for the explanation of UART wakeup feature.

参数

- **uart_num** -- UART number, the max port number is (UART_NUM_MAX -1).
- **out_wakeup_threshold** -- **[out]** output, set to the current value of wakeup threshold for the given UART.

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if out_wakeup_threshold is NULL

esp_err_t **uart_wait_tx_idle_polling**(*uart_port_t* uart_num)

Wait until UART tx memory empty and the last char send ok (polling mode).

-

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Driver not installed

参数 **uart_num** -- UART number

esp_err_t **uart_set_loop_back**(*uart_port_t* uart_num, bool loop_back_en)

Configure TX signal loop back to RX module, just for the test usage.

-

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Driver not installed

参数

- **uart_num** -- UART number
- **loop_back_en** -- Set true to enable the loop back function, else set it false.

void **uart_set_always_rx_timeout**(*uart_port_t* uart_num, bool always_rx_timeout_en)

Configure behavior of UART RX timeout interrupt.

When always_rx_timeout is true, timeout interrupt is triggered even if FIFO is full. This function can cause extra timeout interrupts triggered only to send the timeout event. Call this function only if you want to ensure timeout interrupt will always happen after a byte stream.

参数

- **uart_num** -- UART number
- **always_rx_timeout_en** -- Set to false enable the default behavior of timeout interrupt, set it to true to always trigger timeout interrupt.

Structures

struct **uart_config_t**

UART configuration parameters for uart_param_config function.

Public Members

int **baud_rate**

UART baud rate

uart_word_length_t **data_bits**

UART byte size

uart_parity_t **parity**

UART parity mode

uart_stop_bits_t **stop_bits**

UART stop bits

uart_hw_flowcontrol_t **flow_ctrl**

UART HW flow control mode (cts/rts)

uint8_t **rx_flow_ctrl_thresh**

UART HW RTS threshold

uart_sclk_t **source_clk**

UART source clock selection

lp_uart_sclk_t **lp_source_clk**

LP_UART source clock selection

uint32_t **backup_before_sleep**

If set, the driver will backup/restore the HP UART registers before entering/after exiting sleep mode. By this approach, the system can power off HP UART's power domain. This can save power, but at the expense of more RAM being consumed

struct *uart_config_t*::[anonymous] **flags**

Configuration flags

struct **uart_intr_config_t**

UART interrupt configuration parameters for uart_intr_config function.

Public Members

uint32_t **intr_enable_mask**

UART interrupt enable mask, choose from UART_XXXX_INT_ENA_M under UART_INT_ENA_REG(i), connect with bit-or operator

uint8_t **rx_timeout_thresh**

UART timeout interrupt threshold (unit: time of sending one byte)

uint8_t **txfifo_empty_intr_thresh**

UART TX empty interrupt threshold.

uint8_t **rxfifo_full_thresh**

UART RX full interrupt threshold.

struct **uart_event_t**

Event structure used in UART event queue.

Public Members

uart_event_type_t **type**

UART event type

size_t **size**

UART data size for UART_DATA event

bool **timeout_flag**

UART data read timeout flag for UART_DATA event (no new data received during configured RX TOUT) If the event is caused by FIFO-full interrupt, then there will be no event with the timeout flag before the next byte coming.

Macros

UART_PIN_NO_CHANGE

UART_FIFO_LEN

Length of the HP UART HW FIFO.

UART_HW_FIFO_LEN(uart_num)

Length of the UART HW FIFO.

UART_BITRATE_MAX

Maximum configurable bitrate.

Type Definitions

typedef *intr_handle_t* **uart_isr_handle_t**

Enumerations

enum **uart_event_type_t**

UART event types used in the ring buffer.

Values:

enumerator **UART_DATA**

UART data event

enumerator **UART_BREAK**

UART break event

enumerator **UART_BUFFER_FULL**

UART RX buffer full event

enumerator **UART_FIFO_OVF**

UART FIFO overflow event

enumerator **UART_FRAME_ERR**

UART RX frame error event

enumerator **UART_PARITY_ERR**

UART RX parity event

enumerator **UART_DATA_BREAK**

UART TX data and break event

enumerator **UART_PATTERN_DET**

UART pattern detected

enumerator **UART_WAKEUP**

UART wakeup event

enumerator **UART_EVENT_MAX**

UART event max index

Header File

- components/hal/include/hal/uart_types.h
- This header file can be included with:

```
#include "hal/uart_types.h"
```

Structures

struct **uart_at_cmd_t**

UART AT cmd char configuration parameters Note that this function may different on different chip. Please refer to the TRM at confirguration.

Public Members

uint8_t **cmd_char**

UART AT cmd char

uint8_t **char_num**

AT cmd char repeat number

uint32_t **gap_tout**

gap time(in baud-rate) between AT cmd char

uint32_t **pre_idle**

the idle time(in baud-rate) between the non AT char and first AT char

uint32_t **post_idle**

the idle time(in baud-rate) between the last AT char and the none AT char

struct **uart_sw_flowctrl_t**

UART software flow control configuration parameters.

Public Members

uint8_t **xon_char**

Xon flow control char

uint8_t **xoff_char**

Xoff flow control char

uint8_t **xon_thrd**

If the software flow control is enabled and the data amount in rxfifo is less than xon_thrd, an xon_char will be sent

uint8_t **xoff_thrd**

If the software flow control is enabled and the data amount in rxfifo is more than xoff_thrd, an xoff_char will be sent

Type Definitions

typedef *soc_periph_uart_clk_src_legacy_t* **uart_sclk_t**

UART source clock.

typedef *soc_periph_lp_uart_clk_src_t* **lp_uart_sclk_t**

LP_UART source clock.

Enumerations

enum **uart_port_t**

UART port number, can be UART_NUM_0 ~ (UART_NUM_MAX -1).

Values:

enumerator **UART_NUM_0**

UART port 0

enumerator **UART_NUM_1**
UART port 1

enumerator **UART_NUM_2**
UART port 2

enumerator **UART_NUM_3**
UART port 3

enumerator **UART_NUM_4**
UART port 4

enumerator **LP_UART_NUM_0**
LP UART port 0

enumerator **UART_NUM_MAX**
UART port max

enum **uart_mode_t**
UART mode selection.

Values:

enumerator **UART_MODE_UART**
mode: regular UART mode

enumerator **UART_MODE_RS485_HALF_DUPLEX**
mode: half duplex RS485 UART mode control by RTS pin

enumerator **UART_MODE_IRDA**
mode: IRDA UART mode

enumerator **UART_MODE_RS485_COLLISION_DETECT**
mode: RS485 collision detection UART mode (used for test purposes)

enumerator **UART_MODE_RS485_APP_CTRL**
mode: application control RS485 UART mode (used for test purposes)

enum **uart_word_length_t**
UART word length constants.

Values:

enumerator **UART_DATA_5_BITS**
word length: 5bits

enumerator **UART_DATA_6_BITS**
word length: 6bits

enumerator **UART_DATA_7_BITS**
word length: 7bits

enumerator **UART_DATA_8_BITS**

word length: 8bits

enumerator **UART_DATA_BITS_MAX**

enum **uart_stop_bits_t**

UART stop bits number.

Values:

enumerator **UART_STOP_BITS_1**

stop bit: 1bit

enumerator **UART_STOP_BITS_1_5**

stop bit: 1.5bits

enumerator **UART_STOP_BITS_2**

stop bit: 2bits

enumerator **UART_STOP_BITS_MAX**

enum **uart_parity_t**

UART parity constants.

Values:

enumerator **UART_PARITY_DISABLE**

Disable UART parity

enumerator **UART_PARITY_EVEN**

Enable UART even parity

enumerator **UART_PARITY_ODD**

Enable UART odd parity

enum **uart_hw_flowcontrol_t**

UART hardware flow control modes.

Values:

enumerator **UART_HW_FLOWCTRL_DISABLE**

disable hardware flow control

enumerator **UART_HW_FLOWCTRL_RTS**

enable RX hardware flow control (rts)

enumerator **UART_HW_FLOWCTRL_CTS**

enable TX hardware flow control (cts)

enumerator **UART_HW_FLOWCTRL_CTS_RTS**

enable hardware flow control

enumerator **UART_HW_FLOWCTRL_MAX**

enum **uart_signal_inv_t**

UART signal bit map.

Values:

enumerator **UART_SIGNAL_INV_DISABLE**

Disable UART signal inverse

enumerator **UART_SIGNAL_IRDA_TX_INV**

inverse the UART irda_tx signal

enumerator **UART_SIGNAL_IRDA_RX_INV**

inverse the UART irda_rx signal

enumerator **UART_SIGNAL_RXD_INV**

inverse the UART rxd signal

enumerator **UART_SIGNAL_CTS_INV**

inverse the UART cts signal

enumerator **UART_SIGNAL_DSR_INV**

inverse the UART dsr signal

enumerator **UART_SIGNAL_TXD_INV**

inverse the UART txd signal

enumerator **UART_SIGNAL_RTS_INV**

inverse the UART rts signal

enumerator **UART_SIGNAL_DTR_INV**

inverse the UART dtr signal

GPIO 查找宏指令 UART 外设有供直接连接的专用 IO_MUX 管脚，但也可用非直接的 GPIO 矩阵将信号配置到其他管脚。如要直接连接，需要知道哪一管脚为 UART 通道的专用 IO_MUX 管脚。GPIO 查找宏简化了查找和分配 IO_MUX 管脚的过程，可根据 IO_MUX 管脚编号或所需 UART 通道名称选择一个宏，该宏将返回匹配的对应项。请查看下列示例。

备注： 如需较高的 UART 波特率（超过 40 MHz），即仅使用 IO_MUX 管脚时，可以使用此类宏。在其他情况下可以忽略这些宏，并使用 GPIO 矩阵为 UART 功能配置任一 GPIO 管脚。

---

1. `UART_NUM_2_TXD_DIRECT_GPIO_NUM` 返回 UART 通道 2 TXD 管脚的 IO_MUX 管脚编号（管脚 17）
2. `UART_GPIO19_DIRECT_CHANNEL` 在通过 IO_MUX 连接到 UART 外设时返回 GPIO 19 的 UART 编号（即 UART_NUM_0）
3. GPIO 19 在通过 IO_MUX 用作 UART CTS 管脚时，`UART_CTS_GPIO19_DIRECT_CHANNEL` 将返回 GPIO 19 的 UART 编号（即 UART_NUM_0）。该宏类似于上述宏，但指定了管脚功能，这也是 IO_MUX 分配的一部分。

Header File

- components/soc/esp32p4/include/soc/uart_channel.h
- This header file can be included with:

```
#include "soc/uart_channel.h"
```

Macros

UART_GPIO37_DIRECT_CHANNEL

UART_NUM_0_TXD_DIRECT_GPIO_NUM

UART_GPIO38_DIRECT_CHANNEL

UART_NUM_0_RXD_DIRECT_GPIO_NUM

UART_GPIO8_DIRECT_CHANNEL

UART_NUM_0_RTS_DIRECT_GPIO_NUM

UART_GPIO9_DIRECT_CHANNEL

UART_NUM_0_CTS_DIRECT_GPIO_NUM

UART_TXD_GPIO37_DIRECT_CHANNEL

UART_RXD_GPIO38_DIRECT_CHANNEL

UART_RTS_GPIO8_DIRECT_CHANNEL

UART_CTS_GPIO9_DIRECT_CHANNEL

UART_GPIO10_DIRECT_CHANNEL

UART_NUM_1_TXD_DIRECT_GPIO_NUM

UART_GPIO11_DIRECT_CHANNEL

UART_NUM_1_RXD_DIRECT_GPIO_NUM

UART_GPIO12_DIRECT_CHANNEL

UART_NUM_1_RTS_DIRECT_GPIO_NUM

UART_GPIO13_DIRECT_CHANNEL

UART_NUM_1_CTS_DIRECT_GPIO_NUM

UART_TXD_GPIO10_DIRECT_CHANNEL

UART_RXD_GPIO11_DIRECT_CHANNEL

UART_RTS_GPIO12_DIRECT_CHANNEL

UART_CTS_GPIO13_DIRECT_CHANNEL

2.5.37 USB 设备栈

概述

USB 设备栈（以下简称设备栈）支持在 ESP32-P4 上启用 USB 设备支持。通过使用设备栈，可以为 ESP32-P4 烧录任意具有明确定义的 USB 设备功能（如键盘、鼠标、摄像头）、自定义功能（也称特定供应商类别）或上述功能的组合（也称复合设备）。

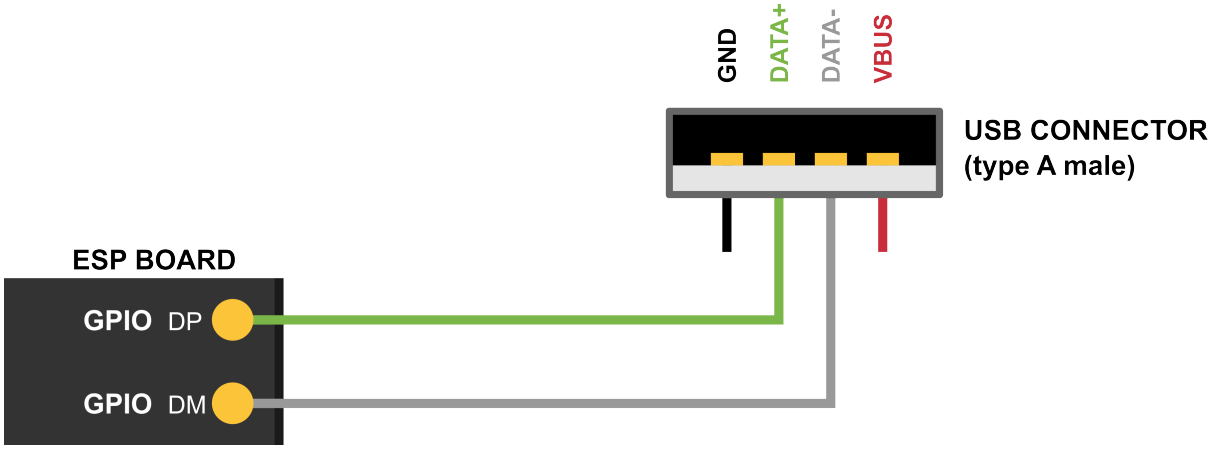
设备栈基于 TinyUSB 栈构建，但对 TinyUSB 进行了一些小的功能扩展和修改，使其更好地集成到 ESP-IDF。设备栈通过 ESP-IDF 组件注册器 作为托管组件分发。

功能列表

- 支持多种设备类别 (CDC, HID, MIDI, MSC)
- 支持复合设备
- 支持特定供应商类别
- 最多支持 15 个端点
  - 8 个输入/输出端点
  - 7 个输入端点
- 自供电设备的 VBUS 监测

硬件连接

ESP32-P4 将 USB D+ 和 D- 信号路由到其专用引脚。为了实现 USB 设备功能，这些引脚应通过某种方式连接到总线（例如，通过 Micro-B 端口、USB-C 端口或直接连接到标准-A 插头）。



备注： 自供电设备还必须通过电压分压器或比较器连接 VBUS，详情请参阅自供电设备。

设备栈结构

设备栈以 TinyUSB 栈为基础，在此基础上，该设备栈实现了以下功能：

- 自定义 USB 描述符
- 支持串行设备
- 通过串行设备重定向标准流
- 提供用于 USB 设备 MSC 类的存储介质（SPI-Flash 和 SD 卡）
- 封装设备栈中处理 TinyUSB 服务的任务

组件依赖项

设备栈通过 ESP-IDF 组件注册器 分发，使用前，请使用以下命令将设备栈组件添加为依赖项：

```
idf.py add-dependency esp_tinyusb
```

配置选项 通过 menuconfig 选项，可以对设备栈进行以下多方面配置：

- TinyUSB 日志的详细程度
- 设备栈任务相关选项
- 默认设备/字符串描述符选项
- 特定类别的选项

配置描述符 结构体 `tinyusb_config_t` 提供了以下与 USB 描述符相关的字段，应进行初始化：

- `device_descriptor`
- `configuration_descriptor`
- `string_descriptor`

全速 USB 设备应初始化以下字段，以提供相应的配置描述符：

- `configuration_descriptor`

高速 USB 设备应初始化以下字段，以提供不同速度下的配置描述符：

- `fs_configuration_descriptor`
- `hs_configuration_descriptor`
- `qualifier_descriptor`

备注： 为符合 USB 2.0 协议规范，需同时初始化 `fs_configuration_descriptor` 和 `hs_configuration_descriptor`。

调用 `tinyusb_driver_install()` 时，设备栈将基于上述字段中提供的描述符实现 USB 设备。

设备栈还提供了默认描述符，将 `tinyusb_driver_install()` 中的相应字段设置为 `NULL` 即可安装。默认描述符包括：

- 默认设备描述符：启用时，将 `device_descriptor` 设置为 `NULL`。默认设备描述符将使用相应的 menuconfig 选项设置的值（如 PID、VID、bcdDevice 等）。
- 默认字符串描述符：启用时，将 `string_descriptor` 设置为 `NULL`。默认字符串描述符将使用相应的 menuconfig 选项设置的值（如制造商、产品和序列字符串描述符选项）。
- 默认配置描述符。某些很少需要自定义配置的类别（如 CDC 和 MSC）将提供默认配置描述符。启用时，将 `configuration_descriptor` 设置为 `NULL`。

安装设备栈

请调用 `tinyusb_driver_install()` 安装设备栈。结构体 `tinyusb_config_t` 指定了设备栈的配置，而 `tinyusb_config_t` 作为参数传递给 `tinyusb_driver_install()`。

备注： 结构体 `tinyusb_config_t` 可以实现零初始化（如 `const tinyusb_config_t tusb_cfg = { 0 };`）或部分初始化（如下所示）。对于结构体中任何初始化为 `0` 或 `NULL` 的成员，设备栈将使用其默认配置，请参阅如下示例。

```
const tinyusb_config_t partial_init = {
    .device_descriptor = NULL,  // 使用在 menuconfig 中指定的默认设备描述符
    .string_descriptor = NULL,  // 使用在 menuconfig 中指定的默认字符串描述符
    .external_phy = false,      // 使用内部 USB PHY
    .configuration_descriptor = NULL, // 使用在 menuconfig
↪中根据设置指定的默认配置描述符
};
```

自供电设备

USB 规范要求自供电设备监测 USB 的 VBUS 信号的电压水平。与总线供电设备相反，即使没有 USB 连接，自供电设备也可以正常工作。通过监测 VBUS 电压水平，自供电设备可以检测连接和断开事件。当 VBUS 电压升高到 4.75 V 以上时视为有效；当 VBUS 电压下降到 4.35 V 以下时视为无效。

在 ESP32-P4 上，需要使用一个 GPIO 作为电压感测管脚，检测 VBUS 处于在规定阈值之上/之下。然而，由于 ESP32-P4 管脚具有 3.3 V 容差，即使 VBUS 上升/下降到高于/低于上述规定阈值，ESP32-P4 仍会显示为逻辑高电平。因此，为了检测 VBUS 是否有效，可以采用以下方法：

- 将 VBUS 连接至电压比较器芯片/电路，该芯片/电路可检测上述阈值（即 4.35 V 和 4.75 V），并向 ESP32-P4 输出 3.3 V 逻辑电平，指示 VBUS 是否有效。
- 如果 VBUS 为 4.4 V，则使用电阻分压器输出 (0.75 x Vdd)（见下图）。

备注： 在这两种情况下，设备从 USB 主机拔出后 3 毫秒内，传感引脚上的电压必须为逻辑低电平。

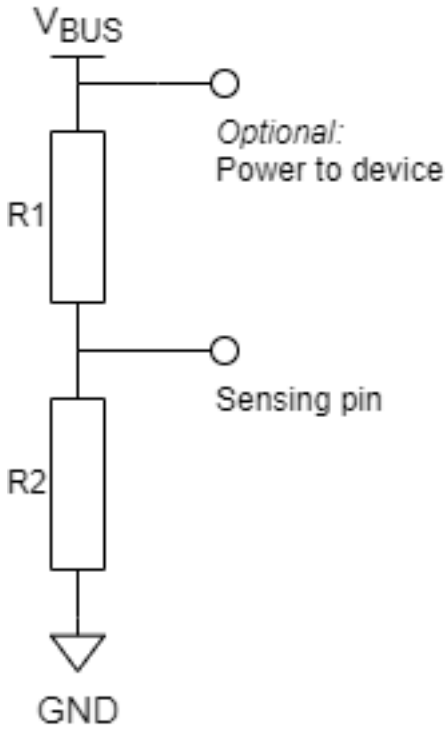


图 44: 用于 VBUS 监测的简易分压器

请在结构体 `tinyusb_config_t` 中将 `self_powered` 设置为 `true`，并将 `vbus_monitor_io` 设置

为用于 VBUS 监测的 GPIO 管脚编号以使用此功能。

USB 串行设备 (CDC-ACM)

如果在 menuconfig 中启用了 CDC 选项，则可以根据 `tinyusb_config_cdcacm_t` 的设置，使用 `tusb_cdc_acm_init()` 初始化 USB 串行设备，请参阅如下示例：

```
const tinyusb_config_cdcacm_t acm_cfg = {
    .usb_dev = TINYUSB_USBDEV_0,
    .cdc_port = TINYUSB_CDC_ACM_0,
    .rx_unread_buf_sz = 64,
    .callback_rx = NULL,
    .callback_rx_wanted_char = NULL,
    .callback_line_state_changed = NULL,
    .callback_line_coding_changed = NULL
};
tusb_cdc_acm_init(&acm_cfg);
```

可以在配置结构体中设置指向 `tusb_cdcacm_callback_t` 函数的指针指定回调函数，或在初始化 USB 串行设备后，调用 `tinyusb_cdcacm_register_callback()` 指定回调函数。

USB 串行控制台 USB 串行设备支持将所有标准输入/输出流 (stdin、stdout、stderr) 重定向到 USB。因此，调用如 `printf()` 等标准库输入/输出函数将导致通过 USB 而不是 UART 发送/接收数据。

建议调用 `esp_tusb_init_console()` 将标准输入/输出流切换到 USB，并调用 `esp_tusb_deinit_console()` 将其切换回 UART。

USB 大容量存储设备 (MSC)

在 menuconfig 中启用 MSC `CONFIG_TINYUSB_MSC_ENABLED` 选项时，可以将 ESP 芯片作为 USB 大容量存储设备使用。按如下示例，可以初始化存储媒介（SPI-Flash 或 SD 卡）。

- SPI-Flash

```
static esp_err_t storage_init_spiflash(wl_handle_t *wl_handle)
{
    ***
    esp_partition_t *data_partition = esp_partition_find_first(ESP_PARTITION_TYPE_
↪DATA, ESP_PARTITION_SUBTYPE_DATA_FAT, NULL);
    ***
    wl_mount(data_partition, wl_handle);
    ***
}
storage_init_spiflash(&wl_handle);

const tinyusb_msc_spiflash_config_t config_spi = {
    .wl_handle = wl_handle
};
tinyusb_msc_storage_init_spiflash(&config_spi);
```

- SD 卡

```
static esp_err_t storage_init_sdmmc(sdmmc_card_t **card)
{
    ***
    sdmmc_host_t host = SDMMC_HOST_DEFAULT();
    sdmmc_slot_config_t slot_config = SDMMC_SLOT_CONFIG_DEFAULT();
    // 对于 SD 卡，设置要使用的总线宽度
```

(下页继续)

(续上页)

```
    slot_config.width = 4;
    slot_config.clk = CONFIG_EXAMPLE_PIN_CLK;
    slot_config.cmd = CONFIG_EXAMPLE_PIN_CMD;
    slot_config.d0 = CONFIG_EXAMPLE_PIN_D0;
    slot_config.d1 = CONFIG_EXAMPLE_PIN_D1;
    slot_config.d2 = CONFIG_EXAMPLE_PIN_D2;
    slot_config.d3 = CONFIG_EXAMPLE_PIN_D3;
    slot_config.flags |= SDMMC_SLOT_FLAG_INTERNAL_PULLUP;

    sd_card = (sdmmc_card_t *)malloc(sizeof(sdmmc_card_t));
    (*host.init)();
    sdmmc_host_init_slot(host.slot, (const sdmmc_slot_config_t *) &slot_config);
    sdmmc_card_init(&host, sd_card);
    ***
}
storage_init_sdmmc(&card);

const tinyusb_msc_sdmmc_config_t config_sdmmc = {
    .card = card
};
tinyusb_msc_storage_init_sdmmc(&config_sdmmc);
```

应用示例

下表列出了 peripherals/usb/device 目录下的代码示例：

代码示例	描述
peripherals/usb/device/tusb_console	设置 ESP32-P4 芯片，通过串行设备连接获取日志输出
peripherals/usb/device/tusb_serial_device	设置 ESP32-P4 芯片，将其作为 USB 串行设备使用
peripherals/usb/device/tusb_midi	设置 ESP32-P4 芯片，将其作为 USB MIDI 设备使用
peripherals/usb/device/tusb_hid	设置 ESP32-P4 芯片，将其作为 USB 人机界面设备使用
peripherals/usb/device/tusb_msc	设置 ESP32-P4 芯片，将其作为 USB 大容量存储设备使用
peripherals/usb/device/tusb_composite_msc_serialdevice	设置 ESP32-P4 芯片，将其作为复合 USB 设备使用 (MSC + CDC)

2.5.38 USB 主机

本文档提供了 USB 主机库的相关信息，按以下章节展开：

章节

- *USB* 主机
  - 概述
  - 架构
  - 用法
  - 示例
  - 主机栈配置
  - *API* 参考
  - 维护注意事项

概述

USB 主机库（以下简称主机库）是 USB 主机栈的最底层，提供面向公众开放的 API。应用程序使用 USB 主机功能时，通常无需与主机库直接交互，而是使用某个主机 Class 驱动提供的 API，这些主机 Class 驱动构建在主机库之上。

然而，由于以下的某些原因（但不仅限于此），有时你可能需要直接使用主机库：

- 需要实现自定义主机 Class 驱动程序
- 需要更低级别的抽象

特性和限制 主机库具有以下特性：

- 支持高速 (HS)、全速 (FS) 和低速 (LS) 设备。
- 支持四种传输类型，即控制传输、块传输、中断传输和同步传输。
- 支持高带宽等时性端点。
- 支持多个 Class 驱动程序同时运行，即主机的多个客户端同时运行。
- 单个设备可以由多个客户端同时使用，如复合设备。
- 主机库及其底层主机栈不会在内部自动创建操作系统任务，任务数量完全由主机库接口的使用方式决定。一般来说，任务数量为 `（运行中的主机 Class 驱动程序数量 + 1）`。

目前，主机库及其底层主机栈存在以下限制：

- 仅支持单个设备，而主机库的 API 支持多设备。
- 仅支持异步传输。
- 仅支持使用发现的首个配置，尚不支持变更为其他配置。
- 尚不支持传输超时。
- ESP32-P4 包含两个 USB-OTG 外设：USB 2.0 OTG 高速和 USB 2.0 OTG 全速。目前仅支持高速实例。
- ESP32-P4 目前无法枚举低速设备。

架构

上图展示了使用 USB 主机功能时涉及的关键实体，包括：

- **主机库**
- 主机库的 **客户端**
- **设备**
- 主机库的 **守护进程任务**

主机库 主机库是 ESP-IDF USB 主机栈中面向公众开放的最底层的 API 层。任何其他 ESP-IDF 组件（如 Class 驱动程序或用户组件），如果需要与连接的 USB 设备通信，只能直接或间接使用主机库 API。

主机库 API 分为两类，即 **库 API** 和 **客户端 API**。

- 客户端 API 负责主机库的客户端与一或多个 USB 设备间的通信，该 API 只能由主机库的注册客户端调用。
- 库 API 负责主机库处理的通信中不特定于单个客户端的通信，如设备枚举。该 API 通常由主机库的守护进程任务调用。

客户端 主机库的客户端指使用主机库与 USB 设备通信的软件组件，如主机 Class 驱动程序或用户组件。每个客户端通常与任务间存在一对一关系，这表明，对特定客户端而言，其所有客户端 API 都应该在同一任务的上下文中调用。

通过将使用主机库的软件组件进行分类，划分为独立的客户端，主机库可以将所有客户端特定事件的处理委托给客户端对应的任务。换句话说，每个客户端任务负责管理与其对应的客户端之间的所有 USB 通信操作和事件处理，无需关心其他客户端的事件。

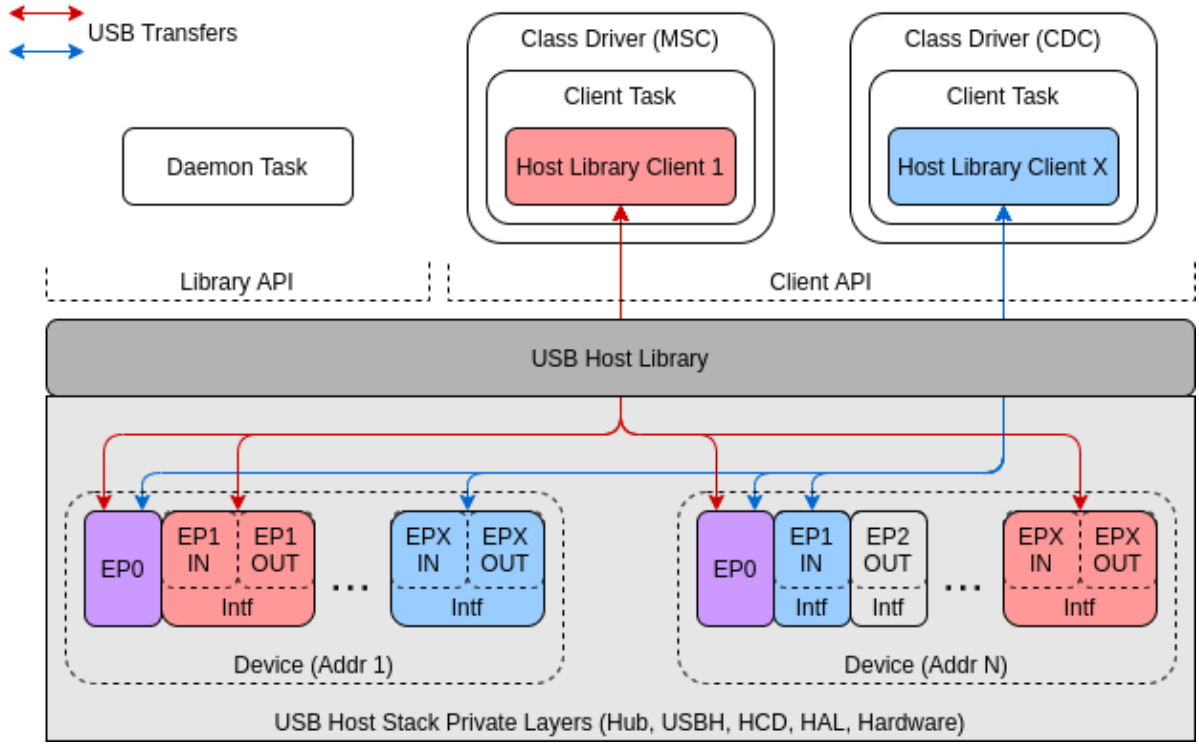


图 45: USB 主机功能涉及的关键实体

守护进程任务 尽管主机库将客户端事件的处理委托给客户端本身，但仍然需要处理主机库事件，即不特定于客户端的事件。主机库事件处理可能涉及以下内容：

- 处理 USB 设备的连接、枚举和断连
- 将控制传输从/向客户端进行重定向
- 将事件转发给客户端

因此，除客户端任务外，主机库也需要一个任务来处理所有的库事件，这个任务通常是主机库守护进程任务。

设备 主机库隔离了客户端与设备处理的细节，包括连接、内存分配和枚举等，客户端只需提供已连接且已枚举的设备列表供选择。默认情况下，在枚举过程中，每个设备都会自动配置为使用找到的第一个配置，即通过获取配置描述符请求返回的第一个配置描述符。对于大多数标准设备，通常将第一个配置的 `bConfigurationValue` 设置为 `1`。启用选项*CONFIG_USB_HOST_ENABLE_ENUM_FILTER_CALLBACK* 后，可以选择不同的 `bConfigurationValue`。获取更多详细信息，请参阅 *Multiple configuration Support*_。

只要不与相同接口通信，两个及以上的客户端可以同时与同一设备通信。然而，多个客户端同时与相同设备的默认端点（即 EP0）通信，将导致它们的控制传输序列化。

要与设备通信，客户端必须满足以下条件：

1. 使用设备地址打开设备，告知主机库，客户端正在使用该设备。
2. 获取将用于通信的接口，防止其他客户端获取相同的接口。
3. 通过已经获取了使用权的设备接口的端点通信通道发送数据传输。客户端的任务负责处理其与 USB 设备通信相关的操作和事件。

用法

主机库及底层主机栈不会创建任何任务，客户端任务和守护进程任务等均需由 Class 驱动程序或用户自行创建。然而，主机库提供了两个事件处理函数，可以处理所有必要的主机库操作，这些函数应从客户端任务和守护进程任务中重复调用。因此，客户端任务和守护进程任务的使用将主要集中于调用这些事件处理函数。

主机库与守护进程任务

基本用法 主机库 API 提供了*usb_host_lib_handle_events()*，可以处理库事件。该函数需要反复调用，通常是从守护进程任务中调用。函数*usb_host_lib_handle_events()* 有以下特点：

- 该函数会持续阻塞，直至需要处理库事件。
- 每次调用该函数都会返回事件标志，有助于了解卸载主机库的时机。

最基础的守护进程任务通常类似以下代码片段：

```
#include "usb/usb_host.h"

void daemon_task(void *arg)
{
    ...
    bool exit = false;
    while (!exit) {
        uint32_t event_flags;
        usb_host_lib_handle_events(portMAX_DELAY, &event_flags);
        if (event_flags & USB_HOST_LIB_EVENT_FLAGS_NO_CLIENTS) {
            ...
        }
        if (event_flags & USB_HOST_LIB_EVENT_FLAGS_ALL_FREE) {
            ...
        }
        ...
    }
    ...
}
```

备注： 了解守护进程任务的完整示例，请前往 peripherals/usb/host/usb_host_lib。

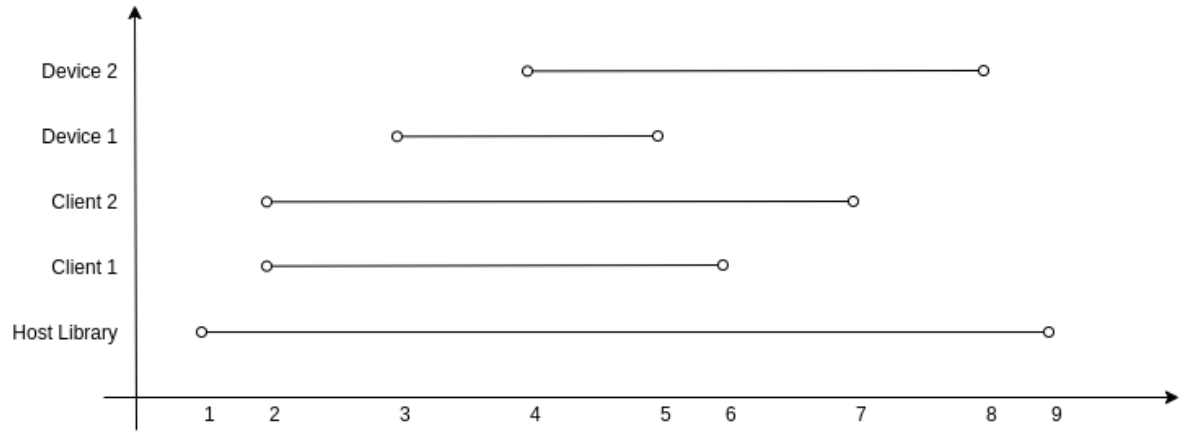


图 46: USB 主机库典型生命周期

生命周期 上图展示了 USB 主机库的典型生命周期，其中涉及多个客户端和设备。具体而言，示例涉及以下内容：

- 两个已注册的客户端（客户端 1 和客户端 2）。
- 两个已连接的设备（设备 1 和设备 2），其中客户端 1 与设备 1 通信，客户端 2 与设备 2 通信。

参考上图可知，典型 USB 主机库生命周期包括以下关键阶段：

1. **调用*usb_host_install()*，安装主机库**。

- 调用任意主机库 API 前，请确保已完成主机库安装。
- 调用 *usb_host_install()* 的位置（如从守护进程任务或其他任务中调用）取决于守护系统任务、客户端任务和系统其余部分间的同步逻辑。

2. **安装完主机库后，调用 *usb_host_client_register()* 注册客户端。**
   - 该注册函数通常从客户端任务调用，而客户端任务需等待来自守护进程任务的信号。
   - 调用过 *usb_host_install()* 后，如有需要，也可以在其他地方调用该注册函数。
3. **设备 1 连接，并进行枚举。**
   - 每个已注册的客户端（本案例中为客户端 1 和客户端 2）都会通过 *USB_HOST_CLIENT_EVENT_NEW_DEV* 事件得到新设备的通知。
   - 客户端 1 开启设备 1，并与之通信。
4. **设备 2 连接，并进行枚举。**
   - 客户端 1 和 2 通过 *USB_HOST_CLIENT_EVENT_NEW_DEV* 事件得到新设备的通知。
   - 客户端 2 开启设备 2，并与之通信。
5. **设备 1 突然断开连接。**
   - 客户端 1 通过 *USB_HOST_CLIENT_EVENT_DEV_GONE* 得到通知，并开始清理，关闭和释放客户端 1 与设备 1 之间的关联资源。
   - 客户端 2 不会收到通知，因为它并未开启设备 1。
6. **客户端 1 完成清理，调用 *usb_host_client_deregister()* 注销客户端。**
   - 该注销函数通常在任务退出前，从客户端任务中调用。
   - 如有需要，只要客户端 1 已完成清理，也可以在其他地方调用该注销函数。
7. **客户端 2 完成与设备 2 的通信，随后关闭设备 2，并自行注销。**
   - 由于客户端 2 是最后一个注销的客户端，通过 *USB_HOST_LIB_EVENT_FLAGS_NO_CLIENTS* 事件标志，守护进程任务可以得知，所有客户端已注销。
   - 设备 2 未释放，仍会进行分配，因为虽然没有任何客户端打开设备 2，但它仍处于连接状态。
8. **所有客户端注销后，守护进程任务开始清理。**
   - 守护进程任务需要先调用 *usb_host_device_free_all()*，释放设备 2。
   - 如果 *usb_host_device_free_all()* 能够成功释放所有设备，函数将返回 *ESP_OK*，表明已释放所有设备。
   - 如果 *usb_host_device_free_all()* 无法成功释放所有设备，例如因为设备仍由某个客户端开启，函数将返回 *ESP_ERR_NOT_FINISHED*。
   - 守护进程任务必须等待 *usb_host_lib_handle_events()* 返回事件标志 *USB_HOST_LIB_EVENT_FLAGS_ALL_FREE*，方知何时所有设备均已释放。
9. 一旦守护进程任务确认所有客户端均已注销，且所有设备均已释放，便可调用 *usb_host_uninstall()*，卸载主机库。

客户端与 Class 驱动程序

基本用法 主机库 API 提供函数 *usb_host_client_handle_events()*，可以处理特定客户端事件。该函数需要反复调用，通常是从客户端任务中调用。函数 *usb_host_client_handle_events()* 有以下特点：

- 该函数可以持续阻塞，直至需要处理客户端事件。
- 该函数的主要目的是发生客户端事件时，调用多个事件处理回调函数。

以下回调函数均从 *usb_host_client_handle_events()* 中调用，因此客户端任务能够获取事件通知。

- 类型为 *usb_host_client_event_cb_t* 的客户端事件回调函数会将客户端事件消息传递给客户端，提示添加、移除设备等事件。
- 类型为 *usb_transfer_cb_t* 的 USB 传输完成回调函数表明，先前由客户端提交的特定 USB 传输已完成。

备注： 考虑到上述回调函数从 *usb_host_client_handle_events()* 中调用，应避免在回调函数内部阻塞，否则将导致 *usb_host_client_handle_events()* 阻塞，阻止其他待处理的客户端事件得到处理。

以下代码片段展示了一个基础的主机 Class 驱动程序及其客户端任务，代码片段中包括：

- 一个简单的客户端任务函数 client_task，它会在循环中调用*usb_host_client_handle_events()*。
- 使用客户端事件回调函数和传输完成回调函数。
- 一个用于 Class 驱动程序的简单状态机。该 Class 驱动程序仅支持打开设备，发送 OUT 传输到 EP1，然后关闭设备。

```
#include <string.h>
#include "usb/usb_host.h"

#define CLASS_DRIVER_ACTION_OPEN_DEV    0x01
#define CLASS_DRIVER_ACTION_TRANSFER    0x02
#define CLASS_DRIVER_ACTION_CLOSE_DEV   0x03

struct class_driver_control {
    uint32_t actions;
    uint8_t dev_addr;
    usb_host_client_handle_t client_hdl;
    usb_device_handle_t dev_hdl;
};

static void client_event_cb(const usb_host_client_event_msg_t *event_msg, void␣
↪*arg)
{
    //该函数从 usb_host_client_handle_events() 中调用，请勿在此阻塞，并尽量保持简洁
    struct class_driver_control *class_driver_obj = (struct class_driver_control␣
↪*)arg;
    switch (event_msg->event) {
        case USB_HOST_CLIENT_EVENT_NEW_DEV:
            class_driver_obj->actions |= CLASS_DRIVER_ACTION_OPEN_DEV;
            class_driver_obj->dev_addr = event_msg->new_dev.address; //
↪存储新设备的地址
            break;
        case USB_HOST_CLIENT_EVENT_DEV_GONE:
            class_driver_obj->actions |= CLASS_DRIVER_ACTION_CLOSE_DEV;
            break;
        default:
            break;
    }
}

static void transfer_cb(usb_transfer_t *transfer)
{
    //该函数从 usb_host_client_handle_events() 中调用，请勿在此阻塞，并尽量保持简洁
    struct class_driver_control *class_driver_obj = (struct class_driver_control␣
↪*)transfer->context;
    printf("Transfer status %d, actual number of bytes transferred %d\n", transfer-
↪>status, transfer->actual_num_bytes);
    class_driver_obj->actions |= CLASS_DRIVER_ACTION_CLOSE_DEV;
}

void client_task(void *arg)
{
    ... //等待主机库安装
    //初始化 Class 驱动程序对象
    struct class_driver_control class_driver_obj = {0};
    //注册客户端
    usb_host_client_config_t client_config = {
        .is_synchronous = false,
        .max_num_event_msg = 5,
        .async = {
            .client_event_callback = client_event_cb,
```

(下页继续)

(续上页)

```
            .callback_arg = &class_driver_obj,
        }
    };
    usb_host_client_register(&client_config, &class_driver_obj.client_hdl);
    //分配一个 USB 传输
    usb_transfer_t *transfer;
    usb_host_transfer_alloc(1024, 0, &transfer);

    //事件处理循环
    bool exit = false;
    while (!exit) {
        //调用客户端事件处理函数
        usb_host_client_handle_events(class_driver_obj.client_hdl, portMAX_DELAY);
        //执行待处理的 Class 驱动程序操作
        if (class_driver_obj.actions & CLASS_DRIVER_ACTION_OPEN_DEV) {
            //开启设备，声明接口 1
            usb_host_device_open(class_driver_obj.client_hdl, class_driver_obj.dev_
↪addr, &class_driver_obj.dev_hdl);
            usb_host_interface_claim(class_driver_obj.client_hdl, class_driver_obj.
↪dev_hdl, 1, 0);
        }
        if (class_driver_obj.actions & CLASS_DRIVER_ACTION_TRANSFER) {
            //发送一个 OUT 传输到 EP1
            memset(transfer->data_buffer, 0xAA, 1024);
            transfer->num_bytes = 1024;
            transfer->device_handle = class_driver_obj.dev_hdl;
            transfer->bEndpointAddress = 0x01;
            transfer->callback = transfer_cb;
            transfer->context = (void *)&class_driver_obj;
            usb_host_transfer_submit(transfer);
        }
        if (class_driver_obj.actions & CLASS_DRIVER_ACTION_CLOSE_DEV) {
            //释放接口，关闭设备
            usb_host_interface_release(class_driver_obj.client_hdl, class_driver_
↪obj.dev_hdl, 1);
            usb_host_device_close(class_driver_obj.client_hdl, class_driver_obj.
↪dev_hdl);
            exit = true;
        }
        ... //处理其他 Class 驱动程序要求的行为
    }

    //清理 Class 驱动程序
    usb_host_transfer_free(transfer);
    usb_host_client_deregister(class_driver_obj.client_hdl);
    ... //删除客户端任务。如有需要，向守护进程任务发送信号。
}
```

备注： 在实际应用中，主机 Class 驱动程序还能支持更多功能，因此也存在更复杂的状态机。主机 Class 驱动程序可能需要：

- 能够开启多个设备
- 解析已开启设备的描述符，确定设备是否是目标 Class
- 按特定顺序，与接口的多个端点通信
- 声明设备的多个接口
- 处理各种错误情况

生命周期 客户端任务与 Class 驱动程序的典型生命周期包括以下关键阶段：

1. 等待与主机库有关的信号完成安装。
2. 通过*usb_host_client_register()* 注册客户端，并分配其他的 Class 驱动程序资源，如使用*usb_host_transfer_alloc()* 分配传输。
3. 对于 Class 驱动程序需要与之通信的各个新设备：
   a. 调用*usb_host_device_addr_list_fill()*，检查设备是否已连接。
   b. 如果设备尚未连接，则等待客户端事件回调函数的*USB_HOST_CLIENT_EVENT_NEW_DEV* 事件。
   c. 调用*usb_host_device_open()* 开启设备。
   d. 分别调用*usb_host_get_device_descriptor()* 和*usb_host_get_active_config_descriptor()* 解析设备和配置描述符。
   e. 调用*usb_host_interface_claim()*，声明设备的必要接口。
4. 调用*usb_host_transfer_submit()* 或*usb_host_transfer_submit_control()*，向设备提交传输。
5. 一旦*USB_HOST_CLIENT_EVENT_DEV_GONE* 事件表示，Class 驱动程序不再需要已打开的设备，或者设备断开连接：
   a. 在这些端点上调用*usb_host_endpoint_halt()* 和*usb_host_endpoint_flush()*，停止先前提交的传输。
   b. 调用*usb_host_interface_release()*，释放先前声明的所有接口。
   c. 调用*usb_host_device_close()*，关闭设备。
6. 调用*usb_host_client_deregister()* 注销客户端，并释放其他 Class 驱动程序资源。
7. 删除客户端任务。如有需要，向守护进程任务发送信号。

示例

主机库示例 peripherals/usb/host/usb_host_lib 展示了 USB 主机库 API 的基本用法，用于创建伪 Class 驱动程序。

Class 驱动程序示例 USB 主机栈提供了大量示例，展示了如何通过使用主机库 API 创建主机 Class 驱动程序。

CDC-ACM

- 通信设备 Class（抽象控制模型）的主机 Class 驱动程序通过 ESP-IDF 组件注册器 作为受管理的组件分发。
- 示例 peripherals/usb/host/cdc/cdc_acm_host 使用 CDC-ACM 主机驱动程序组件，与 CDC-ACM 设备通信。
- 示例 peripherals/usb/host/cdc/cdc_acm_vcp 展示了如何扩展 CDC-ACM 主机驱动程序，与虚拟串口设备交互。
- 示例 esp_modem 中也使用了 CDC-ACM 驱动程序，该程序在这些示例中与蜂窝模块通信。

MSC

- 大容量存储 Class（仅支持批量传输）的主机 Class 驱动程序已部署到 ESP-IDF 组件注册器。
- 示例 peripherals/usb/host/msc 展示了如何使用 MSC 主机驱动程序读写 USB flash 驱动。

HID

- HID（人机接口设备）的主机 class 驱动作为托管组件通过 ESP-IDF 组件注册器 分发。
- 示例 peripherals/usb/host/hid 展示了从具有多个接口的 USB HID 设备接收报告的可能性。

UVC

- USB 视频设备 Class 的主机 Class 驱动程序作为托管组件通过 ESP-IDF 组件注册器 分发。
- 示例 peripherals/usb/host/uvc 展示了如何使用 UVC 主机驱动程序接收来自 USB 摄像头的视频流，并可选择将该流通过 Wi-Fi 转发。

主机栈配置

非兼容设备支持 为了支持某些非兼容或具有特定行为的 USB 设备，可以对 USB 主机栈进行配置。

USB 设备可能是热插拔的，因此必须配置电源开关和设备连接之间的延迟，以及设备内部电源稳定后的延迟。

枚举配置 在枚举已连接 USB 设备的过程中，需要给一些事件配置合适的间隔时间以确保设备正常运行。

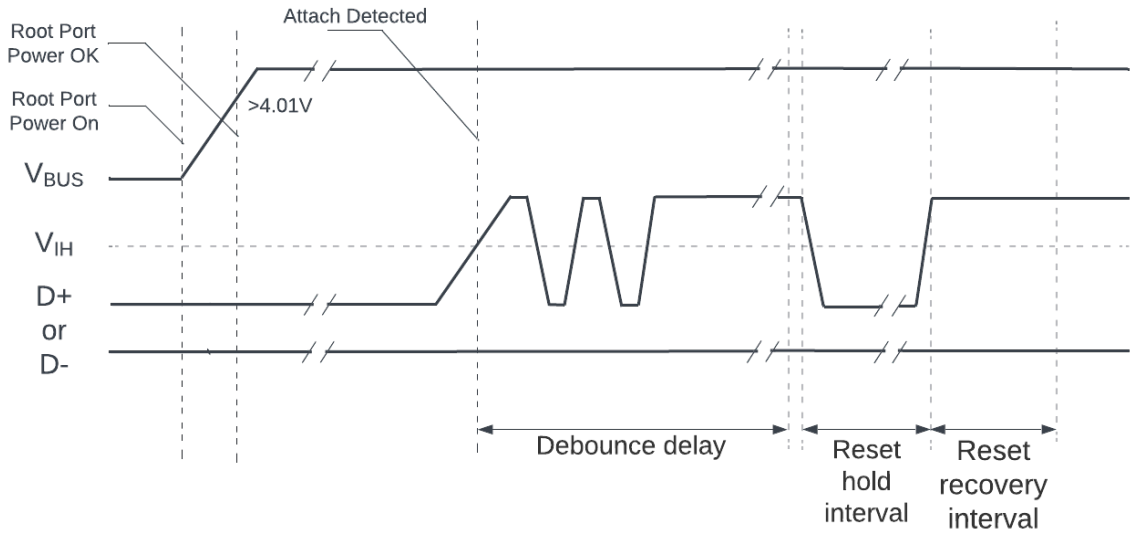


图 47: USB 根集线器上电和连接事件时序

上图展示了与连接设备时开启端口电源和热插拔设备相关的所有间隔时间。

- 端口复位或恢复运行后，USB 系统软件应提供 10 毫秒的恢复时间，此后连接到端口的设备才会响应数据传输。
- 恢复时间结束后，如果设备收到 `SetAddress()` 请求，设备必须能够完成对该请求的处理，并能在 50 毫秒内成功完成请求的状态 (Status) 阶段。
- 状态阶段结束后，设备允许有 2 毫秒的 `SetAddress()` 恢复时间。

备注： 有关连接事件时序的更多信息，请参阅 通用串行总线 2.0 规范 > 第 7.1.7.3 章 连接和断开信令。

可通过 Menuconfig 选项设置 USB 主机栈的可配置参数。

- *CONFIG_USB_HOST_DEBOUNCE_DELAY_MS* 用于配置防抖延迟。
- *CONFIG_USB_HOST_RESET_HOLD_MS* 用于配置重置保持时间。
- *CONFIG_USB_HOST_RESET_RECOVERY_MS* 用于配置重置恢复时间。
- *CONFIG_USB_HOST_SET_ADDR_RECOVERY_MS* 用于配置 `SetAddress()` 恢复时间。

多项配置支持 对于具有多项配置的 USB 设备，可以在设备枚举过程中指定所需的配置编号。

枚举过滤器 枚举过滤器是类型为*usb_host_enum_filter_cb_t* 的回调函数。从新连接的 USB 设备上读取设备描述符后，USB 主机栈会在枚举过程开始时调用枚举过滤器，从而为用户提供读取的设备描述符。借助此回调，用户得以：

- 选择 USB 设备的配置。
- 过滤应该进行枚举的 USB 设备。

在 menuconfig 中启用*CONFIG_USB_HOST_ENABLE_ENUM_FILTER_CALLBACK* 选项即可启用枚举过滤器。可以通过设置*usb_host_config_t::enum_filter_cb* 来指定回调函数，该函数会在调用*usb_host_install()* 时传递至主机库。

API 参考

USB 主机库的 API 包含以下头文件，但应用程序调用该 API 时只需 `#include "usb/usb_host.h"`，该头文件包含了所有 USB 主机库的头文件。

- usb/include/usb/usb_host.h 包含 USB 主机库的函数和类型。
- usb/include/usb/usb_helpers.h 包含与 USB 协议相关的各种辅助函数，如描述符解析等。
- usb/include/usb/usb_types_stack.h 包含在 USB 主机栈的多个层次中使用的类型。
- usb/include/usb/usb_types_ch9.h 包含了与 USB 2.0 规范中第 9 章相关的类型和宏，即描述符和标准请求。

Header File

- components/usb/include/usb/usb_host.h
- This header file can be included with:

```
#include "usb/usb_host.h"
```

- This header file is a part of the API provided by the `usb` component. To declare that your component depends on `usb`, add the following to your CMakeLists.txt:

```
REQUIRES usb
```

or

```
PRIV_REQUIRES usb
```

Functions

esp_err_t **usb_host_install** (const *usb_host_config_t* *config)

Install the USB Host Library.

- This function should be called only once to install the USB Host Library
- This function should be called before any other USB Host Library functions are called

---

备注： If skip_phy_setup is set in the install configuration, the user is responsible for ensuring that the underlying Host Controller is enabled and the USB PHY (internal or external) is already setup before this function is called.

参数 **config** -- **[in]** USB Host Library configuration

返回

- ESP_OK: USB Host installed successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: USB Host Library is not in correct state to be installed (eg. the library itself of one of it's drivers is already installed)
- ESP_ERR_NO_MEM: Insufficient memory

esp_err_t **usb_host_uninstall** (void)

Uninstall the USB Host Library.

- This function should be called to uninstall the USB Host Library, thereby freeing its resources
- All clients must have been deregistered before calling this function

- All devices must have been freed by calling usb_host_device_free_all() and receiving the USB_HOST_LIB_EVENT_FLAGS_ALL_FREE event flag

---

备注： If skip_phy_setup was set when the Host Library was installed, the user is responsible for disabling the underlying Host Controller and USB PHY (internal or external).

返回

- ESP_OK: USB Host uninstalled successfully
- ESP_ERR_INVALID_STATE: USB Host Library is not installed, or has unfinished actions

esp_err_t **usb_host_lib_handle_events**(TickType_t timeout_ticks, uint32_t *event_flags_ret)

Handle USB Host Library events.

- This function handles all of the USB Host Library's processing and should be called repeatedly in a loop
- Check event_flags_ret to see if an flags are set indicating particular USB Host Library events
- This function should never be called by multiple threads simultaneously

---

备注： This function can block

参数

- **timeout_ticks** -- **[in]** Timeout in ticks to wait for an event to occur
- **event_flags_ret** -- **[out]** Event flags that indicate what USB Host Library event occurred.

返回

- ESP_OK: No events to be handled
- ESP_ERR_INVALID_STATE: USB Host Library is not installed
- ESP_ERR_TIMEOUT: Semaphore waiting for events has timed out

esp_err_t **usb_host_lib_unblock**(void)

Unblock the USB Host Library handler.

- This function simply unblocks the USB Host Library event handling function (usb_host_lib_handle_events())

返回

- ESP_OK: USB Host library unblocked successfully
- ESP_ERR_INVALID_STATE: USB Host Library is not installed

esp_err_t **usb_host_lib_info**(*usb_host_lib_info_t* *info_ret)

Get current information about the USB Host Library.

参数 **info_ret** -- **[out]** USB Host Library Information

返回

- ESP_OK: USB Host Library info obtained successfully
- ESP_ERR_INVALID_STATE: USB Host Library is not installed
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_lib_set_root_port_power**(bool enable)

Power the root port ON or OFF.

- Powering ON the root port will allow device connections to occur

- Powering OFF the root port will disconnect all devices downstream off the root port and prevent any further device connections.

备注： If '*usb_host_config_t.root_port_unpowered*' was set on USB Host Library installation, users must call this function to power ON the root port before any device connections can occur.

参数 **enable** -- **[in]** True to power the root port ON, false to power OFF

返回

- ESP_OK: Root port power enabled/disabled
- ESP_ERR_INVALID_STATE: Root port already powered or HUB driver not installed

esp_err_t **usb_host_client_register**(const *usb_host_client_config_t* *client_config, *usb_host_client_handle_t* *client_hdl_ret)

Register a client of the USB Host Library.

- This function registers a client of the USB Host Library
- Once a client is registered, its processing function usb_host_client_handle_events() should be called repeatedly

参数

- **client_config** -- **[in]** Client configuration
- **client_hdl_ret** -- **[out]** Client handle

返回

- ESP_OK: USB Host Library client registered successfully
- ESP_ERR_INVALID_STATE: USB Host Library is not installed
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NO_MEM: Insufficient memory

esp_err_t **usb_host_client_deregister**(*usb_host_client_handle_t* client_hdl)

Deregister a USB Host Library client.

- This function deregisters a client of the USB Host Library
- The client must have closed all previously opened devices before attempting to deregister

参数 **client_hdl** -- **[in]** Client handle

返回

- ESP_OK: USB Host Library client deregistered successfully
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_client_handle_events**(*usb_host_client_handle_t* client_hdl, TickType_t timeout_ticks)

USB Host Library client processing function.

- This function handles all of a client's processing and should be called repeatedly in a loop
- For a particular client, this function should never be called by multiple threads simultaneously

---

备注： This function can block

参数

- **client_hdl** -- **[in]** Client handle
- **timeout_ticks** -- **[in]** Timeout in ticks to wait for an event to occur

返回

- ESP_OK: No event to be handled
- ESP_ERR_INVALID_STATE: USB Host Library is not installed
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_TIMEOUT: Semaphore waiting for events has timed out

esp_err_t **usb_host_client_unblock**(*usb_host_client_handle_t* client_hdl)

Unblock a client.

- This function simply unblocks a client if it is blocked on the usb_host_client_handle_events() function.
- This function is useful when need to unblock a client in order to deregister it.

参数 **client_hdl** -- **[in]** Client handle

返回

- ESP_OK: Client unblocked successfully
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_device_open**(*usb_host_client_handle_t* client_hdl, uint8_t dev_addr, *usb_device_handle_t* *dev_hdl_ret)

Open a device.

- This function allows a client to open a device
- A client must open a device first before attempting to use it (e.g., sending transfers, device requests etc.)

参数

- **client_hdl** -- **[in]** Client handle
- **dev_addr** -- **[in]** Device's address
- **dev_hdl_ret** -- **[out]** Device's handle

返回

- ESP_OK: Device opened successfully
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_device_close**(*usb_host_client_handle_t* client_hdl, *usb_device_handle_t* dev_hdl)

Close a device.

- This function allows a client to close a device
- A client must close a device after it has finished using the device (claimed interfaces must also be released)
- A client must close all devices it has opened before de-registering

---

备注： This function can block

参数

- **client_hdl** -- **[in]** Client handle
- **dev_hdl** -- **[in]** Device handle

返回

- ESP_OK: Device closed successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FOUND: Device address not found among opened devices
- ESP_ERR_INVALID_STATE: The client never opened the device, or the client has not released all the interfaces from the device

esp_err_t **usb_host_device_free_all**(void)

Indicate that all devices can be freed when possible.

- This function marks all devices as waiting to be freed
- If a device is not opened by any clients, it will be freed immediately
- If a device is opened by at least one client, the device will be free when the last client closes that device.
- Wait for the USB_HOST_LIB_EVENT_FLAGS_ALL_FREE flag to be set by usb_host_lib_handle_events() in order to know when all devices have been freed
- This function is useful when cleaning up devices before uninstalling the USB Host Library

返回

- ESP_OK: All devices already freed (i.e., there were no devices)
- ESP_ERR_INVALID_STATE: Client must be deregistered
- ESP_ERR_NOT_FINISHED: There are one or more devices that still need to be freed, wait for USB_HOST_LIB_EVENT_FLAGS_ALL_FREE event

esp_err_t **usb_host_device_addr_list_fill**(int list_len, uint8_t *dev_addr_list, int *num_dev_ret)

Fill a list of device address.

- This function fills an empty list with the address of connected devices
- The Device addresses can then used in usb_host_device_open()
- If there are more devices than the list_len, this function will only fill up to list_len number of devices.

参数

- **list_len** -- **[in]** Length of the empty list
- **dev_addr_list** -- **[inout]** Empty list to be filled
- **num_dev_ret** -- **[out]** Number of devices

返回

- ESP_OK: Device list filled successfully
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_device_info**(*usb_device_handle_t* dev_hdl, *usb_device_info_t* *dev_info)

Get device's information.

- This function gets some basic information of a device
- The device must be opened first before attempting to get its information

---

备注: This function can block

参数

- **dev_hdl** -- **[in]** Device handle
- **dev_info** -- **[out]** Device information

返回

- ESP_OK: Device information obtained successfully
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_get_device_descriptor**(*usb_device_handle_t* dev_hdl, const *usb_device_desc_t* **device_desc)

Get device's device descriptor.

- A client must call usb_host_device_open() first
- No control transfer is sent. The device's descriptor is cached on enumeration

- This function simple returns a pointer to the cached descriptor

备注： No control transfer is sent. The device's descriptor is cached on enumeration

参数

- **dev_hdl** -- **[in]** Device handle
- **device_desc** -- **[out]** Device descriptor

返回

- ESP_OK: Device descriptor obtained successfully
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_get_active_config_descriptor**(*usb_device_handle_t* dev_hdl, const *usb_config_desc_t* **config_desc)

Get device's active configuration descriptor.

- A client must call usb_host_device_open() first
- No control transfer is sent. The device's active configuration descriptor is cached on enumeration
- This function simple returns a pointer to the cached descriptor

备注： This function can block

备注： No control transfer is sent. A device's active configuration descriptor is cached on enumeration

参数

- **dev_hdl** -- **[in]** Device handle
- **config_desc** -- **[out]** Configuration descriptor

返回

- ESP_OK: Active configuration descriptor obtained successfully
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_get_config_desc**(*usb_host_client_handle_t* client_hdl, *usb_device_handle_t* dev_hdl, uint8_t bConfigurationValue, const *usb_config_desc_t* **config_desc_ret)

Get get device's configuration descriptor.

- The USB Host library only caches a device's active configuration descriptor.
- This function reads any configuration descriptor of a particular device (specified by bConfigurationValue).
- This function will read the specified configuration descriptor via control transfers, and allocate memory to store that descriptor.
- Users can call usb_host_free_config_desc() to free the descriptor's memory afterwards.

备注： This function can block

备注： A client must call usb_host_device_open() on the device first

备注： bConfigurationValue starts from index 1

参数

- **client_hdl** -- **[in]** Client handle - usb_host_client_handle_events() should be called repeatedly in a separate task to handle client events
- **dev_hdl** -- **[in]** Device handle
- **bConfigurationValue** -- **[in]** Index of device's configuration descriptor to be read
- **config_desc_ret** -- **[out]** Returned configuration descriptor

返回

- ESP_OK: Configuration descriptor obtained successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_SUPPORTED: Invalid bConfigurationValue value (the device does not have this configuration value)
- ESP_ERR_NO_MEM: Insufficient memory

esp_err_t **usb_host_free_config_desc**(const *usb_config_desc_t* *config_desc)

Free a configuration descriptor.

This function frees a configuration descriptor that was returned by usb_host_get_config_desc()

参数 **config_desc** -- **[out]** Configuration descriptor

返回

- ESP_OK: Configuration descriptor freed successfully
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **usb_host_interface_claim**(*usb_host_client_handle_t* client_hdl, *usb_device_handle_t* dev_hdl, uint8_t bInterfaceNumber, uint8_t bAlternateSetting)

Function for a client to claim a device's interface.

- A client must claim a device's interface before attempting to communicate with any of its endpoints
- Once an interface is claimed by a client, it cannot be claimed by any other client.

备注： This function can block

参数

- **client_hdl** -- **[in]** Client handle
- **dev_hdl** -- **[in]** Device handle
- **bInterfaceNumber** -- **[in]** Interface number
- **bAlternateSetting** -- **[in]** Interface alternate setting number

返回

- ESP_OK: Interface claimed successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: USB Host is not a correct state to claim an interface
- ESP_ERR_NOT_FOUND: Interface or EP not found
- ESP_ERR_NO_MEM: Insufficient memory

esp_err_t **usb_host_interface_release**(*usb_host_client_handle_t* client_hdl, *usb_device_handle_t* dev_hdl, uint8_t bInterfaceNumber)

Function for a client to release a previously claimed interface.

- A client should release a device's interface after it no longer needs to communicate with the interface
- A client must release all of its interfaces of a device it has claimed before being able to close the device

---

备注： This function can block

参数

- **client_hdl** -- **[in]** Client handle
- **dev_hdl** -- **[in]** Device handle
- **bInterfaceNumber** -- **[in]** Interface number

返回

- ESP_OK: Interface released successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_STATE: Client never opened the USB Device, or interface currently can not be freed
- ESP_ERR_NOT_FOUND: Interface number not found in the list of interfaces

esp_err_t **usb_host_endpoint_halt**(*usb_device_handle_t* dev_hdl, uint8_t bEndpointAddress)

Halt a particular endpoint.

- The device must have been opened by a client
- The endpoint must be part of an interface claimed by a client
- Once halted, the endpoint must be cleared using usb_host_endpoint_clear() before it can communicate again

备注： This function can block

参数

- **dev_hdl** -- **[in]** Device handle
- **bEndpointAddress** -- **[in]** Endpoint address

返回

- ESP_OK: Endpoint halted successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FOUND: Endpoint address not found
- ESP_ERR_INVALID_STATE: Endpoint pipe is not in the correct state/condition to execute a command

esp_err_t **usb_host_endpoint_flush**(*usb_device_handle_t* dev_hdl, uint8_t bEndpointAddress)

Flush a particular endpoint.

- The device must have been opened by a client
- The endpoint must be part of an interface claimed by a client
- The endpoint must have been halted (either through a transfer error, or usb_host_endpoint_halt())
- Flushing an endpoint will caused an queued up transfers to be canceled

备注： This function can block

参数

- **dev_hdl** -- **[in]** Device handle
- **bEndpointAddress** -- **[in]** Endpoint address

返回

- ESP_OK: Endpoint flushed successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FOUND: Endpoint address not found
- ESP_ERR_INVALID_STATE: Endpoint pipe is not in the correct state/condition to execute a command

esp_err_t **usb_host_endpoint_clear**(*usb_device_handle_t* dev_hdl, uint8_t bEndpointAddress)

Clear a halt on a particular endpoint.

- The device must have been opened by a client
- The endpoint must be part of an interface claimed by a client
- The endpoint must have been halted (either through a transfer error, or usb_host_endpoint_halt())
- If the endpoint has any queued up transfers, clearing a halt will resume their execution

---

备注： This function can block

参数

- **`dev_hdl`** -- **[in]** Device handle
- **`bEndpointAddress`** -- **[in]** Endpoint address

返回

- ESP_OK: Endpoint cleared successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FOUND: Endpoint address not found
- ESP_ERR_INVALID_STATE: Endpoint pipe is not in the correct state/condition to execute a command

esp_err_t **`usb_host_transfer_alloc`**(size_t data_buffer_size, int num_isoc_packets, *usb_transfer_t* **transfer)

Allocate a transfer object.

- This function allocates a transfer object
- Each transfer object has a fixed sized buffer specified on allocation
- A transfer object can be re-used indefinitely
- A transfer can be submitted using usb_host_transfer_submit() or usb_host_transfer_submit_control()

参数

- **`data_buffer_size`** -- **[in]** Size of the transfer's data buffer
- **`num_isoc_packets`** -- **[in]** Number of isochronous packets in transfer (set to 0 for non-isochronous transfers)
- **`transfer`** -- **[out]** Transfer object

返回

- ESP_OK: Transfer object allocated successfully
- ESP_ERR_NO_MEM: Insufficient memory

esp_err_t **`usb_host_transfer_free`**(*usb_transfer_t* *transfer)

Free a transfer object.

- Free a transfer object previously allocated using usb_host_transfer_alloc()
- The transfer must not be in-flight when attempting to free it
- If a NULL pointer is passed, this function will simply return ESP_OK

参数 **`transfer`** -- **[in]** Transfer object

返回

- ESP_OK: Transfer object freed successfully

esp_err_t **`usb_host_transfer_submit`**(*usb_transfer_t* *transfer)

Submit a non-control transfer.

- Submit a transfer to a particular endpoint. The device and endpoint number is specified inside the transfer
- The transfer must be properly initialized before submitting

- On completion, the transfer's callback will be called from the client's usb_host_client_handle_events() function.

参数 **transfer** -- **[in]** Initialized transfer object
返回
- ESP_OK: Transfer submitted successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FINISHED: Transfer already in-flight
- ESP_ERR_NOT_FOUND: Endpoint address not found
- ESP_ERR_INVALID_STATE: Endpoint pipe is not in a correct state to submit transfer

esp_err_t **usb_host_transfer_submit_control** (*usb_host_client_handle_t* client_hdl, *usb_transfer_t* *transfer)

Submit a control transfer.

- Submit a control transfer to a particular device. The client must have opened the device first
- The transfer must be properly initialized before submitting. The first 8 bytes of the transfer's data buffer should contain the control transfer setup packet
- On completion, the transfer's callback will be called from the client's usb_host_client_handle_events() function.

参数
- **client_hdl** -- **[in]** Client handle
- **transfer** -- **[in]** Initialized transfer object

返回
- ESP_OK: Control transfer submitted successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FINISHED: Transfer already in-flight
- ESP_ERR_NOT_FOUND: Endpoint address not found
- ESP_ERR_INVALID_STATE: Endpoint pipe is not in a correct state to submit transfer

Structures

struct **usb_host_client_event_msg_t**

Client event message.

Client event messages are sent to each client of the USB Host Library in order to notify them of various USB Host Library events such as:

- Addition of new devices
- Removal of existing devices

备注: The event message structure has a union with members corresponding to each particular event. Based on the event type, only the relevant member field should be accessed.

Public Members

usb_host_client_event_t **event**

Type of event

uint8_t **address**

New device's address

struct *usb_host_client_event_msg_t*::[anonymous]::[anonymous] **new_dev**

New device info

usb_device_handle_t **dev_hdl**

The handle of the device that was gone

struct *usb_host_client_event_msg_t*::[anonymous]::[anonymous] **dev_gone**

Gone device info

struct **usb_host_lib_info_t**

Current information about the USB Host Library obtained via usb_host_lib_info()

Public Members

int **num_devices**

Current number of connected (and enumerated) devices

int **num_clients**

Current number of registered clients

struct **usb_host_config_t**

USB Host Library configuration.

Configuration structure of the USB Host Library. Provided in the usb_host_install() function

Public Members

bool **skip_phy_setup**

If set, the USB Host Library will not configure the USB PHY thus allowing the user to manually configure the USB PHY before calling usb_host_install(). Users should set this if they want to use an external USB PHY. Otherwise, the USB Host Library will automatically configure the internal USB PHY

bool **root_port_unpowered**

If set, the USB Host Library will not power on the root port on installation. This allows users to power on the root port manually by calling usb_host_lib_set_root_port_power().

int **intr_flags**

Interrupt flags for the underlying ISR used by the USB Host stack

usb_host_enum_filter_cb_t **enum_filter_cb**

Enumeration filter callback. Enable CONFIG_USB_HOST_ENABLE_ENUM_FILTER_CALLBACK to use this feature. Set to NULL otherwise.

struct **usb_host_client_config_t**

USB Host Library Client configuration.

Configuration structure for a USB Host Library client. Provided in usb_host_client_register()

Public Members

bool **is_synchronous**

Whether the client is asynchronous or synchronous or not. Set to false for now.

int **max_num_event_msg**

Maximum number of event messages that can be stored (e.g., 3)

usb_host_client_event_cb_t **client_event_callback**

Client's event callback function

void ***callback_arg**

Event callback function argument

struct *usb_host_client_config_t*::[anonymous]::[anonymous] **async**

Async callback config

Macros

USB_HOST_LIB_EVENT_FLAGS_NO_CLIENTS

All clients have been deregistered from the USB Host Library

USB_HOST_LIB_EVENT_FLAGS_ALL_FREE

The USB Host Library has freed all devices

Type Definitions

typedef struct usb_host_client_handle_s ***usb_host_client_handle_t**

Handle to a USB Host Library asynchronous client.

An asynchronous client can be registered using usb_host_client_register()

备注： Asynchronous API

---

typedef void (***usb_host_client_event_cb_t**)(const *usb_host_client_event_msg_t* *event_msg, void *arg)

Client event callback.

- Each client of the USB Host Library must register an event callback to receive event messages from the USB Host Library.
- The client event callback is run from the context of the clients usb_host_client_handle_events() function

Enumerations

enum **usb_host_client_event_t**

The type event in a client event message.

Values:

enumerator **USB_HOST_CLIENT_EVENT_NEW_DEV**

A new device has been enumerated and added to the USB Host Library

enumerator **USB_HOST_CLIENT_EVENT_DEV_GONE**

A device opened by the client is now gone

Header File

- components/usb/include/usb/usb_helpers.h
- This header file can be included with:

```
#include "usb/usb_helpers.h"
```

- This header file is a part of the API provided by the `usb` component. To declare that your component depends on `usb`, add the following to your CMakeLists.txt:

```
REQUIRES usb
```

or

```
PRIV_REQUIRES usb
```

Functions

const *usb_standard_desc_t* ***usb_parse_next_descriptor**(const *usb_standard_desc_t* *cur_desc, uint16_t wTotalLength, int *offset)

Get the next descriptor.

Given a particular descriptor within a full configuration descriptor, get the next descriptor within the configuration descriptor. This is a convenience function that can be used to walk each individual descriptor within a full configuration descriptor.

参数

- **cur_desc** -- **[in]** Current descriptor
- **wTotalLength** -- **[in]** Total length of the configuration descriptor
- **offset** -- **[inout]** Byte offset relative to the start of the configuration descriptor. On input, it is the offset of the current descriptor. On output, it is the offset of the returned descriptor.

返回 usb_standard_desc_t* Next descriptor, NULL if end of configuration descriptor reached

const *usb_standard_desc_t* ***usb_parse_next_descriptor_of_type**(const *usb_standard_desc_t* *cur_desc, uint16_t wTotalLength, uint8_t bDescriptorType, int *offset)

Get the next descriptor of a particular type.

Given a particular descriptor within a full configuration descriptor, get the next descriptor of a particular type (i.e., using the bDescriptorType value) within the configuration descriptor.

参数

- **cur_desc** -- **[in]** Current descriptor
- **wTotalLength** -- **[in]** Total length of the configuration descriptor
- **bDescriptorType** -- **[in]** Type of the next descriptor to get
- **offset** -- **[inout]** Byte offset relative to the start of the configuration descriptor. On input, it is the offset of the current descriptor. On output, it is the offset of the returned descriptor.

返回 usb_standard_desc_t* Next descriptor, NULL if end descriptor is not found or configuration descriptor reached

int **usb_parse_interface_number_of_alternate**(const *usb_config_desc_t* *config_desc, uint8_t bInterfaceNumber)

Get the number of alternate settings for a bInterfaceNumber.

Given a particular configuration descriptor, for a particular bInterfaceNumber, get the number of alternate settings available for that interface (i.e., the max possible value of bAlternateSetting for that bInterfaceNumber).

参数

- **config_desc** -- **[in]** Pointer to the start of a full configuration descriptor
- **bInterfaceNumber** -- **[in]** Interface number

返回 int The number of alternate settings that the interface has, -1 if bInterfaceNumber not found

const *usb_intf_desc_t* ***usb_parse_interface_descriptor**(const *usb_config_desc_t* *config_desc, uint8_t bInterfaceNumber, uint8_t bAlternateSetting, int *offset)

Get a particular interface descriptor (using bInterfaceNumber and bAlternateSetting)

Given a full configuration descriptor, get a particular interface descriptor.

备 注: To get the number of alternate settings for a particular bInterfaceNumber, call usb_parse_interface_number_of_alternate()

参数

- **config_desc** -- **[in]** Pointer to the start of a full configuration descriptor
- **bInterfaceNumber** -- **[in]** Interface number
- **bAlternateSetting** -- **[in]** Alternate setting number
- **offset** -- **[out]** Byte offset of the interface descriptor relative to the start of the configuration descriptor. Can be NULL.

返回 const usb_intf_desc_t* Pointer to interface descriptor, NULL if not found.

const *usb_ep_desc_t* ***usb_parse_endpoint_descriptor_by_index**(const *usb_intf_desc_t* *intf_desc, int index, uint16_t wTotalLength, int *offset)

Get an endpoint descriptor within an interface descriptor.

Given an interface descriptor, get the Nth endpoint descriptor of the interface. The number of endpoints in an interface is indicated by the bNumEndpoints field of the interface descriptor.

备注: If bNumEndpoints is 0, it means the interface uses the default endpoint only

---

参数

- **intf_desc** -- **[in]** Pointer to thee start of an interface descriptor
- **index** -- **[in]** Endpoint index
- **wTotalLength** -- **[in]** Total length of the containing configuration descriptor
- **offset** -- **[inout]** Byte offset relative to the start of the configuration descriptor. On input, it is the offset of the interface descriptor. On output, it is the offset of the endpoint descriptor.

返回 const usb_ep_desc_t* Pointer to endpoint descriptor, NULL if not found.

const *usb_ep_desc_t* ***usb_parse_endpoint_descriptor_by_address**(const *usb_config_desc_t* *config_desc, uint8_t bInterfaceNumber, uint8_t bAlternateSetting, uint8_t bEndpointAddress, int *offset)

Get an endpoint descriptor based on an endpoint's address.

Given a configuration descriptor, get an endpoint descriptor based on it's bEndpointAddress, bAlternateSetting, and bInterfaceNumber.

参数

- **config_desc** -- **[in]** Pointer to the start of a full configuration descriptor
- **bInterfaceNumber** -- **[in]** Interface number
- **bAlternateSetting** -- **[in]** Alternate setting number

- **bEndpointAddress** -- **[in]** Endpoint address
- **offset** -- **[out]** Byte offset of the endpoint descriptor relative to the start of the configuration descriptor. Can be NULL

返回 const usb_ep_desc_t* Pointer to endpoint descriptor, NULL if not found.

void **usb_print_device_descriptor**(const *usb_device_desc_t* *devc_desc)

Print device descriptor.

参数 **devc_desc** -- **[in]** Device descriptor

void **usb_print_config_descriptor**(const *usb_config_desc_t* *cfg_desc, *print_class_descriptor_cb* class_specific_cb)

Print configuration descriptor.

- This function prints the full contents of a configuration descriptor (including interface and endpoint descriptors)
- When a non-standard descriptor is encountered, this function will call the class_specific_cb if it is provided

参数

- **cfg_desc** -- **[in]** Configuration descriptor
- **class_specific_cb** -- **[in]** Class specific descriptor callback. Can be NULL

void **usb_print_string_descriptor**(const *usb_str_desc_t* *str_desc)

Print a string descriptor.

This function will only print ASCII characters of the UTF-16 encoded string

参数 **str_desc** -- **[in]** String descriptor

static inline int **usb_round_up_to_mps**(int num_bytes, int mps)

Round up to an integer multiple of endpoint's MPS.

This is a convenience function to round up a size/length to an endpoint's MPS (Maximum packet size). This is useful when calculating transfer or buffer lengths of IN endpoints.

- If MPS <= 0, this function will return 0
- If num_bytes <= 0, this function will return 0

参数

- **num_bytes** -- **[in]** Number of bytes
- **mps** -- **[in]** MPS

返回 int Round up integer multiple of MPS

Type Definitions

typedef void (***print_class_descriptor_cb**)(const *usb_standard_desc_t**)

Print class specific descriptor callback.

Optional callback to be provided to usb_print_config_descriptor() function. The callback is called when when a non-standard descriptor is encountered. The callback should decode the descriptor as print it.

Header File

- components/usb/include/usb/usb_types_stack.h
- This header file can be included with:

```
#include "usb/usb_types_stack.h"
```

- This header file is a part of the API provided by the `usb` component. To declare that your component depends on `usb`, add the following to your CMakeLists.txt:

```
REQUIRES usb
```

or

```
PRIV_REQUIRES usb
```

Structures

struct **usb_parent_dev_info_t**

Parent device information.

Public Members

usb_device_handle_t **dev_hdl**

Device's parent handle

uint8_t **port_num**

Device's parent port number

struct **usb_device_info_t**

Basic information of an enumerated device.

Public Members

usb_parent_dev_info_t **parent**

Device's parent information

usb_speed_t **speed**

Device's speed

uint8_t **dev_addr**

Device's address

uint8_t **bMaxPacketSize0**

The maximum packet size of the device's default endpoint

uint8_t **bConfigurationValue**

Device's current configuration number

const *usb_str_desc_t* ***str_desc_manufacturer**

Pointer to Manufacturer string descriptor (can be NULL)

const *usb_str_desc_t* ***str_desc_product**

Pointer to Product string descriptor (can be NULL)

const *usb_str_desc_t* ***str_desc_serial_num**

Pointer to Serial Number string descriptor (can be NULL)

struct **usb_isoc_packet_desc_t**

Isochronous packet descriptor.

If the number of bytes in an Isochronous transfer is larger than the MPS of the endpoint, the transfer is split into multiple packets transmitted at the endpoint's specified interval. An array of Isochronous packet descriptors describes how an Isochronous transfer should be split into multiple packets.

Public Members

int **num_bytes**

Number of bytes to transmit/receive in the packet. IN packets should be integer multiple of MPS

int **actual_num_bytes**

Actual number of bytes transmitted/received in the packet

usb_transfer_status_t **status**

Status of the packet

struct **usb_transfer_s**

USB transfer structure.

Public Members

uint8_t *const **data_buffer**

Pointer to data buffer

const size_t **data_buffer_size**

Size of the data buffer in bytes

int **num_bytes**

Number of bytes to transfer. Control transfers should include the size of the setup packet. Isochronous transfer should be the total transfer size of all packets. For non-control IN transfers, num_bytes should be an integer multiple of MPS.

int **actual_num_bytes**

Actual number of bytes transferred

uint32_t **flags**

Transfer flags

usb_device_handle_t **device_handle**

Device handle

uint8_t **bEndpointAddress**

Endpoint Address

usb_transfer_status_t **status**

Status of the transfer

uint32_t **timeout_ms**

Timeout (in milliseconds) of the packet (currently not supported yet)

usb_transfer_cb_t **callback**

Transfer callback

void ***context**

Context variable for transfer to associate transfer with something

const int **num_isoc_packets**

Only relevant to Isochronous. Number of service periods (i.e., intervals) to transfer data buffer over.

usb_isoc_packet_desc_t **isoc_packet_desc**[]

Descriptors for each Isochronous packet

Macros

USB_TRANSFER_FLAG_ZERO_PACK

Terminate Bulk/Interrupt OUT transfer with a zero length packet.

OUT transfers normally terminate when the Host has transferred the exact amount of data it needs to the device. However, for bulk and interrupt OUT transfers, if the transfer size just happened to be a multiple of MPS, it will be impossible to know the boundary between two consecutive transfers to the same endpoint.

Therefore, this flag will cause the transfer to automatically add a zero length packet (ZLP) at the end of the transfer if the following conditions are met:

- The target endpoint is a Bulk/Interrupt OUT endpoint (Host to device)
- The transfer's length (i.e., transfer.num_bytes) is a multiple of the endpoint's MPS

Otherwise, this flag has no effect.

Users should check whether their target device's class requires a ZLP, as not all Bulk/Interrupt OUT endpoints require them. For example:

- For MSC Bulk Only Transport class, the Host MUST NEVER send a ZLP. Bulk transfer boundaries are determined by the CBW and CSW instead
- For CDC Ethernet, the Host MUST ALWAYS send a ZLP if a segment (i.e., a transfer) is a multiple of MPS (See 3.3.1 Segment Delineation)

备注： See USB2.0 specification 5.7.3 and 5.8.3 for more details

---

备注： IN transfers normally terminate when the Host as receive the exact amount of data it needs (must be multiple of MPS) or the endpoint sends a short packet to the Host (For bulk OUT only). Indicates that a bulk OUT transfers should always terminate with a short packet, even if it means adding an extra zero length packet

---

Type Definitions

typedef struct usb_device_handle_s ***usb_device_handle_t**

Handle of a USB Device connected to a USB Host.

typedef bool (***usb_host_enum_filter_cb_t**)(const *usb_device_desc_t* *dev_desc, uint8_t *bConfigurationValue)

Enumeration filter callback.

This callback is called at the beginning of the enumeration process for a newly attached device. Through this callback, users are able to:

- filter which devices should be enumerated
- select the configuration number to use when enumerating the device

The device descriptor is passed to this callback to allow users to filter devices based on Vendor ID, Product ID, and class code.

Attention This callback must be non-blocking
Attention This callback must not submit any USB transfers

Param dev_desc [in] Device descriptor of the device to enumerate
Param bConfigurationValue [out] Configuration number to use when enumerating the device (starts with 1)
Return bool
- true: USB device will be enumerated
- false: USB device will not be enumerated

typedef struct *usb_transfer_s* **usb_transfer_t**

USB transfer structure.

This structure is used to represent a transfer from a software client to an endpoint over the USB bus. Some of the fields are made const on purpose as they are fixed on allocation. Users should call the appropriate USB Host Library function to allocate a USB transfer structure instead of allocating this structure themselves.

The transfer type is inferred from the endpoint this transfer is sent to. Depending on the transfer type, users should note the following:

- Bulk: This structure represents a single bulk transfer. If the number of bytes exceeds the endpoint's MPS, the transfer will be split into multiple MPS sized packets followed by a short packet.
- Control: This structure represents a single control transfer. This first 8 bytes of the data_buffer must be filled with the setup packet (see *usb_setup_packet_t*). The num_bytes field should be the total size of the transfer (i.e., size of setup packet + wLength).
- Interrupt: Represents an interrupt transfer. If num_bytes exceeds the MPS of the endpoint, the transfer will be split into multiple packets, and each packet is transferred at the endpoint's specified interval.
- Isochronous: Represents a stream of bytes that should be transferred to an endpoint at a fixed rate. The transfer is split into packets according to the each isoc_packet_desc. A packet is transferred at each interval of the endpoint. If an entire ISOC URB was transferred without error (skipped packets do not count as errors), the URB's overall status and the status of each packet descriptor will be updated, and the actual_num_bytes reflects the total bytes transferred over all packets. If the ISOC URB encounters an error, the entire URB is considered erroneous so only the overall status will updated.

备注： For Bulk/Control/Interrupt IN transfers, the num_bytes must be a integer multiple of the endpoint's MPS

备注： This structure should be allocated via usb_host_transfer_alloc()

备注： Once the transfer has be submitted, users should not modify the structure until the transfer has completed

typedef void (***usb_transfer_cb_t**)(*usb_transfer_t* *transfer)

USB transfer completion callback.

Enumerations

enum **usb_speed_t**

USB Standard Speeds.

Values:

enumerator **USB_SPEED_LOW**

USB Low Speed (1.5 Mbit/s)

enumerator **USB_SPEED_FULL**

USB Full Speed (12 Mbit/s)

enumerator **USB_SPEED_HIGH**

USB High Speed (480 Mbit/s)

enum **usb_transfer_type_t**

The type of USB transfer.

备注： The enum values need to match the bmAttributes field of an EP descriptor

Values:

enumerator **USB_TRANSFER_TYPE_CTRL**

enumerator **USB_TRANSFER_TYPE_ISOCHRONOUS**

enumerator **USB_TRANSFER_TYPE_BULK**

enumerator **USB_TRANSFER_TYPE_INTR**

enum **usb_transfer_status_t**

The status of a particular transfer.

Values:

enumerator **USB_TRANSFER_STATUS_COMPLETED**

The transfer was successful (but may be short)

enumerator **USB_TRANSFER_STATUS_ERROR**

The transfer failed because due to excessive errors (e.g. no response or CRC error)

enumerator **USB_TRANSFER_STATUS_TIMED_OUT**

The transfer failed due to a time out

enumerator **USB_TRANSFER_STATUS_CANCELED**

The transfer was canceled

enumerator **USB_TRANSFER_STATUS_STALL**

The transfer was stalled

enumerator **USB_TRANSFER_STATUS_OVERFLOW**

The transfer as more data was sent than was requested

enumerator **USB_TRANSFER_STATUS_SKIPPED**

ISOC packets only. The packet was skipped due to system latency or bus overload

enumerator **USB_TRANSFER_STATUS_NO_DEVICE**

The transfer failed because the target device is gone

Header File

- components/usb/include/usb/usb_types_ch9.h
- This header file can be included with:

```
#include "usb/usb_types_ch9.h"
```

- This header file is a part of the API provided by the `usb` component. To declare that your component depends on `usb`, add the following to your CMakeLists.txt:

```
REQUIRES usb
```

or

```
PRIV_REQUIRES usb
```

Unions

union **usb_setup_packet_t**

#include <usb_types_ch9.h> Structure representing a USB control transfer setup packet.

See Table 9-2 of USB2.0 specification for more details

Public Members

uint8_t **bmRequestType**

Characteristics of request

uint8_t **bRequest**

Specific request

uint16_t **wValue**

Word-sized field that varies according to request

uint16_t **wIndex**

Word-sized field that varies according to request; typically used to pass an index or offset

uint16_t **wLength**

Number of bytes to transfer if there is a data stage

struct *usb_setup_packet_t*::[anonymous] **USB_DESC_ATTR**

USB descriptor attributes

uint8_t **val**[USB_SETUP_PACKET_SIZE]

Descriptor value

union **usb_device_status_t**

#include <usb_types_ch9.h> Structure representing a USB device status.

See Figures 9-4 Information Returned by a GetStatus() Request to a Device of USB2.0 specification for more details

Public Members

uint16_t **self_powered**

1 - Device is currently self-powered, 0 - bus powered

uint16_t **remote_wakeup**

1 - the ability of the device to signal remote wakeup is enabled, 0 - the ability of the device to signal remote wakeup is disabled.

uint16_t **reserved**

reserved

struct *usb_device_status_t*::[anonymous] **USB_DESC_ATTR**

Packed

uint16_t **val**

Device status value

union **usb_standard_desc_t**

#include <usb_types_ch9.h> USB standard descriptor.

All USB standard descriptors start with these two bytes. Use this type when traversing over configuration descriptors

Public Members

uint8_t **bLength**

Size of the descriptor in bytes

uint8_t **bDescriptorType**

Descriptor Type

struct *usb_standard_desc_t*::[anonymous] **USB_DESC_ATTR**

USB descriptor attributes

uint8_t **val**[USB_STANDARD_DESC_SIZE]

Descriptor value

union **usb_device_desc_t**

#include <usb_types_ch9.h> Structure representing a USB device descriptor.

See Table 9-8 of USB2.0 specification for more details

Public Members

uint8_t **bLength**

Size of the descriptor in bytes

uint8_t **bDescriptorType**

DEVICE Descriptor Type

uint16_t **bcdUSB**

USB Specification Release Number in Binary-Coded Decimal (i.e., 2.10 is 210H)

uint8_t **bDeviceClass**

Class code (assigned by the USB-IF)

uint8_t **bDeviceSubClass**

Subclass code (assigned by the USB-IF)

uint8_t **bDeviceProtocol**

Protocol code (assigned by the USB-IF)

uint8_t **bMaxPacketSize0**

Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid)

uint16_t **idVendor**

Vendor ID (assigned by the USB-IF)

uint16_t **idProduct**

Product ID (assigned by the manufacturer)

uint16_t **bcdDevice**

Device release number in binary-coded decimal

uint8_t **iManufacturer**

Index of string descriptor describing manufacturer

uint8_t **iProduct**

Index of string descriptor describing product

uint8_t **iSerialNumber**

Index of string descriptor describing the device’s serial number

uint8_t **bNumConfigurations**

Number of possible configurations

struct *usb_device_desc_t*::[anonymous] **USB_DESC_ATTR**

USB descriptor attributes

uint8_t **val**[USB_DEVICE_DESC_SIZE]

Descriptor value

union **usb_config_desc_t**

#include <usb_types_ch9.h> Structure representing a short USB configuration descriptor.

See Table 9-10 of USB2.0 specification for more details

备注： The full USB configuration includes all the interface and endpoint descriptors of that configuration.

Public Members

uint8_t **bLength**

Size of the descriptor in bytes

uint8_t **bDescriptorType**

CONFIGURATION Descriptor Type

uint16_t **wTotalLength**

Total length of data returned for this configuration

uint8_t **bNumInterfaces**

Number of interfaces supported by this configuration

uint8_t **bConfigurationValue**

Value to use as an argument to the SetConfiguration() request to select this configuration

uint8_t **iConfiguration**

Index of string descriptor describing this configuration

uint8_t **bmAttributes**

Configuration characteristics

uint8_t **bMaxPower**

Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational.

struct *usb_config_desc_t*::[anonymous] **USB_DESC_ATTR**

USB descriptor attributes

uint8_t **val**[USB_CONFIG_DESC_SIZE]

Descriptor value

union **usb_iad_desc_t**

#include <usb_types_ch9.h> Structure representing a USB interface association descriptor.

Public Members

uint8_t **bLength**

Size of the descriptor in bytes

uint8_t **bDescriptorType**

INTERFACE ASSOCIATION Descriptor Type

uint8_t **bFirstInterface**

Interface number of the first interface that is associated with this function

uint8_t **bInterfaceCount**

Number of contiguous interfaces that are associated with this function

uint8_t **bFunctionClass**

Class code (assigned by USB-IF)

uint8_t **bFunctionSubClass**

Subclass code (assigned by USB-IF)

uint8_t **bFunctionProtocol**

Protocol code (assigned by USB-IF)

uint8_t **iFunction**

Index of string descriptor describing this function

struct *usb_iad_desc_t*::[anonymous] **USB_DESC_ATTR**

USB descriptor attributes

uint8_t **val**[USB_IAD_DESC_SIZE]

Descriptor value

union **usb_intf_desc_t**

#include <usb_types_ch9.h> Structure representing a USB interface descriptor.

See Table 9-12 of USB2.0 specification for more details

Public Members

uint8_t **bLength**

Size of the descriptor in bytes

uint8_t **bDescriptorType**

INTERFACE Descriptor Type

uint8_t **bInterfaceNumber**

Number of this interface.

uint8_t **bAlternateSetting**

Value used to select this alternate setting for the interface identified in the prior field

uint8_t **bNumEndpoints**

Number of endpoints used by this interface (excluding endpoint zero).

uint8_t **bInterfaceClass**

Class code (assigned by the USB-IF)

uint8_t **bInterfaceSubClass**

Subclass code (assigned by the USB-IF)

uint8_t **bInterfaceProtocol**

Protocol code (assigned by the USB)

uint8_t **iInterface**

Index of string descriptor describing this interface

struct *usb_intf_desc_t*::[anonymous] **USB_DESC_ATTR**

USB descriptor attributes

uint8_t **val**[USB_INTF_DESC_SIZE]

Descriptor value

union **usb_ep_desc_t**

#include <usb_types_ch9.h> Structure representing a USB endpoint descriptor.

See Table 9-13 of USB2.0 specification for more details

Public Members

uint8_t **bLength**

Size of the descriptor in bytes

uint8_t **bDescriptorType**

ENDPOINT Descriptor Type

uint8_t **bEndpointAddress**

The address of the endpoint on the USB device described by this descriptor

uint8_t **bmAttributes**

This field describes the endpoint’ s attributes when it is configured using the bConfigurationValue.

uint16_t **wMaxPacketSize**

Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.

uint8_t **bInterval**

Interval for polling Isochronous and Interrupt endpoints. Expressed in frames or microframes depending on the device operating speed (1 ms for Low-Speed and Full-Speed or 125 us for USB High-Speed and above).

struct *usb_ep_desc_t*::[anonymous] **USB_DESC_ATTR**

USB descriptor attributes

uint8_t **val**[USB_EP_DESC_SIZE]

Descriptor value

union **usb_str_desc_t**

#include <usb_types_ch9.h> Structure representing a USB string descriptor.

Public Members

uint8_t **bLength**

Size of the descriptor in bytes

uint8_t **bDescriptorType**

STRING Descriptor Type

uint16_t **wData**[]

UTF-16LE encoded

struct *usb_str_desc_t*::[anonymous] **USB_DESC_ATTR**

USB descriptor attributes

uint8_t **val**[USB_STR_DESC_SIZE]

Descriptor value

Macros

USB_DESC_ATTR

USB_B_DESCRIPTOR_TYPE_DEVICE

Descriptor types from USB2.0 specification table 9.5.

USB_B_DESCRIPTOR_TYPE_CONFIGURATION

USB_B_DESCRIPTOR_TYPE_STRING

USB_B_DESCRIPTOR_TYPE_INTERFACE

USB_B_DESCRIPTOR_TYPE_ENDPOINT

USB_B_DESCRIPTOR_TYPE_DEVICE_QUALIFIER

USB_B_DESCRIPTOR_TYPE_OTHER_SPEED_CONFIGURATION

USB_B_DESCRIPTOR_TYPE_INTERFACE_POWER

USB_B_DESCRIPTOR_TYPE_OTG

Descriptor types from USB 2.0 ECN.

USB_B_DESCRIPTOR_TYPE_DEBUG

USB_B_DESCRIPTOR_TYPE_INTERFACE_ASSOCIATION

USB_B_DESCRIPTOR_TYPE_SECURITY

Descriptor types from Wireless USB spec.

USB_B_DESCRIPTOR_TYPE_KEY

USB_B_DESCRIPTOR_TYPE_ENCRYPTION_TYPE

USB_B_DESCRIPTOR_TYPE_BOS

USB_B_DESCRIPTOR_TYPE_DEVICE_CAPABILITY

USB_B_DESCRIPTOR_TYPE_WIRELESS_ENDPOINT_COMP

USB_B_DESCRIPTOR_TYPE_WIRE_ADAPTER

USB_B_DESCRIPTOR_TYPE_RPIPE

USB_B_DESCRIPTOR_TYPE_CS_RADIO_CONTROL

USB_B_DESCRIPTOR_TYPE_PIPE_USAGE

Descriptor types from UAS specification.

USB_SETUP_PACKET_SIZE

Size of a USB control transfer setup packet in bytes.

USB_BM_REQUEST_TYPE_DIR_OUT

Bit masks belonging to the bmRequestType field of a setup packet.

USB_BM_REQUEST_TYPE_DIR_IN

USB_BM_REQUEST_TYPE_TYPE_STANDARD

USB_BM_REQUEST_TYPE_TYPE_CLASS

USB_BM_REQUEST_TYPE_TYPE_VENDOR

USB_BM_REQUEST_TYPE_TYPE_RESERVED

USB_BM_REQUEST_TYPE_TYPE_MASK

USB_BM_REQUEST_TYPE_RECIP_DEVICE

USB_BM_REQUEST_TYPE_RECIP_INTERFACE

USB_BM_REQUEST_TYPE_RECIP_ENDPOINT

USB_BM_REQUEST_TYPE_RECIP_OTHER

USB_BM_REQUEST_TYPE_RECIP_MASK

USB_B_REQUEST_GET_STATUS

Bit masks belonging to the bRequest field of a setup packet.

USB_B_REQUEST_CLEAR_FEATURE

USB_B_REQUEST_SET_FEATURE

USB_B_REQUEST_SET_ADDRESS

USB_B_REQUEST_GET_DESCRIPTOR

USB_B_REQUEST_SET_DESCRIPTOR

USB_B_REQUEST_GET_CONFIGURATION

USB_B_REQUEST_SET_CONFIGURATION

USB_B_REQUEST_GET_INTERFACE

USB_B_REQUEST_SET_INTERFACE

USB_B_REQUEST_SYNCH_FRAME

USB_W_VALUE_DT_DEVICE

Bit masks belonging to the wValue field of a setup packet.

USB_W_VALUE_DT_CONFIG

USB_W_VALUE_DT_STRING

USB_W_VALUE_DT_INTERFACE

USB_W_VALUE_DT_ENDPOINT

USB_W_VALUE_DT_DEVICE_QUALIFIER

USB_W_VALUE_DT_OTHER_SPEED_CONFIG

USB_W_VALUE_DT_INTERFACE_POWER

USB_SETUP_PACKET_INIT_GET_STATUS(setup_pkt_ptr)

Initializer for a GET_STATUS request.

Sets the address of a connected device

USB_SETUP_PACKET_INIT_SET_ADDR(setup_pkt_ptr, addr)

Initializer for a SET_ADDRESS request.

Sets the address of a connected device

USB_SETUP_PACKET_INIT_GET_DEVICE_DESC(setup_pkt_ptr)

Initializer for a request to get a device's device descriptor.

USB_SETUP_PACKET_INIT_GET_CONFIG(setup_pkt_ptr)

Initializer for a request to get a device's current configuration number.

USB_SETUP_PACKET_INIT_GET_CONFIG_DESC(setup_pkt_ptr, desc_index, desc_len)

Initializer for a request to get one of the device's current configuration descriptor.

- desc_index indicates the configuration's index number
- Number of bytes of the configuration descriptor to get

USB_SETUP_PACKET_INIT_SET_CONFIG(setup_pkt_ptr, config_num)

Initializer for a request to set a device's current configuration number.

USB_SETUP_PACKET_INIT_SET_INTERFACE(setup_pkt_ptr, intf_num, alt_setting_num)

Initializer for a request to set an interface's alternate setting.

USB_SETUP_PACKET_INIT_GET_STR_DESC(setup_pkt_ptr, string_index, lang_id, desc_len)

Initializer for a request to get an string descriptor.

USB_STANDARD_DESC_SIZE

Size of dummy USB standard descriptor.

USB_DEVICE_DESC_SIZE

Size of a USB device descriptor in bytes.

USB_CLASS_PER_INTERFACE

Possible base class values of the bDeviceClass field of a USB device descriptor.

USB_CLASS_AUDIO

USB_CLASS_COMM

USB_CLASS_HID

USB_CLASS_PHYSICAL

USB_CLASS_STILL_IMAGE

USB_CLASS_PRINTER

USB_CLASS_MASS_STORAGE

USB_CLASS_HUB

USB_CLASS_CDC_DATA

USB_CLASS_CSCID

USB_CLASS_CONTENT_SEC

USB_CLASS_VIDEO

USB_CLASS_WIRELESS_CONTROLLER

USB_CLASS_PERSONAL_HEALTHCARE

USB_CLASS_AUDIO_VIDEO

USB_CLASS_BILLBOARD

USB_CLASS_USB_TYPE_C_BRIDGE

USB_CLASS_MISC

USB_CLASS_APP_SPEC

USB_CLASS_VENDOR_SPEC

USB_SUBCLASS_VENDOR_SPEC

Vendor specific subclass code.

USB_CONFIG_DESC_SIZE

Size of a short USB configuration descriptor in bytes.

备注： The size of a full USB configuration includes all the interface and endpoint descriptors of that configuration.

USB_BM_ATTRIBUTES_ONE

Bit masks belonging to the bmAttributes field of a configuration descriptor.

Must be set

USB_BM_ATTRIBUTES_SELFPOWER

Self powered

USB_BM_ATTRIBUTES_WAKEUP

Can wake-up

USB_BM_ATTRIBUTES_BATTERY

Battery powered

USB_IAD_DESC_SIZE

Size of a USB interface association descriptor in bytes.

USB_INTF_DESC_SIZE

Size of a USB interface descriptor in bytes.

USB_EP_DESC_SIZE

Size of a USB endpoint descriptor in bytes.

USB_B_ENDPOINT_ADDRESS_EP_NUM_MASK

Bit masks belonging to the bEndpointAddress field of an endpoint descriptor.

USB_B_ENDPOINT_ADDRESS_EP_DIR_MASK

USB_W_MAX_PACKET_SIZE_MPS_MASK

Bit masks belonging to the wMaxPacketSize field of endpoint descriptor.

USB_W_MAX_PACKET_SIZE_MULT_MASK

USB_BM_ATTRIBUTES_XFERTYPE_MASK

Bit masks belonging to the bmAttributes field of an endpoint descriptor.

USB_BM_ATTRIBUTES_XFER_CONTROL

USB_BM_ATTRIBUTES_XFER_ISOC

USB_BM_ATTRIBUTES_XFER_BULK

USB_BM_ATTRIBUTES_XFER_INT

USB_BM_ATTRIBUTES_SYNCTYPE_MASK

USB_BM_ATTRIBUTES_SYNC_NONE

USB_BM_ATTRIBUTES_SYNC_ASYNC

USB_BM_ATTRIBUTES_SYNC_ADAPTIVE

USB_BM_ATTRIBUTES_SYNC_SYNC

USB_BM_ATTRIBUTES_USAGETYPE_MASK

USB_BM_ATTRIBUTES_USAGE_DATA

USB_BM_ATTRIBUTES_USAGE_FEEDBACK

USB_BM_ATTRIBUTES_USAGE_IMPLICIT_FB

USB_EP_DESC_GET_XFERTYPE(desc_ptr)

Macro helpers to get information about an endpoint from its descriptor.

USB_EP_DESC_GET_EP_NUM(desc_ptr)

USB_EP_DESC_GET_EP_DIR(desc_ptr)

USB_EP_DESC_GET_MPS(desc_ptr)

USB_EP_DESC_GET_MULT(desc_ptr)

USB_EP_DESC_GET_SYNCTYPE(desc_ptr)

USB_EP_DESC_GET_USAGETYPE(desc_ptr)

USB_STR_DESC_SIZE

Size of a short USB string descriptor in bytes.

Enumerations

enum **usb_device_state_t**

USB2.0 device states.

See Table 9-1 of USB2.0 specification for more details

备注： The USB_DEVICE_STATE_NOT_ATTACHED is not part of the USB2.0 specification, but is a catch all state for devices that need to be cleaned up after a sudden disconnection or port error.

Values:

enumerator **USB_DEVICE_STATE_NOT_ATTACHED**

The device was previously configured or suspended, but is no longer attached (either suddenly disconnected or a port error)

enumerator **USB_DEVICE_STATE_ATTACHED**

Device is attached to the USB, but is not powered.

enumerator **USB_DEVICE_STATE_POWERED**

Device is attached to the USB and powered, but has not been reset.

enumerator **USB_DEVICE_STATE_DEFAULT**

Device is attached to the USB and powered and has been reset, but has not been assigned a unique address. Device responds at the default address.

enumerator **USB_DEVICE_STATE_ADDRESS**

Device is attached to the USB, powered, has been reset, and a unique device address has been assigned. Device is not configured.

enumerator **USB_DEVICE_STATE_CONFIGURED**

Device is attached to the USB, powered, has been reset, has a unique address, is configured, and is not suspended. The host may now use the function provided by the device.

enumerator **USB_DEVICE_STATE_SUSPENDED**

Device is, at minimum, attached to the USB and is powered and has not seen bus activity for 3 ms. It may also have a unique address and be configured for use. However, because the device is suspended, the host may not use the device ' s function.

维护注意事项

备注: 有关 USB 主机栈内部实现的更多细节，请参阅*USB Host Maintainers Notes (Introduction)*。

USB Host Maintainers Notes (Introduction)

This document contains information regarding the implementation details of the USB Host stack. This document is intended for the maintainers and third-party contributors of the USB Host stack. Users of the USB Host stack should refer to *USB 主机* instead.

警告: The implementations details of the USB Host stack is categorized as private API. Thus, all layers (other than the USB Host Library) do not adhere to *ESP-IDF's versioning scheme* (i.e., breaking changes are permitted).

This document is split into the following sections:

USB Host Maintainers Notes (Design Guidelines)

Design Considerations The design of the Host Stack takes into account the following design considerations:

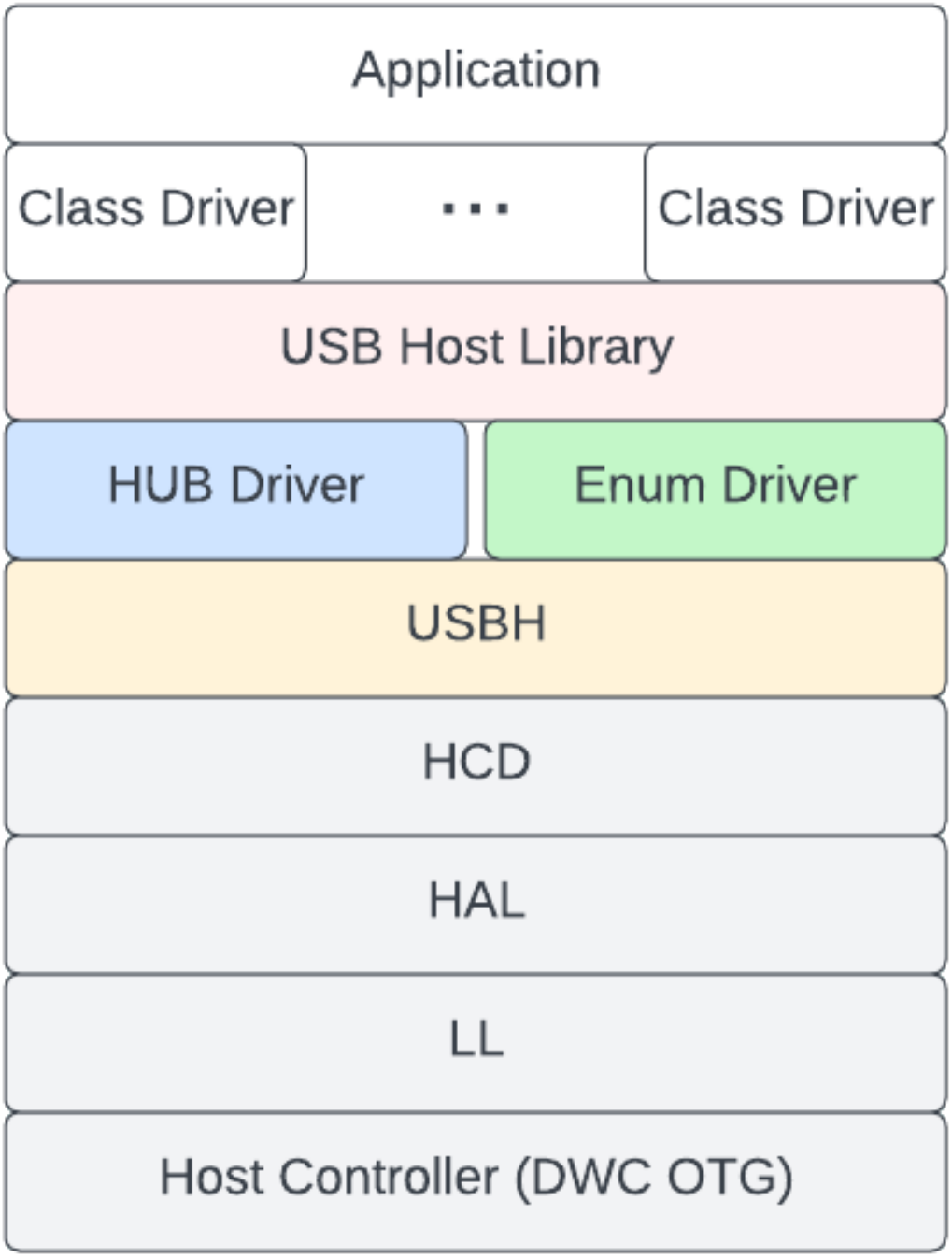
Limited Hardware Resources:

The embedded nature of Host Stack means limited hardware resources (such as memory and processing power) when compared to larger host systems.

USB2.0 Chapter 10:

Chapter 10 of the USB 2.0 specification specifies certain requirements of USB Host systems, in particular the required layers of the USB Host's system software.

Diverse Use Case Complexities:


Application
Class Driver
...
Class Driver
USB Host Library
HUB Driver
Enum Driver
USBH
HCD
HAL
LL
Host Controller (DWC OTG)

The embedded nature of the Host Stack also means a wide range of use cases with differing complexities. Some USB Host applications aim to only support a single vendor specific device, whereas other applications require support for a wide range of devices of different classes.

Requirements Given the design considerations above, the Host Stack was designed with the following set of requirements:

DMA Support **Requirement: The Host Stack must support DMA.**

The Host Stack must support DMA in order to reduce CPU's workload. DMA support allows the automatic copying of USB transfer data to/from the Host Controller without CPU intervention. This is especially critical given the embedded nature of the CPU (i.e., lower CPU frequencies) and large maximum data payloads of USB transfers (e.g., 1023 bytes for isochronous transfers).

Minimize Memory Copies **Requirement: The Host Stack should minimize the amount of memory copies when passing data between layers.**

Various data and objects (e.g., USB transfers) need to be passed between multiple layers of the Host Stack. The Host Stack should minimize the amount of memory copies that occur between layers by allocating the data's/object's memory once, and simply passing a pointer to that data/object between the layers. Therefore, the Host Stack requires some standardized data types shared across multiple layers (see USB2.0 Section 10.3.4).

Event Driven **Requirement: The Host Stack must allow for event driven operation (i.e., the Host Stack's API must not be polling).**

The Host Stack needs to support some CPU intensive use application scenarios such as video streaming (i.e., UVC class). Therefore, the Host Stack should minimize CPU usage by allowing for completely event driven operation, thus reserving the majority of CPU time for the application itself (i.e., video encoding/decoding in this case).

The Host Stack needs to communicate events across the layers using interrupts, callbacks, and FreeRTOS synchronization primitives (e.g., queues and semaphores).

No Task Creation **Requirement: All layers of the Host Stack below (and including) the Host Library layer must not create any tasks.**

Task stacks are generally one of the most memory intensive parts of an ESP-IDF applications. Given the wide range of applications scenarios, the number of tasks created (and their stack sizes) can vary greatly. For example...

- applications that require low latency or high throughput (such as isochronous transfers) may choose to create a dedicated task to handle those transfers in order to minimize latency.
- applications that do not have strict latency requirements (such as bulk transfers) may choose to handle those transfers from a shared task in order to save some memory.

Therefore, all layers of the Host Stack below (and including) the Host Library layer **must not** create any tasks. Instead, these layers should expose handlers functions to be called from tasks created by the upper layers. Task creation will be delegated to the class driver or application layer.

Operable at Different Layers Given the wide range of use case complexities, the Host Stack must be operable at different layers, allowing users to use the Host Stack at a lower layer (e.g., the HCD or HAL) or at a higher layer (e.g., a class driver).

Being operable at different layers allows the users to decide on the appropriate trade-off between convenience, control, and optimization for their application when using the Host Stack. For example...

- Host Stack applications that support a dedicated custom device may want to use a lower level of abstraction for better optimization, control, and simpler API.
- Host Stack applications that need to support a wide range of device classes requires the full Host Stack so that device enumeration is automatically handled.

Coding Conventions The Host Stack follows the following set of coding rules/guidelines for better code readability and maintainability:

Symbols Use Layer Name As Prefix For each layer of the Host Stack, the symbols exposed by that layer (i.e., functions, types, macros) must be prefixed with that layer's name. For example, the symbols exposed by the HCD layer will be prefixed `hcd_.../HCD_...`.

However, internal symbols (e.g., static functions) **should not** be prefixed with their layer's name. This makes it easier to differentiate between internal and external symbols when modifying that layer's source code.

Critical Section Functions Prefixed With _ In each layer of the Host Stack, there are various static functions that must be called inside a critical section. The names of these functions are prefixed with _ (e.g., `_func_called_from_crit()`) to make it easier for maintainers to differentiate which functions should be called from critical sections. For example...

```
some_func();  // Called outside critical section
taskENTER_CRITICAL(&some_lock);
_some_func_crit();  // Called inside critical section. _ prefix makes it easier to␣
↪differentiate
taskEXIT_CRITICAL(&some_lock);
```

Grouping Structure Members by Locking Mechanism Some layers of the Host Stack utilize multiple locking schemes (e.g., critical sections and task mutexes) to ensure thread safety, where each locking scheme offers a different level of protection. However, member variables of the same object can be protected by different locking scheme. Therefore, to clearly demarcate the different locking schemes and their associated variables, structure members are grouped by locking scheme as nested structures.

表 7: Locking Scheme

Locking Scheme	Nested Structure	Description
Critical Sections	`dynamic`	Shared data accessed from both a task context and ISR context are protected by a critical section.
Task Mutexes	`mux_protected`	Shared data accessed from only a task context are protected by a FreeRTOS Mutex
Single Thread	`single_thread`	Data that is only ever accessed by the same task do not require the use of any locks.
Constant	`constant`	Constant data is set once during the object's instantiation and never changed again. Thus, any task or ISR can freely the constant data without the use of locks, so long as the variable is never written to.

Grouping structure members by locking scheme makes the code more maintainable as it makes clear which locking scheme is required when accessing a particular member variable, as demonstrated in the code snippet below:

```
typedef struct some_obj some_obj_t;

some_obj_t obj;

// Accessing dynamic members requires critical section
taskENTER_CRITICAL(&some_lock);
obj.dynamic.varA = 1;
taskEXIT_CRITICAL(&some_lock);

// Accessing mutex protected members requires taking the mutex
xSemaphoreTake(&some_mux, portMAX_DELAY);
```

(下页继续)

(续上页)

```
obj.mux_protected.varB = 1;
xSemaphoreGive(&some_mux);

// Accessing single thread members does not require locking so long as this is the␣
↪only task to access it
obj.single_thread.varC = 1;

// Accessing constant members requires no locking. But only read access is allowed
int local_var = obj.constant.varD;
```

USB Host Maintainers Notes (Architecture) The Host Stack is roughly split into multiple layers of abstraction, with each layer representing different USB concepts and a different level of USB Host operation. For example, a higher layer may present an abstraction of devices and application data transfers, whereas a lower layer may present an abstraction of endpoints and USB transfers.

Layer Descriptions The layers of the Host Stack are described in the following table. The layers are ordered from lowest layer (i.e, furthest away from the user) to highest layer (i.e., closest to the user).

表 8: Host Stack Layers

Layer	Files	Description
Host Controller (DWC_OTG)	N/A	This layer represents the USB Controller Hardware of the ESP32-P4. The API presented by this layer is the register interface of the controller.
LL	`usbh_ll.h`	The LL (Low Level) layer abstracts the basic register access of the USB controller according to ESP-IDF's *Hardware Abstraction API Guidelines*. In other words, this layer provides APIs to access the controller's registers and format/parse the controller's DMA descriptors.
HAL	`usbh_hal.h`, `usbh_hal.c`	The HAL (Hardware Abstraction Layer) abstracts the operating steps of the USB controller into functions according to ESP-IDF's *Hardware Abstraction API Guidelines*. This layer also abstracts the controller's host port and host channels, and provides APIs to operate the them.
HCD	`hcd.h`, `hcd.c`	The HCD (Host Controller Driver) acts as hardware agnostic API for all USB controllers (i.e., an API that can theoretically be used with any USB controller implementation). This layer also abstracts the root port (i.e., root hub) and USB pipes.
USBH and Hub Driver	`usbh.h`, `usbh.c`	The USBH (USB Host Driver) layer is equivalent to the USBD layer described in chapter 10 of the USB2.0 specification. The USBH presents an abstraction of USB devices, internally manages a list of connected devices (i.e., device pool), and also arbitrates device sharing between clients (i.e., tracks which endpoints are in use and also presents a shared endpoint 0).
Hub Driver	`hub.h`, `hub.c`	The Hub Driver layer acts as a special client of the USBH that is responsible for handling device attachment/detachment, and notifying the USBH of such events. For device attachment, the Hub Driver also handles the enumeration process as well.
USB Host Library	`usb_host.h`, `usb_host.c`	The USB Host Library layer is the lowest public API layer of the Host Stack and presents the concept of USB Host Clients. The abstraction of clients allows for multiple class drivers to coexist simultaneously (where each class roughly maps to a single client) and also acts as a mechanism for division of labor (where each client is responsible for its own processing and event handling).
Host Class Drivers	See the ESP-USB repository or the USB Host examples in ESP-IDF (via peripherals/usb/host).	The Host Class Drivers implement the host side of a particular device class (e.g., CDC, MSC, HID). The exposed API is specific to each class driver.

Layer Dependencies The Host Stack roughly follows a top to bottom hierarchy with inter-layer dependencies. Given layers A (highest), B, and C (lowest), the Host Stack has the following inter-layer dependency rules:

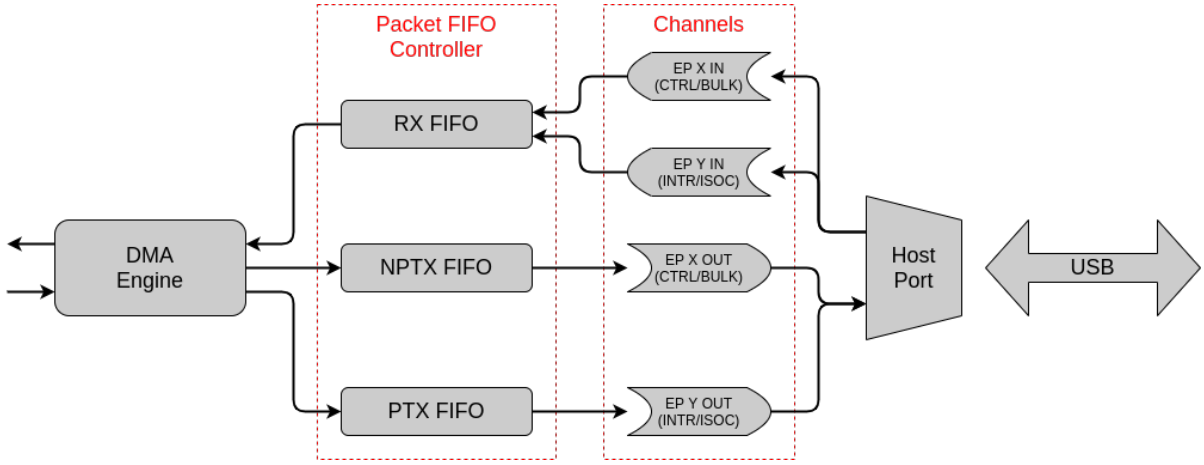
- a particular layer can use the API of any layer directly below (Layer A using layer B is allowed)
- a particular layer can use the API of any layer indirectly below (Layer A using layer C is allowed) i.e., skipping layers.
- a particular layer must not use the API of any layer above (Layer C using layer A/B is forbidden)

备注: Layer skipping is permitted in order to circumvent the need to repeat the same abstraction across multiple layers. For example, the abstraction of pipes are presented at the HCD layer but are used by multiple layers above.

USB Host Maintainers Notes (DWC_OTG Controller) The ESP32-P4 uses a DesignWare USB 2.0 On-the-Go Controller (henceforth referred to as DWC_OTG in this document) as its underlying hardware controller, where the DWC_OTG operates in Host Mode with Scatter/Gather DMA enabled.

备注： This section only summarizes the operation of the DWC_OTG operation in Host Mode at a high level. For full details of the DWC_OTG, refer to the DWC_OTG Databook and Programming Guide.

Host Mode Operating Model A simplified version of the operating model of the DWC_OTG in Host Mode is illustrated in the diagram below. The diagram contains some of the key concepts and terms regarding DWC_OTG Host Mode.



备注： Refer to Databook section 2.1.4 (Host Architecture) for more details

---

Host Port The Host Port represents the single USB port provided by the DWC_OTG (in USB terms, this can be thought a single USB port of the root hub of the bus). The Host Port can only connect to a single device, though more devices can be connected via hub devices.

The Host Port is responsible for:

- detecting direct device connections/disconnections
- detecting the speed of the directly connected device
- issuing various bus signals (such as suspend, resume, reset)

Host Channels In Host Mode, the DWC_OTG uses channels to communicate with device endpoints, where one channel maps to a particular endpoint (in USB terms, channels are the hardware representation of pipes). For example, a channel will map to EP 1 OUT. Each channel has its own set of CSRs (Control and Status Registers) so that they can independently configured and controlled independently. A channel's CSRs are used to...

- specify the details of the channel's target endpoint (e.g., device address, EP number, transfer type, direction)
- start a transfer on the channel (e.g., by setting up DMA descriptors)

When using Scatter/Gather DMA, transfers on Host Channels are completely event driven. Users simply fill out the appropriate DMA descriptors, fill in the channel's CSRs, then enable the channel. The channel will then generate an interrupt when the transfer completes.

Data FIFOs In Host Mode, the DWC_OTG uses multiple FIFOs as a staging area for the data payloads of USB transfers. When using DMA, the DMA engine will copy data between the TX/RX FIFOs and ESP32-P4's internal memory:

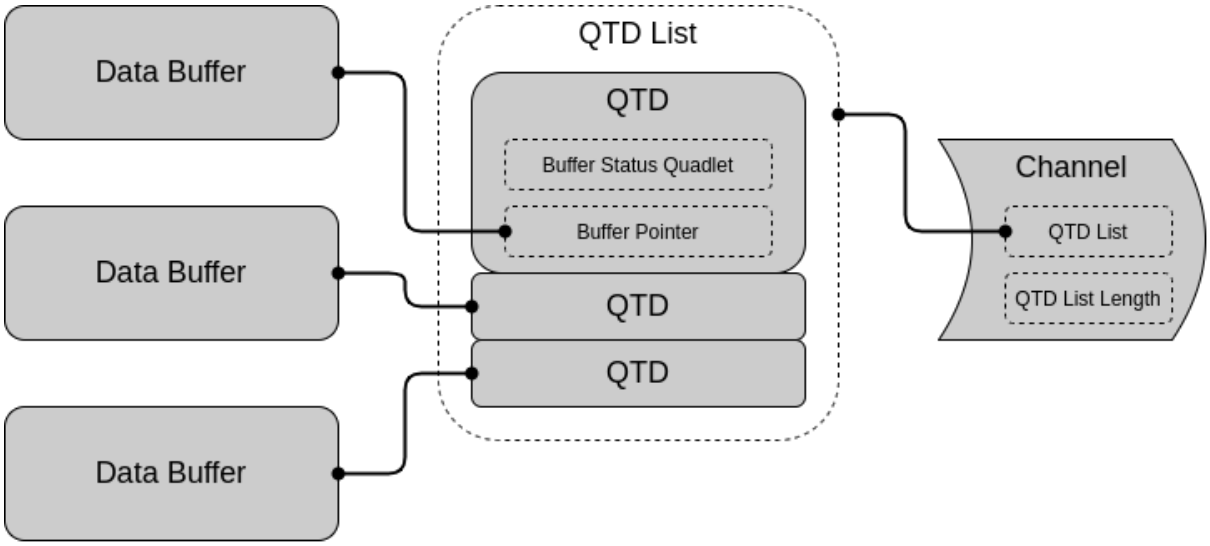
- For an OUT transfer, the transfer's data payload is copied from main memory to one of the TX FIFOs by DMA. The MAC Layer will then transmit that data payload in accordance to USB packet formatting.
- For an IN transfer, the MAC Layer will parse the received USB packet and store the received data payload in the RX FIFO. The data is then copied to main memory by DMA.

The destination FIFO depends on the direction and transfer type of the channel:

- All IN channel data goes to the RX FIFO
- All non-periodic (i.e., Control and Bulk) OUT channel data goes to the Non-periodic TX (NPTX) FIFO
- All periodic (i.e., Interrupt and Isochronous) OUT channel data goes to the Periodic TX (PTX) FIFO

备注： The separation of non-periodic and periodic OUT channels to the NPTX and PTX FIFOs is due to the periodic nature of Interrupt and Isochronous endpoints (specified by the `bInterval` value of the endpoint). The DWC_OTG automatically schedules these periodic transfers, thus a separate PTX FIFO allows these periodic transfers to be staged separately.

DMA Engine The DMA engine is responsible for copying data between the FIFOs and main memory. In Host Mode Scatter/Gather DMA, a particular channel can carry out multiple transfers automatically without software intervention. The following diagram illustrates the DWC_OTG Host Mode Scatter/Gather DMA Memory Structures.



- Each USB transfer is described by a Queue Transfer Descriptor (QTD). Each QTD consists of:
  - A 32-bit Buffer Status Quadlet specifying details of the transfer, and also reports the status of the transfer on completion. The Buffer Status Quadlet has bits to specify whether the QTD should generate an interrupt and/or halt the channel on completion.
  - A 32-bit pointer to the data buffer containing the data payload for OUT transfers, or an empty buffer used to store the data payload for IN transfers.
- The data payload of each QTD can be larger than the MPS (Maximum Packet Size) of its target endpoint. The DWC_OTG hardware automatically handles splitting of the transfer into multiple transactions.
- Multiple QTDs can be placed into a single QTD List. A channel will then execute each QTD in the list automatically, and optionally loop back around if configured to do so.
- Before a channel starts data transfer, it is configured with a QTD list (and QTD list length). Once the channel is enabled, USB transfers are executed automatically by the hardware.
- A channel can generate interrupts (configurable) on completion of a particular QTD, or an entire QTD list.

备注： Refer to Programming Guide section 6.2.1 (Descriptor Memory Structures) for more details

---

Hardware Configuration The DWC_OTG IP is configurable. The notable Host related configurations of the ESP32-P4's DWC_OTG are listed below:

表 9: ESP32-P4's DWC_OTG Configuration

Description	Configuration
Host and Device Mode support with OTG	`OTG_MODE = 0`
High Speed (HS), Full Speed (FS) and Low Speed (LS) support	`OTG_FSPHY_INTERFACE = 2`, `OTG_HSPHY_INTERFACE = 3`
Internal DMA controller with Scatter/Gather DMA	`OTG_ARCHITECTURE = 2`, `OTG_EN_DESC_DMA = 1`
Split transfers not supported	`OTG_SINGLE_POINT = 1`
16 Host Mode channels	`OTG_NUM_HOST_CHAN = 16`
All transfer types supported, including ISOC and INTR OUT transfers	`OTG_EN_PERIO_HOST = 1`
Dynamically sized Data FIFO of 4096 bytes (1024 lines)	`OTG_DFIFO_DYNAMIC = 1`, `OTG_DFIFO_DEPTH = 1024`
Only 4 periodic and 4 non-periodic transactions per microframe	`OTG_NPERIO_TX_QUEUE_DEPTH = 4`, `OTG_PERIO_TX_QUEUE_DEPTH = 4`

Scatter/Gather DMA Transfer The basic operating procedure for Host channels transfers consists of the following steps:

1. Prepare data buffers, QTDs, and QTD list. In particular, which QTDs should halt the channel (and generate an interrupt) on completion.
2. Set channel/endpoint characteristics via CSRs (such as EP address, transfer type, EP MPS etc).
3. Set channel's QTD list related CSRs (such as QTD list pointer and QTD list length) and channel interrupt CSRs
4. Enable the channel. Transfers are now handled automatically by hardware using DMA.
5. The Channel generates an interrupt on a channel event (e.g., QTD completion or channel error).
6. Parse the channel interrupt to determine what event occurred.
7. Parse the QTDs to determine the result of each individual transfer.

However, there are some minor differences in channel operation and QTD list usage depending on the transfer type.

Bulk Bulk transfers are a simplest. Each QTD represents a bulk transfer of a particular direction, where the DWC_OTG automatically splits a particular QTD into multiple MPS sized transactions. Thus it is possible to fill a QTD list with multiple bulk transfers, and have the entire list executed automatically (i.e., only interrupt on completion of the last QTD).

Control Control transfers are more complicated as they are bi-directional (i.e., each control transfer stage can have a different direction). Thus, a separate QTD is required for each stage, and each QTD must halt the channel on completion. Halting the channel after each QTD allows changing the channel's direction to be changed by reconfiguring the channel's CSRs. Thus a typical control transfer consists of 3 QTDs (one for each stage).

Interrupt In accordance with the USB2.0 specification, interrupt transfers executes transactions at the endpoints specified service period (i.e., `bInterval`). A particular interrupt endpoint may not execute more than one interrupt transaction within a service period. The service period is specified in number of microframes/frames, thus a particular interrupt endpoint will generally execute one transaction every Nth microframe/frame until the transfer is complete. For interrupt channels, the service period of a particular channel (i.e., `bInterval`) is specified via the Host Frame List (see section 6.5 of programming guide for more details).

备注： HS USB allows an interrupt endpoint to have 3 interrupt transactions in a single microframe. See USB2.0 specification section 5.7.3 (Interrupt Transfer Packet Size Constraints) for more details.

Thus, interrupt transfers in Host Mode Scatter/Gather DMA have the following peculiarities:

- If a QTD payload is larger than the endpoint's MPS, the channel will automatically split the transfer into multiple MPS sized transactions (similar to bulk transfers). However, each transaction **is executed at endpoint's specified service period** (i.e., one transaction per `bInterval`) until the transfer completes.
- For Interrupt IN transfers, if a short packet is received (i.e., transaction's data payload is < MPS), this indicates that the endpoint has no more data to send. In this case:
  - the channel generates an extra channel interrupt even if the transfer's QTD did not set the IOC (interrupt on complete) bit.
  - however, the channel is not halted even if this extra channel interrupt is generated.
  - software must then use this extra interrupt to manually halt the interrupt channel (thus canceling any remaining QTDs in the QTD list).

备注： Due to the interrupt transfer peculiarities, it may be easier for software allocate a QTD for each transaction instead of an entire transfer.

Isochronous In accordance with the USB2.0 specification, isochronous transfers executes transactions at the endpoints specified service period (i.e., `bInterval`) in order to achieve a constant rate of data transfer. A particular isochronous endpoint may not execute more than one isochronous transaction within a service period. The service period is specified in number of microframes/frames, thus a particular isochronous endpoint will generally execute one transaction every Nth microframe/frame until the transfer is complete. For isochronous channels, the service period of a particular channel (i.e., `bInterval`) is specified via the Host Frame List (see section 6.5 of programming guide for more details).

However, unlike interrupt transactions, isochronous transactions are not retried on failure (or NAK), due to the need to maintain the constant data rate.

备注： HS USB allows an isochronous endpoint to have 3 interrupt transactions in a single microframe. See USB2.0 specification section 5.6.3 (Isochronous Transfer Packet Size Constraints) for more details.

Thus, isochronous transfers in Host Mode Scatter/Gather DMA have the following peculiarities:

- A QTD must be allocated for each microframe/frame. However, non-service service period QTDs should be left blank (i.e., only ever Nth QTD should be filled if the channel's service period is every Nth microframe/frame).
- **Each filled QTD must represent a single transaction instead of a transfer**.
- Because isochronous transactions are not retried on failure, the status each completed QTD must be checked.

Supplemental Notes Some of the DWC_OTG's behaviors are not mentioned in the Databook or Programming Guide. This section describes some of those behaviors that are relevant to the Host stack's implementation.

Port Errors Do Not Trigger a Channel Interrupt If a port error occurs (such as a sudden disconnection or port over-current) while there are one or more active channels...

- The active channels remains active (i.e., `HCCHAR.ChEna` remains set) and no channel interrupts are generated.
- Channels could in theory be disabled by setting `HCCHAR.ChDis`, but this does not work for Isochronous channels as the channel disabled interrupt is never generated.

Therefore, on port errors, a controller soft reset should be used to ensure all channels are disabled.

Port Reset Interrupts

- When the DWC_OTG issues a reset signal on its port, and during the reset signal the device disconnects, the disconnection interrupt (i.e., `HPRT.PrtConnDet`) is not generated until the reset is deasserted.
- When resetting an already enabled port (i.e., `HPRT.PrtEna`) such as a second reset during enumeration or a run-time reset, a Port Enable/Disable Change interrupt (i.e., `HPRT.PrtEnChng`) is generated both on the assertion and deassertion of the reset signal.

USB Host Driver (USBH)

Introduction The USB Host Driver (henceforth referred to as USBH) provides a USB Host software interface which abstracts USB devices. The USBH interface provides APIs to...

- manage the device pool (i.e., adding and removing devices)
- address and configure a device (i.e., setting device and configuration descriptors)
- submit transfers to a particular endpoint of a device

Requirements

USB Specification Requirements Chapter 10 of the USB 2.0 specification outlines some requirements of the USBH (referred to as USBD in the specification). The design of the USBH takes into consideration these requirements from the specification.

- Default pipe of a device is owned by the USBH
- All other pipes are owned and managed by clients of the USBH
- USBH interface must provide the following services
  - Configuration and command mechanism
  - Transfer services via both command and pipe mechanisms
  - Event notification
  - Status reporting and error recovery

Host Stack Requirements In addition to the USB 2.0 specification requirements, the USBH also takes into consideration the requirements set for the overall Host Stack (see *USB Host Maintainers Notes (Design Guidelines)*):

- USBH must not instantiate any tasks/threads
- USBH must be event driven, providing event callbacks and an event processing function

Implementation & Usage

Events & Processing The USBH is completely event driven and all event handling is done via then `usbh_process()` function. The `usbh_config_t.proc_req_cb` callback provided on USBH installation will be called when processing is required. Typically, `usbh_process()` will be called from a dedicated thread/task.

The USBH exposes the following event callbacks:

- `usbh_event_cb_t` used to indicate various events regarding a particular device and control transfers to EP0. This callback is called from the context of `usbh_process()`
- `usbh_ep_cb_t` used to indicate events for all other endpoints. This callback is not called from the `usbh_process()` context (currently called from an HCD interrupt context).

Device Pool The USBH keeps track of all currently connected devices by internally maintaining a device pool (simply a linked list) where each device is represented by a device object.

The USB 2.0 specification "assumes a specialized client of the USBD, called a hub driver, that acts as a clearinghouse for the addition and removal of devices from a particular hub". As a result, the USBH is completely reliant on an external client(s) (typically a Hub Driver) to inform the USBH of device addition and removal. The USBH provides the following APIs for device addition and removal:

- `usbh_devs_add()` which will allocate a new device object and add it to the device pool. The newly added device will be unenumerated, meaning the device object will...
  - be assigned to address 0
  - have no device and configuration descriptor
- `usbh_devs_remove()` which will indicate to the USBH that a device has been removed (such as due to a disconnection or a port error).
  - If the device is not currently opened (i.e., used by one or more clients), the USBH will free the underlying device object immediately.
  - If the device is currently opened, a `USBH_EVENT_DEV_GONE` event will be propagated and the device will be flagged for removal. The last client to close the device will free the device object.
  - When a device object is freed, a `USBH_EVENT_DEV_FREE` event will be propagated. This event is used to indicate that the device's upstream port can be recycled.

Device Enumeration Newly added devices will need to be enumerated. The USBH provides various `usbh_dev_set_...()` functions to enumerate the device, such as assigning the device's address and setting device/configuration/string descriptors. Given that USBH devices can be shared by multiple clients, attempting to enumerate a device while another client has opened the device can cause issues.

Thus, before calling any `usbh_dev_set_...()` enumeration function, a device must be locked for enumeration by calling `usbh_dev_enum_lock()`. This prevents the device from being opened by any other client but the enumerating client.

After enumeration is complete, the enumerating client can call `usbh_devs_trigger_new_dev_event()` to propagate a `USBH_EVENT_NEW_DEV` event.

Device Usage Clients that want to use a device must open the device by calling `usbh_devs_open()` and providing the device's address. The device's address can either be obtained from a `USBH_EVENT_NEW_DEV` event or by calling `usbh_devs_addr_list_fill()`.

Opening a device will do the following:

- Return a `usb_device_handle_t` device handle which can be used to refer to the device in various USBH functions
- Increment the device's internal `open_count` which indicates how many clients have opened the device. As long as `open_count > 0`, the underlying device object will not be freed, thus guaranteeing that the device handle refers to a valid device object.

Once a client no longer needs to use a device, the client should call `usbh_devs_close()` thus invalidating the device handle.

备注: Most device related APIs accept `usb_device_handle_t` as an argument, which means that the calling client must have previously opened the device to obtain the device handle beforehand. This design choice is intentional in order to enforce an "open before use" pattern.

However, a limited set of APIs (e.g., `usbh_devs_remove()`) refer to devices using a Unique Identifier (`uid`) which is assigned on device addition (see `usbh_devs_add()`). The use of `uid` in these functions allows their callers to refer to a device **without needing to open it** due to the lack of a `usb_device_handle_t`.

As a result, it is possible that a caller of a `uid` function may refer to a device that has already been freed. Thus, callers should account for a fact that these functions may return *`ESP_ERR_NOT_FOUND`*.

Endpoints & Transfers USBH supports transfer to default (i.e., EP0) and non-default endpoints.

For non-default endpoints:

- A client must first allocate the endpoint by calling `usbh_ep_alloc()` which assigns a `usbh_ep_cb_t` callback and returns a `usbh_ep_handle_t` endpoint handle so that the endpoint can be referred to.
- A client can then enqueue a `urb_t` transfer to the endpoint by calling `usbh_ep_enqueue_urb()`.
- The `usbh_ep_cb_t` callback is called to indicate transfer completion
- The client must then dequeue the transfer using `usbh_ep_dequeue_urb()`

Default endpoints are owned and managed by the USBH, thus API for control transfers are different:

- EP0 is always allocated for each device, thus clients do no need to allocate EP0 or provide an endpoint callback.
- Clients call should call `usbh_dev_submit_ctrl_urb()` to submit a control transfer to a device's EP0.
- A `USBH_EVENT_CTRL_XFER` event will be propagated when the transfer is complete
- Control transfers do not need to be dequeued

USB Host Enumeration Driver (Enum)

Introduction The USB Host Enumeration Driver (henceforth referred to as Enum Driver) provides a software interface which abstracts away the USB device enumeration process. The Enum Driver provides a simple API to start, proceed, complete and cancel the enumeration of a particular device. Internally, the Enum Driver will handle all stages of the enumeration process such as requesting various descriptors and setting the device's configuration.

Requirements

USB Specification Requirements Chapter 9.1.2 of the USB 2.0 specification outlines some actions, when a USB device is attached to a powered port.

The design of the Enum Driver takes into consideration following actions:

- **The hub performs the required reset processing for the port.** `enum_config_t.enum_event_cb` call with `ENUM_EVENT_RESET_REQUIRED` event used for Hub Driver notification.
- **The host assigns a unique address to the USB device.** The Enum Driver keeps the device address value and assigns a unique address to the USB device, moving the device to the Address state.
- **The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use**. The Enum Driver reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
- **The host reads the configuration information from the device by reading each configuration zero to n-1, where n is the number of configurations.** This requirement simplified to reading only one configuration information from the device by reading configuration with default number, which could be selected by `enum_config_t.enum_filter_cb` call.
- **The host assigns a configuration value to the device.** The Enum Driver assigns a configuration value to the device.

备注： Typically, most USB devices only contain a single configuration. Thus, the Enum Driver defaults to selecting the configuration with *bConfigurationValue = 1*.

If users would like to select a different configuration, the Enum Driver provides an enumeration filter callback feature (enabled via *CONFIG_USB_HOST_ENABLE_ENUM_FILTER_CALLBACK*). This callback is called during the enumeration process and allows users to decide what devices to enumerate, and which `bConfigurationValue` to use.

备注： For more detailed information about Bus enumeration, please refer to USB 2.0 Specification > Chapter 9.1.2 **Bus Enumeration**.

Host Stack Requirements In addition to the USB 2.0 specification requirements, the Enum Driver also takes into consideration the requirements set for the overall Host Stack (see *USB Host Maintainers Notes (Design Guidelines)*):

- Enum Driver must not instantiate any tasks/threads
- Enum Driver must be event driven, providing event callbacks and an event processing function
- Enum Driver must use only API from underlying layer (USBH)

Implementation & Usage

Host Stack Interaction The Enum Driver takes place between USB Host layer and USBH layer with a possibility to select configuration with `enum_config_t.enum_event_cb` callback provided on Enum Driver installation.

Events & Processing The Enum Driver is completely event driven and all event handling is done via the `enum_process()` function. The `enum_config_t.proc_req_cb` callback provided on Enum Driver installation will be called when processing is required. Typically, `enum_process()` will be called from a shared USB Host stack thread/task.

The Enum Driver exposes the following event callbacks:

- `enum_event_cb_t` used to indicate various events regarding an enumeration process. This callback is called from the context of `enum_process()`

The Enum Driver exposes the following events:

- `ENUM_EVENT_STARTED` Enumeration process has been started
- `ENUM_EVENT_RESET_REQUIRED` Enumeration process requires device reset
- `ENUM_EVENT_COMPLETED` Enumeration process has been completed
- `ENUM_EVENT_CANCELED` Enumeration process has been canceled (due to internal error or via `enum_cancel()` function call)

Device Enumeration The USB device enumeration process implemented in the Enum Driver is mostly based same process in the Windows USB stack. The Enum Driver's enumeration process involves the following steps:

1. First device descriptor request (to obtain the MPS of EP0)
2. Second device reset (to workaround some non-compliant devices)
3. Set device address
4. Second device descriptor request (to obtain and store the full device descriptor)
5. Configuration descriptor request (to obtain the full configuration descriptor of selected configuration)
6. Language ID Table request (checks to see if en-US is supported)
7. Manufacturer string descriptor request
8. Product string descriptor request
9. Serial number string descriptor request
10. Set configuration request (sets the device to target configuration number)

备注： String descriptors are optional. If a device does not support string descriptors, these stages could be omitted.

Enumeration Stages The Enum Driver splits the enumeration process into multiple stages which are executed linearly. Depending on the connected device, some stages (such as fetching the string descriptors) can be skipped. When a stage completes, a call to the `enum_config_t.proc_req_cb` callback must be made to trigger a subsequent call of `enum_process()`.The subsequent call of `enum_process()` will then select and execute the next stage of enumeration. Stage completion can trigger the `enum_config_t.proc_req_cb` callback in one of the following ways:

- Inside the control transfer completion callback (for stages that send a control transfer)
- Direct call to `enum_config_t.proc_req_cb` (for stages that don't need to wait for any event)
- Inside `enum_proceed()` (for stages that require some action to be carried out outside the Enum Driver)

Any control transfer made during enumeration is split into two stages, where the first stage executes the transfer and the second stage (suffixed with `_CHECK`) will check the results of the transfers.

When requesting a variable length descriptors (e.g., configuration or string descriptors), the request is split into two control transfers. The first control transfer is fixed in length which only reads the header of the descriptor. The `bLength` field of the descriptor's header indicates the full length of the entire descriptor and is used to set the size of the second transfer which fetches the entire descriptor. As a result, any request for a variable length descriptor is split into four stages:

- Get short **ANY** descriptor (prefixed with `GET_SHORT_...`)
- Check short **ANY** descriptor (prefixed with `CHECK_SHORT_...`)
- Get full **ANY** descriptor (prefixed with `GET_FULL_...`)
- Check full **ANY** descriptor (prefixed with `CHECK_FULL_...`)

备注： Retrieving the Device Descriptor is an exception here because the second reset is taken place after retrieving short Device Descriptor.

Cancel Enumeration In some cases (such as a device disconnection), an ongoing enumeration process may need to be cancelled. An ongoing enumeration can be cancelled (regardless of its current stage) by calling `enum_cancel()` which will change the enumeration process's current stage to `ENUM_STAGE_CANCEL`.

On the next call to `enum_process`, the Enum Driver will execute the `ENUM_STAGE_CANCEL` which does the following:

- releases the device's enumeration lock.
- frees all resources related to the current device.
- propagates the `ENUM_EVENT_CANCELED` event.

USB Host External Hub Driver (Ext Hub)

Introduction The External Hub Driver (henceforth referred to as Ext Hub Driver)

Requirements

USB Specification Requirements Chapter 11 of the USB 2.0 specification outlines some aspects, when a USB Hub device is attached to a powered port.

The design of the Ext Driver takes into consideration the following:

- **Connectivity behavior**
- **Power management**
- **Device connect/disconnect detection**
- **Bus fault detection and recovery**
- **High-, full-, and low-speed device support**

备注： For more detailed information, please refer to USB 2.0 Specification > Chapter 11.1 **Overview**.

Host Stack Requirements In addition to the USB 2.0 specification requirements, the Ext Hub Driver also takes into consideration the requirements set for the overall Host Stack (see *USB Host Maintainers Notes (Design Guidelines)*):

- Ext Hub Driver must not instantiate any tasks/threads
- Ext Hub Driver must be event driven, providing event callbacks and an event processing function
- Ext Hub Driver must use only API from underlying layer (USBH)

Implementation & Usage

Host Stack Interaction The Ext Hub Driver takes place between USB Host layer and USBH layer, next to the Hub Driver. The Hub Driver and the Ext Hub Driver were split into two Drivers to achieve the goal of logic distinguishing between root Hub and external Hub.

Device handling The Ext Hub Driver can be installed via `ext_hub_install()` call and uninstalled via `ext_hub_uninstall()` call. After installation the Ext Hub provides the following APIs for external Hub addition and removal:

- `ext_hub_new_dev()` which will verify the device class (*HUB_CLASSCODE (09H)*) and, if the device has the Hub class, the Ext Hub Driver:
  - allocates a new device object
  - adds it to the external device pool
  - starts the process of Hub configuration (retrieving Hub Descriptor, Device status and Hub status)
- `ext_hub_dev_gone()` which will verify the device in the Ext Hub Driver list and start the process of external Hub device removing.

Events & Processing The Ext Hub Driver is completely event driven and all event handling is done via the `ext_hub_process()` function. The `ext_hub_config_t.proc_req_cb` callback provided on the Ext Hub Driver installation will be called when processing is required. Typically, `ext_hub_process()` will be called from the Hub Driver `hub_process()` processing function.

The Ext Hub Driver does not expose any event callback.

Todo:

- USB Host Maintainers Notes (HAL & LL)
- USB Host Maintainers Notes (HCD)
- USB Host Maintainers Notes (Hub)
- USB Host Maintainers Notes (USB Host Library)

Introduction The ESP-IDF USB Host Stack allows the ESP32-P4 to operate as a USB Host. Operating as a USB Host allows the ESP32-P4 to communicate with a wide range of USB devices. However, most USB Host Stack implementations do not run on embedded hardware (i.e., runs on PCs and smartphones), thus have comparatively more resources (i.e., memory and CPU speed).

The implementation of the ESP-IDF USB Host Stack (henceforth referred to as the Host Stack) takes into account the embedded nature of the ESP32-P4 which is reflected in various aspects of the Host Stack's design.

Features & Limitations **The Host Stack currently supports the following notable features:**

- Supports HS (High Speed)
- Supports FS (Full Speed) and LS (Low Speed) devices
- Supports all transfer types (Control, Bulk, Isochronous, and Interrupt)
- Automatically enumerates connected devices
- Allows multiple class drivers (i.e., Clients of the USB Host Library) to run simultaneously and share the same device (i.e., composite devices)

The Host Stack currently has the following notable limitations:

- No Hub support (currently only supports a single device)

本部分的 API 示例代码存放在 ESP-IDF 示例项目的 peripherals 目录下。

2.6 项目配置

2.6.1 简介

ESP-IDF 使用基于 kconfiglib 的 esp-idf-kconfig 包，而 kconfiglib 是 Kconfig 系统的 Python 扩展。Kconfig 提供了编译时的项目配置机制，以及多种类型的配置选项（如整数、字符串和布尔值等）。Kconfig 文件指定了选项之间的依赖关系、默认值、组合方式等。

了解所有可用功能，请查看 Kconfig 和 kconfiglib 扩展。

2.6.2 项目配置菜单

应用程序开发人员可以通过 `idf.py menuconfig` 构建目标，在终端中打开项目配置菜单。

更新后，此配置将保存在项目根目录的 `sdkconfig` 文件中。借助 `sdkconfig`，应用程序构建目标将在构建目录中生成 `sdkconfig.h` 文件，并使得 `sdkconfig` 选项可用于项目构建系统和源文件。

2.6.3 使用 sdkconfig.defaults

在某些情况下，例如 `sdkconfig` 文件处于版本控制状态时，构建系统可能会不便于更改 `sdkconfig` 文件。要避免上述情况，可以在构建系统中创建 `sdkconfig.defaults` 文件。该文件可以手动或自动创建，且构建系统永远不会对其进行更改。该文件包含所有不同于默认选项的重要选项，其格式与 `sdkconfig` 文件格式相同。如果用户记得所有已更改的配置，则可以手动创建 `sdkconfig.defaults`，或运行 `idf.py save-defconfig` 命令来自动生成此文件。

`sdkconfig.defaults` 创建后，用户可以删除 `sdkconfig` 或将其添加到版本控制系统的忽略列表中（例如 `git` 的 `.gitignore` 文件）。项目构建目标将自动创建 `sdkconfig` 文件，填充 `sdkconfig.defaults` 文件中的设置，并将其他设置配置为默认值。请注意，构建时 `sdkconfig.defaults` 中的设置不会覆盖 `sdkconfig` 的已有设置。了解更多信息，请查看自定义 *sdkconfig* 的默认值。

2.6.4 Kconfig 格式规定

Kconfig 文件的格式规定如下：

- 在所有菜单中，选项名称的前缀需保持一致。目前，前缀长度应为至少 3 个字符。
- 每级采用 4 个空格的缩进方式，子项需比父项多缩进一级。例如，`menu` 缩进 0 个空格，`menu` 中的 `config` 则缩进 4 个空格，`config` 中的 `help` 缩进 8 个空格，`help` 下的文本缩进 12 个空格。
- 行末不得出现尾随空格。
- 选项最长为 50 个字符。
- 每行最长为 120 个字符。

备注： 菜单中不同配置的 `help` 小节默认视为 reStructuredText 格式，以便生成相应选项的参考文档。

格式检查器

esp-idf-kconfig 软件包中的 `kconfcheck` 工具可以检查 Kconfig 文件是否符合上述格式规定。检查器会检查作为参数给出的所有 Kconfig 和 `Kconfig.projbuild` 文件，并生成一个后缀为 `.new` 的新文件，如有格式错误，便会在此文件中提供修改建议。注意，检查器不能解决所有格式问题，开发人员仍需终审并修改文件，使其通过测试。例如，在没有其他误导性格式的情况下，检查器能够更正缩进，但无法为菜单内选项提供常用的前缀。

esp-idf-kconfig 软件包可以在 ESP-IDF 环境中使用。运行命令 python -m kconfcheck <path_to_kconfig_file> 即可调用检查工具。

如需了解更多内容，请参考 esp-idf-kconfig 相关文档。

2.6.5 Kconfig 选项的向后兼容性

标准 Kconfig 工具会忽略 sdkconfig 中的未知选项。因此，如果开发人员对某些选项进行了自定义设置，但这些选项在 ESP-IDF 新版本中重新命名，标准 Kconfig 工具将忽略原有设置。以下功能可以避免上述情况：

1. 工具链使用 kconfgen 预处理 sdkconfig 文件。例如，menuconfig 会读取这些文件，从而保留旧选项设置。
2. kconfgen 递归查找 ESP-IDF 目录中所有包含新旧 Kconfig 选项名称的 sdkconfig.rename 文件。在 sdkconfig 文件中，新选项将替换旧选项。针对单个目标的重命名可以放在特定目标的重命名文件 sdkconfig.rename.TARGET 中，其中 TARGET 是目标名称，例如 sdkconfig.rename.esp32s2。
3. kconfgen 通过添加兼容性语句列表（即经过修改后，将旧选项的值设置为新选项的值），后处理 sdkconfig 文件，并生成所有构建结果（sdkconfig.h、sdkconfig.cmake 以及 auto.conf）。如果用户在其代码中仍然使用旧选项，此举可以防止用户代码出现问题。
4. kconfgen 会自动生成*Deprecated options and their replacements*。

2.6.6 配置选项参考

以下小节包含由 Kconfig 文件自动生成的 ESP-IDF 可用选项列表。请注意，由于所选选项不同，下列某些选项可能在 menuconfig 界面中默认不可见。

按照惯例，所有选项名称均为大写字母加下划线。当 Kconfig 生成 sdkconfig 和 sdkconfig.h 文件时，选项名称会以 CONFIG_ 为前缀。因此，如果 Kconfig 文件定义了 ENABLE_FOO 选项且 menuconfig 中选择了该选项，则 sdkconfig 和 sdkconfig.h 文件也将定义 CONFIG_ENABLE_FOO。在以下小节中，选项名称也以 CONFIG_ 为前缀，与源代码相同。

Build type

Contains:

- *CONFIG_APP_BUILD_TYPE*
- *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*
- *CONFIG_APP_REPRODUCIBLE_BUILD*
- *CONFIG_APP_NO_BLOBS*

CONFIG_APP_BUILD_TYPE

Application build type

Found in: Build type

Select the way the application is built.

By default, the application is built as a binary file in a format compatible with the ESP-IDF bootloader. In addition to this application, 2nd stage bootloader is also built. Application and bootloader binaries can be written into flash and loaded/executed from there.

Another option, useful for only very small and limited applications, is to only link the .elf file of the application, such that it can be loaded directly into RAM over JTAG or UART. Note that since IRAM and DRAM sizes are very limited, it is not possible to build any complex application this way. However for some kinds of testing and debugging, this option may provide faster iterations, since the application does not need to be written into flash.

Note: when APP_BUILD_TYPE_RAM is selected and loaded with JTAG, ESP-IDF does not contain all the startup code required to initialize the CPUs and ROM memory (data/bss). Therefore it is necessary to execute a bit of ROM code prior to executing the application. A gdbinit file may look as follows (for ESP32):

> # Connect to a running instance of OpenOCD target remote :3333 # Reset and halt the target mon reset halt # Run to a specific point in ROM code, # where most of initialization is complete. thb *0x40007d54 c # Load the application into RAM load # Run till app_main tb app_main c

Execute this gdbinit file as follows:

> xtensa-esp32-elf-gdb build/app-name.elf -x gdbinit

Example gdbinit files for other targets can be found in tools/test_apps/system/gdb_loadable_elf/

When loading the BIN with UART, the ROM will jump to ram and run the app after finishing the ROM startup code, so there's no additional startup initialization required. You can use the *load_ram* in esptool.py to load the generated .bin file into ram and execute.

Example: esptool.py --chip {chip} -p {port} -b {baud} --no-stub load_ram {app.bin}

Recommended sdkconfig.defaults for building loadable ELF files is as follows. CONFIG_APP_BUILD_TYPE_RAM is required, other options help reduce application memory footprint.

> CONFIG_APP_BUILD_TYPE_RAM=y CONFIG_VFS_SUPPORT_TERMIOS= CONFIG_NEWLIB_NANO_FORMAT=y CONFIG_ESP_SYSTEM_PANIC_PRINT_HALT=y CONFIG_ESP_DEBUG_STUBS_ENABLE= CONFIG_ESP_ERR_TO_NAME_LOOKUP=

Available options:

- Default (binary application + 2nd stage bootloader) (CONFIG_APP_BUILD_TYPE_APP_2NDBOOT)
- Build app runs entirely in RAM (EXPERIMENTAL) (CONFIG_APP_BUILD_TYPE_RAM)

CONFIG_APP_BUILD_TYPE_PURE_RAM_APP

Build app without SPI_FLASH/PSRAM support (saves ram)

Found in: Build type

If this option is enabled, external memory and related peripherals, such as Cache, MMU, Flash and PSRAM, won't be initialized. Corresponding drivers won't be introduced either. Components that depend on the spi_flash component will also be unavailable, such as app_update, etc. When this option is enabled, about 26KB of RAM space can be saved.

CONFIG_APP_REPRODUCIBLE_BUILD

Enable reproducible build

Found in: Build type

If enabled, all date, time, and path information would be eliminated. A .gdbinit file would be create automatically. (or will be append if you have one already)

Default value:

- No (disabled)

CONFIG_APP_NO_BLOBS

No Binary Blobs

Found in: Build type

If enabled, this disables the linking of binary libraries in the application build. Note that after enabling this Wi-Fi/Bluetooth will not work.

Default value:

- No (disabled)

Bootloader config

Contains:

- *CONFIG_BOOTLOADER_LOG_LEVEL*
- *Bootloader manager*
- *CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION*
- *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*
- *CONFIG_BOOTLOADER_REGION_PROTECTION_ENABLE*
- *CONFIG_BOOTLOADER_APP_TEST*
- *CONFIG_BOOTLOADER_FACTORY_RESET*
- *CONFIG_BOOTLOADER_HOLD_TIME_GPIO*
- *CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC*
- *Serial Flash Configurations*
- *CONFIG_BOOTLOADER_SKIP_VALIDATE_ALWAYS*
- *CONFIG_BOOTLOADER_SKIP_VALIDATE_ON_POWER_ON*
- *CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP*
- *CONFIG_BOOTLOADER_WDT_ENABLE*
- *CONFIG_BOOTLOADER_VDDSDIO_BOOST*

Bootloader manager

Contains:

- *CONFIG_BOOTLOADER_PROJECT_VER*
- *CONFIG_BOOTLOADER_COMPILE_TIME_DATE*

CONFIG_BOOTLOADER_COMPILE_TIME_DATE

Use time/date stamp for bootloader

Found in: Bootloader config > Bootloader manager

If set, then the bootloader will be built with the current time/date stamp. It is stored in the bootloader description structure. If not set, time/date stamp will be excluded from bootloader image. This can be useful for getting the same binary image files made from the same source, but at different times.

CONFIG_BOOTLOADER_PROJECT_VER

Project version

Found in: Bootloader config > Bootloader manager

Project version. It is placed in ”version” field of the esp_bootloader_desc structure. The type of this field is ”uint32_t”.

Range:

- from 0 to 4294967295

Default value:

- 1

CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION

Bootloader optimization Level

Found in: Bootloader config

This option sets compiler optimization level (gcc -O argument) for the bootloader.

- The default "Size" setting will add the -0s flag to CFLAGS.
- The "Debug" setting will add the -Og flag to CFLAGS.
- The "Performance" setting will add the -O2 flag to CFLAGS.

Note that custom optimization levels may be unsupported.

Available options:

- Size (-Os) (CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION_SIZE)
- Debug (-Og) (CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION_DEBUG)
- Optimize for performance (-O2) (CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION_PERF)
- Debug without optimization (-O0) (Deprecated, will be removed in IDF v6.0) (CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION_NONE)

CONFIG_BOOTLOADER_LOG_LEVEL

Bootloader log verbosity

Found in: Bootloader config

Specify how much output to see in bootloader logs.

Available options:

- No output (CONFIG_BOOTLOADER_LOG_LEVEL_NONE)
- Error (CONFIG_BOOTLOADER_LOG_LEVEL_ERROR)
- Warning (CONFIG_BOOTLOADER_LOG_LEVEL_WARN)
- Info (CONFIG_BOOTLOADER_LOG_LEVEL_INFO)
- Debug (CONFIG_BOOTLOADER_LOG_LEVEL_DEBUG)
- Verbose (CONFIG_BOOTLOADER_LOG_LEVEL_VERBOSE)

Serial Flash Configurations

Contains:

- *CONFIG_BOOTLOADER_FLASH_DC_AWARE*
- *CONFIG_BOOTLOADER_CACHE_32BIT_ADDR_QUAD_FLASH*
- *CONFIG_BOOTLOADER_FLASH_XMC_SUPPORT*

CONFIG_BOOTLOADER_FLASH_DC_AWARE

Allow app adjust Dummy Cycle bits in SPI Flash for higher frequency (READ HELP FIRST)

Found in: Bootloader config > Serial Flash Configurations

This will force 2nd bootloader to be loaded by DOUT mode, and will restore Dummy Cycle setting by resetting the Flash

CONFIG_BOOTLOADER_FLASH_XMC_SUPPORT

Enable the support for flash chips of XMC (READ DOCS FIRST)

Found in: Bootloader config > Serial Flash Configurations

Perform the startup flow recommended by XMC. Please consult XMC for the details of this flow. XMC chips will be forbidden to be used, when this option is disabled.

DON'T DISABLE THIS UNLESS YOU KNOW WHAT YOU ARE DOING.

comment "Features below require specific hardware (READ DOCS FIRST!)"

Default value:

- Yes (enabled)

CONFIG_BOOTLOADER_CACHE_32BIT_ADDR_QUAD_FLASH

Enable cache access to 32-bit-address (over 16MB) range of SPI Flash (READ DOCS FIRST)

Found in: Bootloader config > Serial Flash Configurations

Enabling this option allows the CPU to access 32-bit-address flash beyond 16M range. 1. This option only valid for 4-line flash. Octal flash doesn't need this. 2. This option is experimental, which means it can' t use on all flash chips stable, for more information, please contact Espressif Business support.

CONFIG_BOOTLOADER_VDDSDIO_BOOST

VDDSDIO LDO voltage

Found in: Bootloader config

If this option is enabled, and VDDSDIO LDO is set to 1.8V (using eFuse or MTDI bootstrapping pin), bootloader will change LDO settings to output 1.9V instead. This helps prevent flash chip from browning out during flash programming operations.

This option has no effect if VDDSDIO is set to 3.3V, or if the internal VDDSDIO regulator is disabled via eFuse.

Available options:

- 1.8V (CONFIG_BOOTLOADER_VDDSDIO_BOOST_1_8V)
- 1.9V (CONFIG_BOOTLOADER_VDDSDIO_BOOST_1_9V)

CONFIG_BOOTLOADER_FACTORY_RESET

GPIO triggers factory reset

Found in: Bootloader config

Allows to reset the device to factory settings: - clear one or more data partitions; - boot from "factory" partition. The factory reset will occur if there is a GPIO input held at the configured level while device starts up. See settings below.

Default value:

- No (disabled)

CONFIG_BOOTLOADER_NUM_PIN_FACTORY_RESET

Number of the GPIO input for factory reset

Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET

The selected GPIO will be configured as an input with internal pull-up enabled (note that on some SoCs. not all pins have an internal pull-up, consult the hardware datasheet for details.) To trigger a factory reset, this GPIO must be held high or low (as configured) on startup.

Default value:

- 4 if *CONFIG_BOOTLOADER_FACTORY_RESET*

CONFIG_BOOTLOADER_FACTORY_RESET_PIN_LEVEL

Factory reset GPIO level

Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET

Pin level for factory reset, can be triggered on low or high.

Available options:

- Reset on GPIO low (CONFIG_BOOTLOADER_FACTORY_RESET_PIN_LOW)
- Reset on GPIO high (CONFIG_BOOTLOADER_FACTORY_RESET_PIN_HIGH)

CONFIG_BOOTLOADER_OTA_DATA_ERASE

Clear OTA data on factory reset (select factory partition)

Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET

The device will boot from ”factory” partition (or OTA slot 0 if no factory partition is present) after a factory reset.

CONFIG_BOOTLOADER_DATA_FACTORY_RESET

Comma-separated names of partitions to clear on factory reset

Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET

Allows customers to select which data partitions will be erased while factory reset.

Specify the names of partitions as a comma-delimited with optional spaces for readability. (Like this: ”nvs, phy_init, ...”) Make sure that the name specified in the partition table and here are the same. Partitions of type ”app” cannot be specified here.

Default value:

- ”nvs” if *CONFIG_BOOTLOADER_FACTORY_RESET*

CONFIG_BOOTLOADER_APP_TEST

GPIO triggers boot from test app partition

Found in: Bootloader config

Allows to run the test app from ”TEST” partition. A boot from ”test” partition will occur if there is a GPIO input pulled low while device starts up. See settings below.

CONFIG_BOOTLOADER_NUM_PIN_APP_TEST

Number of the GPIO input to boot TEST partition

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_TEST

The selected GPIO will be configured as an input with internal pull-up enabled. To trigger a test app, this GPIO must be pulled low on reset. After the GPIO input is deactivated and the device reboots, the old application will boot. (factory or OTA[x]). Note that GPIO34-39 do not have an internal pullup and an external one must be provided.

Range:

- from 0 to 39 if *CONFIG_BOOTLOADER_APP_TEST*

Default value:

- 18 if *CONFIG_BOOTLOADER_APP_TEST*

CONFIG_BOOTLOADER_APP_TEST_PIN_LEVEL

App test GPIO level

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_TEST

Pin level for app test, can be triggered on low or high.

Available options:

- Enter test app on GPIO low (CONFIG_BOOTLOADER_APP_TEST_PIN_LOW)
- Enter test app on GPIO high (CONFIG_BOOTLOADER_APP_TEST_PIN_HIGH)

CONFIG_BOOTLOADER_HOLD_TIME_GPIO

Hold time of GPIO for reset/test mode (seconds)

Found in: Bootloader config

The GPIO must be held low continuously for this period of time after reset before a factory reset or test partition boot (as applicable) is performed.

Default value:

- 5 if *CONFIG_BOOTLOADER_FACTORY_RESET* || *CONFIG_BOOTLOADER_APP_TEST*

CONFIG_BOOTLOADER_REGION_PROTECTION_ENABLE

Enable protection for unmapped memory regions

Found in: Bootloader config

Protects the unmapped memory regions of the entire address space from unintended accesses. This will ensure that an exception will be triggered whenever the CPU performs a memory operation on unmapped regions of the address space.

Default value:

- Yes (enabled)

CONFIG_BOOTLOADER_WDT_ENABLE

Use RTC watchdog in start code

Found in: Bootloader config

Tracks the execution time of startup code. If the execution time is exceeded, the RTC_WDT will restart system. It is also useful to prevent a lock up in start code caused by an unstable power source. NOTE: Tracks the execution time starts from the bootloader code - re-set timeout, while selecting the source for slow_clk - and ends calling app_main. Re-set timeout is needed due to WDT uses a SLOW_CLK clock source. After changing a frequency slow_clk a time of WDT needs to re-set for new frequency. slow_clk depends on RTC_CLK_SRC (INTERNAL_RC or EXTERNAL_CRYSTAL).

Default value:

- Yes (enabled)

CONFIG_BOOTLOADER_WDT_DISABLE_IN_USER_CODE

Allows RTC watchdog disable in user code

Found in: Bootloader config > CONFIG_BOOTLOADER_WDT_ENABLE

If this option is set, the ESP-IDF app must explicitly reset, feed, or disable the rtc_wdt in the app's own code. If this option is not set (default), then rtc_wdt will be disabled by ESP-IDF before calling the app_main() function.

Use function wdt_hal_feed() for resetting counter of RTC_WDT. For esp32/s2 you can also use rtc_wdt_feed().

Use function wdt_hal_disable() for disabling RTC_WDT. For esp32/s2 you can also use rtc_wdt_disable().

Default value:

- No (disabled)

CONFIG_BOOTLOADER_WDT_TIME_MS

Timeout for RTC watchdog (ms)

Found in: Bootloader config > CONFIG_BOOTLOADER_WDT_ENABLE

Verify that this parameter is correct and more then the execution time. Pay attention to options such as reset to factory, trigger test partition and encryption on boot - these options can increase the execution time. Note: RTC_WDT will reset while encryption operations will be performed.

Range:

- from 0 to 120000

Default value:

- 9000

CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE

Enable app rollback support

Found in: Bootloader config

After updating the app, the bootloader runs a new app with the "ESP_OTA_IMG_PENDING_VERIFY" state set. This state prevents the re-run of this app. After the first boot of the new app in the user code, the function should be called to confirm the operability of the app or vice versa about its non-operability. If the app is working, then it is marked as valid. Otherwise, it is marked as not valid and rolls back to the previous working app. A reboot is performed, and the app is booted before the software update. Note: If during the first boot a new app the power goes out or the WDT works, then roll back will happen. Rollback is possible only between the apps with the same security versions.

Default value:

- No (disabled)

CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK

Enable app anti-rollback support

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE

This option prevents rollback to previous firmware/application image with lower security version.

Default value:

- No (disabled) if *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*

CONFIG_BOOTLOADER_APP_SECURE_VERSION

eFuse secure version of app

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK

The secure version is the sequence number stored in the header of each firmware. The security version is set in the bootloader, version is recorded in the eFuse field as the number of set ones. The allocated number of bits in the efuse field for storing the security version is limited (see BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD option).

Bootloader: When bootloader selects an app to boot, an app is selected that has a security version greater or equal that recorded in eFuse field. The app is booted with a higher (or equal) secure version.

The security version is worth increasing if in previous versions there is a significant vulnerability and their use is not acceptable.

Your partition table should has a scheme with ota_0 + ota_1 (without factory).

Default value:

- 0 if *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD

Size of the efuse secure version field

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK

The size of the efuse secure version field. Its length is limited to 32 bits for ESP32 and 16 bits for ESP32-S2. This determines how many times the security version can be increased.

Range:

- from 1 to 16 if *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

Default value:

- 16 if *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

CONFIG_BOOTLOADER_EFUSE_SECURE_VERSION_EMULATE

Emulate operations with efuse secure version(only test)

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK

This option allows to emulate read/write operations with all eFuses and efuse secure version. It allows to test anti-rollback implementation without permanent write eFuse bits. There should be an entry in partition table with following details: *emul_efuse, data, efuse, , 0x2000*.

This option enables: EFUSE_VIRTUAL and EFUSE_VIRTUAL_KEEP_IN_FLASH.

Default value:

- No (disabled) if *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP

Skip image validation when exiting deep sleep

Found in: Bootloader config

This option disables the normal validation of an image coming out of deep sleep (checksums, SHA256, and signature). This is a trade-off between wakeup performance from deep sleep, and image integrity checks.

Only enable this if you know what you are doing. It should not be used in conjunction with using deep_sleep() entry and changing the active OTA partition as this would skip the validation upon first load of the new OTA partition.

It is possible to enable this option with Secure Boot if ”allow insecure options” is enabled, however it’s strongly recommended to NOT enable it as it may allow a Secure Boot bypass.

Default value:

- No (disabled) if *CONFIG_SECURE_BOOT* && *CONFIG_SECURE_BOOT_INSECURE*

CONFIG_BOOTLOADER_SKIP_VALIDATE_ON_POWER_ON

Skip image validation from power on reset (READ HELP FIRST)

Found in: Bootloader config

Some applications need to boot very quickly from power on. By default, the entire app binary is read from flash and verified which takes up a significant portion of the boot time.

Enabling this option will skip validation of the app when the SoC boots from power on. Note that in this case it's not possible for the bootloader to detect if an app image is corrupted in the flash, therefore it's not possible to safely fall back to a different app partition. Flash corruption of this kind is unlikely but can happen if there is a serious firmware bug or physical damage.

Following other reset types, the bootloader will still validate the app image. This increases the chances that flash corruption resulting in a crash can be detected following soft reset, and the bootloader will fall back to a valid app image. To increase the chances of successfully recovering from a flash corruption event, keep the option BOOTLOADER_WDT_ENABLE enabled and consider also enabling BOOTLOADER_WDT_DISABLE_IN_USER_CODE - then manually disable the RTC Watchdog once the app is running. In addition, enable both the Task and Interrupt watchdog timers with reset options set.

Default value:

- No (disabled)

CONFIG_BOOTLOADER_SKIP_VALIDATE_ALWAYS

Skip image validation always (READ HELP FIRST)

Found in: Bootloader config

Selecting this option prevents the bootloader from ever validating the app image before booting it. Any flash corruption of the selected app partition will make the entire SoC unbootable.

Although flash corruption is a very rare case, it is not recommended to select this option. Consider selecting "Skip image validation from power on reset" instead. However, if boot time is the only important factor then it can be enabled.

Default value:

- No (disabled)

CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC

Reserve RTC FAST memory for custom purposes

Found in: Bootloader config

This option allows the customer to place data in the RTC FAST memory, this area remains valid when rebooted, except for power loss. This memory is located at a fixed address and is available for both the bootloader and the application. (The application and bootloader must be compiled with the same option). The RTC FAST memory has access only through PRO_CPU.

Default value:

- No (disabled)

CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC_IN_CRC

Include custom memory in the CRC calculation

Found in: Bootloader config > CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC

This option allows the customer to use the legacy bootloader behavior when the RTC FAST memory CRC calculation takes place. When this option is enabled, the allocated user custom data will be taken into account in the CRC calculation. This means that any change to the custom data would need a CRC update to prevent the bootloader from marking this data as corrupted. If this option is disabled, the custom data will not be taken into account when calculating the RTC FAST memory CRC. The user

custom data can be changed freely, without the need to update the CRC. THIS OPTION MUST BE THE SAME FOR BOTH THE BOOTLOADER AND THE APPLICATION BUILDS.

Default value:

- No (disabled) if *CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC*

CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC_SIZE

Size in bytes for custom purposes

Found in: Bootloader config > *CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC*

This option reserves in RTC FAST memory the area for custom purposes. If you want to create your own bootloader and save more information in this area of memory, you can increase it. It must be a multiple of 4 bytes. This area (rtc_retain_mem_t) is reserved and has access from the bootloader and an application.

Default value:

- 0 if *CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC*

Security features

Contains:

- *CONFIG_SECURE_BOOT_INSECURE*
- *CONFIG_SECURE_SIGNED_APPS_SCHEME*
- *CONFIG_SECURE_SIGNED_ON_BOOT_NO_SECURE_BOOT*
- *CONFIG_SECURE_FLASH_CHECK_ENC_EN_IN_APP*
- *CONFIG_SECURE_BOOT_ECDSA_KEY_LEN_SIZE*
- *CONFIG_SECURE_BOOT_ENABLE_AGGRESSIVE_KEY_REVOKE*
- *CONFIG_SECURE_FLASH_ENC_ENABLED*
- *CONFIG_SECURE_BOOT*
- *CONFIG_SECURE_FLASH_ENCRYPT_ONLY_IMAGE_LEN_IN_APP_PART*
- *CONFIG_SECURE_BOOT_FLASH_BOOTLOADER_DEFAULT*
- *CONFIG_SECURE_BOOTLOADER_KEY_ENCODING*
- *Potentially insecure options*
- *CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT*
- *CONFIG_SECURE_BOOT_VERIFICATION_KEY*
- *CONFIG_SECURE_BOOTLOADER_MODE*
- *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES*
- *CONFIG_SECURE_UART_ROM_DL_MODE*
- *CONFIG_SECURE_SIGNED_ON_UPDATE_NO_SECURE_BOOT*

CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT

Require signed app images

Found in: Security features

Require apps to be signed to verify their integrity.

This option uses the same app signature scheme as hardware secure boot, but unlike hardware secure boot it does not prevent the bootloader from being physically updated. This means that the device can be secured against remote network access, but not physical access. Compared to using hardware Secure Boot this option is much simpler to implement.

CONFIG_SECURE_SIGNED_APPS_SCHEME

App Signing Scheme

Found in: Security features

Select the Secure App signing scheme. Depends on the Chip Revision. There are two secure boot versions:

1. **Secure boot V1**
   - Legacy custom secure boot scheme. Supported in ESP32 SoC.
2. **Secure boot V2**
   - RSA based secure boot scheme. Supported in ESP32-ECO3 (ESP32 Chip Revision 3 onwards), ESP32-S2, ESP32-C3, ESP32-S3 SoCs.
   - ECDSA based secure boot scheme. Supported in ESP32-C2 SoC.

Available options:

- ECDSA (CONFIG_SECURE_SIGNED_APPS_ECDSA_SCHEME)
  Embeds the ECDSA public key in the bootloader and signs the application with an ECDSA key. Refer to the documentation before enabling.
- RSA (CONFIG_SECURE_SIGNED_APPS_RSA_SCHEME)
  Appends the RSA-3072 based Signature block to the application. Refer to <Secure Boot Version 2 documentation link> before enabling.
- ECDSA (V2) (CONFIG_SECURE_SIGNED_APPS_ECDSA_V2_SCHEME)
  For Secure boot V2 (e.g., ESP32-C2 SoC), appends ECDSA based signature block to the application. Refer to documentation before enabling.

CONFIG_SECURE_BOOT_ECDSA_KEY_LEN_SIZE

ECDSA key size

Found in: Security features

Select the ECDSA key size. Two key sizes are supported

- 192 bit key using NISTP192 curve
- 256 bit key using NISTP256 curve (Recommended)

The advantage of using 256 bit key is the extra randomness which makes it difficult to be bruteforced compared to 192 bit key. At present, both key sizes are practically implausible to bruteforce.

Available options:

- Using ECC curve NISTP192 (CONFIG_SECURE_BOOT_ECDSA_KEY_LEN_192_BITS)
- Using ECC curve NISTP256 (Recommended) (CONFIG_SECURE_BOOT_ECDSA_KEY_LEN_256_BITS)

CONFIG_SECURE_SIGNED_ON_BOOT_NO_SECURE_BOOT

Bootloader verifies app signatures

Found in: Security features

If this option is set, the bootloader will be compiled with code to verify that an app is signed before booting it.

If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option doesn't add significant security by itself so most users will want to leave it disabled.

Default value:

- No (disabled) if *CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT* && *CONFIG_SECURE_SIGNED_APPS_ECDSA_SCHEME*

CONFIG_SECURE_SIGNED_ON_UPDATE_NO_SECURE_BOOT

Verify app signature on update

Found in: Security features

If this option is set, any OTA updated apps will have the signature verified before being considered valid.

When enabled, the signature is automatically checked whenever the esp_ota_ops.h APIs are used for OTA updates, or esp_image_format.h APIs are used to verify apps.

If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option still adds significant security against network-based attackers by preventing spoofing of OTA updates.

Default value:

- Yes (enabled) if *CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT*

CONFIG_SECURE_BOOT

Enable hardware Secure Boot in bootloader (READ DOCS FIRST)

Found in: Security features

Build a bootloader which enables Secure Boot on first boot.

Once enabled, Secure Boot will not boot a modified bootloader. The bootloader will only load a partition table or boot an app if the data has a verified digital signature. There are implications for reflashing updated apps once secure boot is enabled.

When enabling secure boot, JTAG and ROM BASIC Interpreter are permanently disabled by default.

Default value:

- No (disabled)

CONFIG_SECURE_BOOT_VERSION

Select secure boot version

Found in: Security features > CONFIG_SECURE_BOOT

Select the Secure Boot Version. Depends on the Chip Revision. Secure Boot V2 is the new RSA / ECDSA based secure boot scheme.

- RSA based scheme is supported in ESP32 (Revision 3 onwards), ESP32-S2, ESP32-C3 (ECO3), ESP32-S3.
- ECDSA based scheme is supported in ESP32-C2 SoC.

Please note that, RSA or ECDSA secure boot is property of specific SoC based on its HW design, supported crypto accelerators, die-size, cost and similar parameters. Please note that RSA scheme has requirement for bigger key sizes but at the same time it is comparatively faster than ECDSA verification.

Secure Boot V1 is the AES based (custom) secure boot scheme supported in ESP32 SoC.

Available options:

- Enable Secure Boot version 1 (CONFIG_SECURE_BOOT_V1_ENABLED)
  Build a bootloader which enables secure boot version 1 on first boot. Refer to the Secure Boot section of the ESP-IDF Programmer's Guide for this version before enabling.
- Enable Secure Boot version 2 (CONFIG_SECURE_BOOT_V2_ENABLED)
  Build a bootloader which enables Secure Boot version 2 on first boot. Refer to Secure Boot V2 section of the ESP-IDF Programmer's Guide for this version before enabling.

CONFIG_SECURE_BOOTLOADER_MODE

Secure bootloader mode

Found in: Security features

Available options:

- One-time flash (CONFIG_SECURE_BOOTLOADER_ONE_TIME_FLASH)
  On first boot, the bootloader will generate a key which is not readable externally or by software. A digest is generated from the bootloader image itself. This digest will be verified on each subsequent boot.
  Enabling this option means that the bootloader cannot be changed after the first time it is booted.
- Reflashable (CONFIG_SECURE_BOOTLOADER_REFLASHABLE)
  Generate a reusable secure bootloader key, derived (via SHA-256) from the secure boot signing key.
  This allows the secure bootloader to be re-flashed by anyone with access to the secure boot signing key.
  This option is less secure than one-time flash, because a leak of the digest key from one device allows reflashing of any device that uses it.

CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES

Sign binaries during build

Found in: Security features

Once secure boot or signed app requirement is enabled, app images are required to be signed.

If enabled (default), these binary files are signed as part of the build process. The file named in "Secure boot private signing key" will be used to sign the image.

If disabled, unsigned app/partition data will be built. They must be signed manually using espsecure.py. Version 1 to enable ECDSA Based Secure Boot and Version 2 to enable RSA based Secure Boot. (for example, on a remote signing server.)

CONFIG_SECURE_BOOT_SIGNING_KEY

Secure boot private signing key

Found in: Security features > CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES

Path to the key file used to sign app images.

Key file is an ECDSA private key (NIST256p curve) in PEM format for Secure Boot V1. Key file is an RSA private key in PEM format for Secure Boot V2.

Path is evaluated relative to the project directory.

You can generate a new signing key by running the following command: espsecure.py generate_signing_key secure_boot_signing_key.pem

See the Secure Boot section of the ESP-IDF Programmer's Guide for this version for details.

Default value:

- "secure_boot_signing_key.pem" if *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES*

CONFIG_SECURE_BOOT_VERIFICATION_KEY

Secure boot public signature verification key

Found in: Security features

Path to a public key file used to verify signed images. Secure Boot V1: This ECDSA public key is compiled into the bootloader and/or app, to verify app images.

Key file is in raw binary format, and can be extracted from a PEM formatted private key using the espsecure.py extract_public_key command.

Refer to the Secure Boot section of the ESP-IDF Programmer's Guide for this version before enabling.

CONFIG_SECURE_BOOT_ENABLE_AGGRESSIVE_KEY_REVOKE

Enable Aggressive key revoke strategy

Found in: Security features

If this option is set, ROM bootloader will revoke the public key digest burned in efuse block if it fails to verify the signature of software bootloader with it. Revocation of keys does not happen when enabling secure boot. Once secure boot is enabled, key revocation checks will be done on subsequent boot-up, while verifying the software bootloader

This feature provides a strong resistance against physical attacks on the device.

NOTE: Once a digest slot is revoked, it can never be used again to verify an image This can lead to permanent bricking of the device, in case all keys are revoked because of signature verification failure.

Default value:

- No (disabled) if *CONFIG_SECURE_BOOT*

CONFIG_SECURE_BOOT_FLASH_BOOTLOADER_DEFAULT

Flash bootloader along with other artifacts when using the default flash command

Found in: Security features

When Secure Boot V2 is enabled, by default the bootloader is not flashed along with other artifacts like the application and the partition table images, i.e. bootloader has to be separately flashed using the command *idf.py bootloader flash*, whereas, the application and partition table can be flashed using the command *idf.py flash* itself. Enabling this option allows flashing the bootloader along with the other artifacts by invocation of the command *idf.py flash*.

If this option is enabled make sure that even the bootloader is signed using the correct secure boot key, otherwise the bootloader signature verification would fail, as hash of the public key which is present in the bootloader signature would not match with the digest stored into the efuses and thus the device will not be able to boot up.

Default value:

- No (disabled) if *CONFIG_SECURE_BOOT_V2_ENABLED* && *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES*

CONFIG_SECURE_BOOTLOADER_KEY_ENCODING

Hardware Key Encoding

Found in: Security features

In reflashable secure bootloader mode, a hardware key is derived from the signing key (with SHA-256) and can be written to eFuse with espefuse.py.

Normally this is a 256-bit key, but if 3/4 Coding Scheme is used on the device then the eFuse key is truncated to 192 bits.

This configuration item doesn't change any firmware code, it only changes the size of key binary which is generated at build time.

Available options:

- No encoding (256 bit key) (CONFIG_SECURE_BOOTLOADER_KEY_ENCODING_256BIT)
- 3/4 encoding (192 bit key) (CONFIG_SECURE_BOOTLOADER_KEY_ENCODING_192BIT)

CONFIG_SECURE_BOOT_INSECURE

Allow potentially insecure options

Found in: Security features

You can disable some of the default protections offered by secure boot, in order to enable testing or a custom combination of security features.

Only enable these options if you are very sure.

Refer to the Secure Boot section of the ESP-IDF Programmer's Guide for this version before enabling.

Default value:

- No (disabled) if *CONFIG_SECURE_BOOT*

CONFIG_SECURE_FLASH_ENC_ENABLED

Enable flash encryption on boot (READ DOCS FIRST)

Found in: Security features

If this option is set, flash contents will be encrypted by the bootloader on first boot.

Note: After first boot, the system will be permanently encrypted. Re-flashing an encrypted system is complicated and not always possible.

Read *flash 加密* before enabling.

Default value:

- No (disabled)

CONFIG_SECURE_FLASH_ENCRYPTION_KEYSIZE

Size of generated XTS-AES key

Found in: Security features > CONFIG_SECURE_FLASH_ENC_ENABLED

Size of generated XTS-AES key.

- AES-128 uses a 256-bit key (32 bytes) derived from 128 bits (16 bytes) burned in half Efuse key block. Internally, it calculates SHA256(128 bits)
- AES-128 uses a 256-bit key (32 bytes) which occupies one Efuse key block.
- AES-256 uses a 512-bit key (64 bytes) which occupies two Efuse key blocks.

This setting is ignored if either type of key is already burned to Efuse before the first boot. In this case, the pre-burned key is used and no new key is generated.

Available options:

- AES-128 key derived from 128 bits (SHA256(128 bits)) (CONFIG_SECURE_FLASH_ENCRYPTION_AES128_DERIVED)
- AES-128 (256-bit key) (CONFIG_SECURE_FLASH_ENCRYPTION_AES128)
- AES-256 (512-bit key) (CONFIG_SECURE_FLASH_ENCRYPTION_AES256)

CONFIG_SECURE_FLASH_ENCRYPTION_MODE

Enable usage mode

Found in: Security features > CONFIG_SECURE_FLASH_ENC_ENABLED

By default Development mode is enabled which allows ROM download mode to perform flash encryption operations (plaintext is sent to the device, and it encrypts it internally and writes ciphertext to flash.) This mode is not secure, it's possible for an attacker to write their own chosen plaintext to flash.

Release mode should always be selected for production or manufacturing. Once enabled it's no longer possible for the device in ROM Download Mode to use the flash encryption hardware.

When EFUSE_VIRTUAL is enabled, SECURE_FLASH_ENCRYPTION_MODE_RELEASE is not available. For CI tests we use IDF_CI_BUILD to bypass it ("export IDF_CI_BUILD=1"). We do not recommend bypassing it for other purposes.

Refer to the Flash Encryption section of the ESP-IDF Programmer's Guide for details.

Available options:

- Development (NOT SECURE) (CONFIG_SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT)
- Release (CONFIG_SECURE_FLASH_ENCRYPTION_MODE_RELEASE)

Potentially insecure options Contains:

- *CONFIG_SECURE_BOOT_V2_ALLOW_EFUSE_RD_DIS*
- *CONFIG_SECURE_BOOT_ALLOW_SHORT_APP_PARTITION*
- *CONFIG_SECURE_BOOT_ALLOW_JTAG*
- *CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC*
- *CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE*
- *CONFIG_SECURE_BOOT_ALLOW_UNUSED_DIGEST_SLOTS*
- *CONFIG_SECURE_FLASH_REQUIRE_ALREADY_ENABLED*
- *CONFIG_SECURE_FLASH_SKIP_WRITE_PROTECTION_CACHE*

CONFIG_SECURE_BOOT_ALLOW_JTAG

Allow JTAG Debugging

Found in: Security features > Potentially insecure options

If not set (default), the bootloader will permanently disable JTAG (across entire chip) on first boot when either secure boot or flash encryption is enabled.

Setting this option leaves JTAG on for debugging, which negates all protections of flash encryption and some of the protections of secure boot.

Only set this option in testing environments.

Default value:

- No (disabled) if *CONFIG_SECURE_BOOT_INSECURE* || *CONFIG_SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT*

CONFIG_SECURE_BOOT_ALLOW_SHORT_APP_PARTITION

Allow app partition length not 64KB aligned

Found in: Security features > Potentially insecure options

If not set (default), app partition size must be a multiple of 64KB. App images are padded to 64KB length, and the bootloader checks any trailing bytes after the signature (before the next 64KB boundary) have not been written. This is because flash cache maps entire 64KB pages into the address space. This prevents an attacker from appending unverified data after the app image in the flash, causing it to be mapped into the address space.

Setting this option allows the app partition length to be unaligned, and disables padding of the app image to this length. It is generally not recommended to set this option, unless you have a legacy partitioning scheme which doesn't support 64KB aligned partition lengths.

CONFIG_SECURE_BOOT_V2_ALLOW_EFUSE_RD_DIS

Allow additional read protecting of efuses

Found in: Security features > Potentially insecure options

If not set (default, recommended), on first boot the bootloader will burn the WR_DIS_RD_DIS efuse when Secure Boot is enabled. This prevents any more efuses from being read protected.

If this option is set, it will remain possible to write the EFUSE_RD_DIS efuse field after Secure Boot is enabled. This may allow an attacker to read-protect the BLK2 efuse (for ESP32) and BLOCK4-BLOCK10 (i.e. BLOCK_KEY0-BLOCK_KEY5)(for other chips) holding the public key digest, causing an immediate denial of service and possibly allowing an additional fault injection attack to bypass the signature protection.

NOTE: Once a BLOCK is read-protected, the application will read all zeros from that block

NOTE: If "UART ROM download mode (Permanently disabled (recommended))" or "UART ROM download mode (Permanently switch to Secure mode (recommended))" is set, then it is __NOT__ possible to read/write efuses using espefuse.py utility. However, efuse can be read/written from the application

CONFIG_SECURE_BOOT_ALLOW_UNUSED_DIGEST_SLOTS

Leave unused digest slots available (not revoke)

Found in: Security features > Potentially insecure options

If not set (default), during startup in the app all unused digest slots will be revoked. To revoke unused slot will be called esp_efuse_set_digest_revoke(num_digest) for each digest. Revoking unused digest slots makes ensures that no trusted keys can be added later by an attacker. If set, it means that you have a plan to use unused digests slots later.

Note that if you plan to enable secure boot during the first boot up, the bootloader will intentionally revoke the unused digest slots while enabling secure boot, even if the above config is enabled because keeping the unused key slots un-revoked would a security hazard. In case for any development workflow if you need to avoid this revocation, you should enable secure boot externally (host based mechanism) rather than enabling it during the boot up, so that the bootloader would not need to enable secure boot and thus you could avoid its revocation strategy.

Default value:

- No (disabled) if *CONFIG_SECURE_BOOT_INSECURE*

CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC

Leave UART bootloader encryption enabled

Found in: Security features > Potentially insecure options

If not set (default), the bootloader will permanently disable UART bootloader encryption access on first boot. If set, the UART bootloader will still be able to access hardware encryption.

It is recommended to only set this option in testing environments.

Default value:

- No (disabled) if *CONFIG_SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT*

CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE

Leave UART bootloader flash cache enabled

Found in: Security features > Potentially insecure options

If not set (default), the bootloader will permanently disable UART bootloader flash cache access on first boot. If set, the UART bootloader will still be able to access the flash cache.

Only set this option in testing environments.

Default value:

- No (disabled) if *CONFIG_SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT* && (SOC_EFUSE_DIS_DOWNLOAD_ICACHE || SOC_EFUSE_DIS_DOWNLOAD_DCACHE)

CONFIG_SECURE_FLASH_REQUIRE_ALREADY_ENABLED

Require flash encryption to be already enabled

Found in: Security features > Potentially insecure options

If not set (default), and flash encryption is not yet enabled in eFuses, the 2nd stage bootloader will enable flash encryption: generate the flash encryption key and program eFuses. If this option is set, and flash encryption is not yet enabled, the bootloader will error out and reboot. If flash encryption is enabled in eFuses, this option does not change the bootloader behavior.

Only use this option in testing environments, to avoid accidentally enabling flash encryption on the wrong device. The device needs to have flash encryption already enabled using espefuse.py.

Default value:

- No (disabled) if *CONFIG_SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT*

CONFIG_SECURE_FLASH_SKIP_WRITE_PROTECTION_CACHE

Skip write-protection of DIS_CACHE (DIS_ICACHE, DIS_DCACHE)

Found in: Security features > Potentially insecure options

If not set (default, recommended), on the first boot the bootloader will burn the write-protection of DIS_CACHE(for ESP32) or DIS_ICACHE/DIS_DCACHE(for other chips) eFuse when Flash Encryption is enabled. Write protection for cache disable efuse prevents the chip from being blocked if it is set by accident. App and bootloader use cache so disabling it makes the chip useless for IDF. Due to other eFuses are linked with the same write protection bit (see the list below) then write-protection will not be done if these SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC, SECURE_BOOT_ALLOW_JTAG or SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE options are selected to give a chance to turn on the chip into the release mode later.

List of eFuses with the same write protection bit: ESP32: MAC, MAC_CRC, DISABLE_APP_CPU, DISABLE_BT, DIS_CACHE, VOL_LEVEL_HP_INV.

ESP32-C3: DIS_ICACHE, DIS_USB_JTAG, DIS_DOWNLOAD_ICACHE, DIS_USB_SERIAL_JTAG, DIS_FORCE_DOWNLOAD, DIS_TWAI, JTAG_SEL_ENABLE, DIS_PAD_JTAG, DIS_DOWNLOAD_MANUAL_ENCRYPT.

ESP32-C6: SWAP_UART_SDIO_EN, DIS_ICACHE, DIS_USB_JTAG, DIS_DOWNLOAD_ICACHE, DIS_USB_SERIAL_JTAG, DIS_FORCE_DOWNLOAD, DIS_TWAI, JTAG_SEL_ENABLE, DIS_PAD_JTAG, DIS_DOWNLOAD_MANUAL_ENCRYPT.

ESP32-H2: DIS_ICACHE, DIS_USB_JTAG, POWERGLITCH_EN, DIS_FORCE_DOWNLOAD, SPI_DOWNLOAD_MSPI_DIS, DIS_TWAI, JTAG_SEL_ENABLE, DIS_PAD_JTAG, DIS_DOWNLOAD_MANUAL_ENCRYPT.

ESP32-S2: DIS_ICACHE, DIS_DCACHE, DIS_DOWNLOAD_ICACHE, DIS_DOWNLOAD_DCACHE, DIS_FORCE_DOWNLOAD, DIS_USB, DIS_TWAI, DIS_BOOT_REMAP, SOFT_DIS_JTAG, HARD_DIS_JTAG, DIS_DOWNLOAD_MANUAL_ENCRYPT.

ESP32-S3: DIS_ICACHE, DIS_DCACHE, DIS_DOWNLOAD_ICACHE, DIS_DOWNLOAD_DCACHE, DIS_FORCE_DOWNLOAD, DIS_USB_OTG, DIS_TWAI, DIS_APP_CPU, DIS_PAD_JTAG, DIS_DOWNLOAD_MANUAL_ENCRYPT, DIS_USB_JTAG, DIS_USB_SERIAL_JTAG, STRAP_JTAG_SEL, USB_PHY_SEL.

CONFIG_SECURE_FLASH_ENCRYPT_ONLY_IMAGE_LEN_IN_APP_PART

Encrypt only the app image that is present in the partition of type app

Found in: Security features

If set (default), optimise encryption time for the partition of type APP, by only encrypting the app image that is present in the partition, instead of the whole partition. The image length used for encryption is derived from the image metadata, which includes the size of the app image, checksum, hash and also the signature sector when secure boot is enabled.

If not set, the whole partition of type APP would be encrypted, which increases the encryption time but might be useful if there is any custom data appended to the firmware image.

CONFIG_SECURE_FLASH_CHECK_ENC_EN_IN_APP

Check Flash Encryption enabled on app startup

Found in: Security features

If set (default), in an app during startup code, there is a check of the flash encryption eFuse bit is on (as the bootloader should already have set it). The app requires this bit is on to continue work otherwise abort.

If not set, the app does not care if the flash encryption eFuse bit is set or not.

Default value:

- Yes (enabled) if *CONFIG_SECURE_FLASH_ENC_ENABLED*

CONFIG_SECURE_UART_ROM_DL_MODE

UART ROM download mode

Found in: Security features

Available options:

- UART ROM download mode (Permanently disabled (recommended)) (CONFIG_SECURE_DISABLE_ROM_DL_MODE)
  If set, during startup the app will burn an eFuse bit to permanently disable the UART ROM Download Mode. This prevents any future use of esptool.py, espefuse.py and similar tools.
  Once disabled, if the SoC is booted with strapping pins set for ROM Download Mode then an error is printed instead.
  It is recommended to enable this option in any production application where Flash Encryption and/or Secure Boot is enabled and access to Download Mode is not required.
  It is also possible to permanently disable Download Mode by calling esp_efuse_disable_rom_download_mode() at runtime.
- UART ROM download mode (Permanently switch to Secure mode (recommended)) (CONFIG_SECURE_ENABLE_SECURE_ROM_DL_MODE)
  If set, during startup the app will burn an eFuse bit to permanently switch the UART ROM Download Mode into a separate Secure Download mode. This option can only work if Download Mode is not already disabled by eFuse.
  Secure Download mode limits the use of Download Mode functions to update SPI config, changing baud rate, basic flash write and a command to return a summary of currently enabled security features (*get_security_info*).
  Secure Download mode is not compatible with the esptool.py flasher stub feature, espefuse.py, read/writing memory or registers, encrypted download, or any other features that interact with unsupported Download Mode commands.
  Secure Download mode should be enabled in any application where Flash Encryption and/or Secure Boot is enabled. Disabling this option does not immediately cancel the

benefits of the security features, but it increases the potential ”attack surface” for an attacker to try and bypass them with a successful physical attack.
It is also possible to enable secure download mode at runtime by calling esp_efuse_enable_rom_secure_download_mode()
Note: Secure Download mode is not available for ESP32 (includes revisions till ECO3).

- UART ROM download mode (Enabled (not recommended)) (CONFIG_SECURE_INSECURE_ALLOW_DL_MODE)
This is a potentially insecure option. Enabling this option will allow the full UART download mode to stay enabled. This option SHOULD NOT BE ENABLED for production use cases.

Application manager

Contains:

- *CONFIG_APP_EXCLUDE_PROJECT_NAME_VAR*
- *CONFIG_APP_EXCLUDE_PROJECT_VER_VAR*
- *CONFIG_APP_PROJECT_VER_FROM_CONFIG*
- *CONFIG_APP_RETRIEVE_LEN_ELF_SHA*
- *CONFIG_APP_COMPILE_TIME_DATE*

CONFIG_APP_COMPILE_TIME_DATE

Use time/date stamp for app

Found in: Application manager

If set, then the app will be built with the current time/date stamp. It is stored in the app description structure. If not set, time/date stamp will be excluded from app image. This can be useful for getting the same binary image files made from the same source, but at different times.

CONFIG_APP_EXCLUDE_PROJECT_VER_VAR

Exclude PROJECT_VER from firmware image

Found in: Application manager

The PROJECT_VER variable from the build system will not affect the firmware image. This value will not be contained in the esp_app_desc structure.

Default value:

- No (disabled)

CONFIG_APP_EXCLUDE_PROJECT_NAME_VAR

Exclude PROJECT_NAME from firmware image

Found in: Application manager

The PROJECT_NAME variable from the build system will not affect the firmware image. This value will not be contained in the esp_app_desc structure.

Default value:

- No (disabled)

CONFIG_APP_PROJECT_VER_FROM_CONFIG

Get the project version from Kconfig

Found in: Application manager

If this is enabled, then config item APP_PROJECT_VER will be used for the variable PROJECT_VER. Other ways to set PROJECT_VER will be ignored.

Default value:

- No (disabled)

CONFIG_APP_PROJECT_VER

Project version

Found in: Application manager > CONFIG_APP_PROJECT_VER_FROM_CONFIG

Project version

Default value:

- 1 if *CONFIG_APP_PROJECT_VER_FROM_CONFIG*

CONFIG_APP_RETRIEVE_LEN_ELF_SHA

The length of APP ELF SHA is stored in RAM(chars)

Found in: Application manager

At startup, the app will read the embedded APP ELF SHA-256 hash value from flash and convert it into a string and store it in a RAM buffer. This ensures the panic handler and core dump will be able to print this string even when cache is disabled. The size of the buffer is APP_RETRIEVE_LEN_ELF_SHA plus the null terminator. Changing this value will change the size of this buffer, in bytes.

Range:

- from 8 to 64

Default value:

- 9

Boot ROM Behavior

Contains:

- *CONFIG_BOOT_ROM_LOG_SCHEME*

CONFIG_BOOT_ROM_LOG_SCHEME

Permanently change Boot ROM output

Found in: Boot ROM Behavior

Controls the Boot ROM log behavior. The rom log behavior can only be changed for once, specific eFuse bit(s) will be burned at app boot stage.

Available options:

- Always Log (CONFIG_BOOT_ROM_LOG_ALWAYS_ON)
  Always print ROM logs, this is the default behavior.
- Permanently disable logging (CONFIG_BOOT_ROM_LOG_ALWAYS_OFF)
  Don't print ROM logs.
- Log on GPIO High (CONFIG_BOOT_ROM_LOG_ON_GPIO_HIGH)
  Print ROM logs when GPIO level is high during start up. The GPIO number is chip dependent, e.g. on ESP32-S2, the control GPIO is GPIO46.
- Log on GPIO Low (CONFIG_BOOT_ROM_LOG_ON_GPIO_LOW)
  Print ROM logs when GPIO level is low during start up. The GPIO number is chip dependent, e.g. on ESP32-S2, the control GPIO is GPIO46.

Serial flasher config

Contains:

- *CONFIG_ESPTOOLPY_AFTER*
- *CONFIG_ESPTOOLPY_BEFORE*
- *CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE*
- *CONFIG_ESPTOOLPY_NO_STUB*
- *CONFIG_ESPTOOLPY_FLASH_SAMPLE_MODE*
- *CONFIG_ESPTOOLPY_FLASHSIZE*
- *CONFIG_ESPTOOLPY_FLASHMODE*
- *CONFIG_ESPTOOLPY_FLASHFREQ*

CONFIG_ESPTOOLPY_NO_STUB

Disable download stub

Found in: Serial flasher config

The flasher tool sends a precompiled download stub first by default. That stub allows things like compressed downloads and more. Usually you should not need to disable that feature

CONFIG_ESPTOOLPY_FLASHMODE

Flash SPI mode

Found in: Serial flasher config

Mode the flash chip is flashed in, as well as the default mode for the binary to run in.

Available options:

- QIO (CONFIG_ESPTOOLPY_FLASHMODE_QIO)
- QOUT (CONFIG_ESPTOOLPY_FLASHMODE_QOUT)
- DIO (CONFIG_ESPTOOLPY_FLASHMODE_DIO)
- DOUT (CONFIG_ESPTOOLPY_FLASHMODE_DOUT)
- OPI (CONFIG_ESPTOOLPY_FLASHMODE_OPI)

CONFIG_ESPTOOLPY_FLASH_SAMPLE_MODE

Flash Sampling Mode

Found in: Serial flasher config

Available options:

- STR Mode (CONFIG_ESPTOOLPY_FLASH_SAMPLE_MODE_STR)
- DTR Mode (CONFIG_ESPTOOLPY_FLASH_SAMPLE_MODE_DTR)

CONFIG_ESPTOOLPY_FLASHFREQ

Flash SPI speed

Found in: Serial flasher config

Available options:

- 120 MHz (READ DOCS FIRST) (CONFIG_ESPTOOLPY_FLASHFREQ_120M)

  - Optional feature for QSPI Flash. Read docs and enable *CONFIG_SPI_FLASH_HPM_ENA* first!
  - Flash 120 MHz SDR mode is stable.
  - Flash 120 MHz DDR mode is an experimental feature, it works when the temperature is stable.

    Risks: If your chip powers on at a certain temperature, then after the temperature increases or decreases by approximately 20 Celsius degrees (depending on the chip), the program will crash randomly.
- 80 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_80M)
- 64 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_64M)
- 60 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_60M)
- 48 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_48M)
- 40 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_40M)
- 32 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_32M)
- 30 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_30M)
- 26 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_26M)
- 24 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_24M)
- 20 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_20M)
- 16 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_16M)
- 15 MHz (CONFIG_ESPTOOLPY_FLASHFREQ_15M)

CONFIG_ESPTOOLPY_FLASHSIZE

Flash size

Found in: Serial flasher config

SPI flash size, in megabytes

Available options:

- 1 MB (CONFIG_ESPTOOLPY_FLASHSIZE_1MB)
- 2 MB (CONFIG_ESPTOOLPY_FLASHSIZE_2MB)
- 4 MB (CONFIG_ESPTOOLPY_FLASHSIZE_4MB)
- 8 MB (CONFIG_ESPTOOLPY_FLASHSIZE_8MB)
- 16 MB (CONFIG_ESPTOOLPY_FLASHSIZE_16MB)
- 32 MB (CONFIG_ESPTOOLPY_FLASHSIZE_32MB)
- 64 MB (CONFIG_ESPTOOLPY_FLASHSIZE_64MB)
- 128 MB (CONFIG_ESPTOOLPY_FLASHSIZE_128MB)

CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE

Detect flash size when flashing bootloader

Found in: Serial flasher config

If this option is set, flashing the project will automatically detect the flash size of the target chip and update the bootloader image before it is flashed.

Enabling this option turns off the image protection against corruption by a SHA256 digest. Updating the bootloader image before flashing would invalidate the digest.

CONFIG_ESPTOOLPY_BEFORE

Before flashing

Found in: Serial flasher config

Configure whether esptool.py should reset the ESP32 before flashing.

Automatic resetting depends on the RTS & DTR signals being wired from the serial port to the ESP32. Most USB development boards do this internally.

Available options:

- Reset to bootloader (CONFIG_ESPTOOLPY_BEFORE_RESET)
- No reset (CONFIG_ESPTOOLPY_BEFORE_NORESET)

CONFIG_ESPTOOLPY_AFTER

After flashing

Found in: Serial flasher config

Configure whether esptool.py should reset the ESP32 after flashing.

Automatic resetting depends on the RTS & DTR signals being wired from the serial port to the ESP32. Most USB development boards do this internally.

Available options:

- Reset after flashing (CONFIG_ESPTOOLPY_AFTER_RESET)
- Stay in bootloader (CONFIG_ESPTOOLPY_AFTER_NORESET)

Partition Table

Contains:

- *CONFIG_PARTITION_TABLE_CUSTOM_FILENAME*
- *CONFIG_PARTITION_TABLE_MD5*
- *CONFIG_PARTITION_TABLE_OFFSET*
- *CONFIG_PARTITION_TABLE_TYPE*

CONFIG_PARTITION_TABLE_TYPE

Partition Table

Found in: Partition Table

The partition table to flash to the ESP32. The partition table determines where apps, data and other resources are expected to be found.

The predefined partition table CSV descriptions can be found in the components/partition_table directory. These are mostly intended for example and development use, it's expect that for production use you will copy one of these CSV files and create a custom partition CSV for your application.

Available options:

- Single factory app, no OTA (CONFIG_PARTITION_TABLE_SINGLE_APP)
  This is the default partition table, designed to fit into a 2MB or larger flash with a single 1MB app partition.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_singleapp.csv
  This partition table is not suitable for an app that needs OTA (over the air update) capability.
- Single factory app (large), no OTA (CONFIG_PARTITION_TABLE_SINGLE_APP_LARGE)
  This is a variation of the default partition table, that expands the 1MB app partition size to 1.5MB to fit more code.

The corresponding CSV file in the IDF directory is components/partition_table/partitions_singleapp_large.csv
This partition table is not suitable for an app that needs OTA (over the air update) capability.

- Factory app, two OTA definitions (CONFIG_PARTITION_TABLE_TWO_OTA)
  This is a basic OTA-enabled partition table with a factory app partition plus two OTA app partitions. All are 1MB, so this partition table requires 4MB or larger flash size.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_two_ota.csv
- Custom partition table CSV (CONFIG_PARTITION_TABLE_CUSTOM)
  Specify the path to the partition table CSV to use for your project.
  Consult the Partition Table section in the ESP-IDF Programmers Guide for more information.
- Single factory app, no OTA, encrypted NVS (CONFIG_PARTITION_TABLE_SINGLE_APP_ENCRYPTED_NVS)
  This is a variation of the default "Single factory app, no OTA" partition table that supports encrypted NVS when using flash encryption. See the Flash Encryption section in the ESP-IDF Programmers Guide for more information.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_singleapp_encr_nvs.csv
- Single factory app (large), no OTA, encrypted NVS (CONFIG_PARTITION_TABLE_SINGLE_APP_LARGE_ENC_NVS)
  This is a variation of the "Single factory app (large), no OTA" partition table that supports encrypted NVS when using flash encryption. See the Flash Encryption section in the ESP-IDF Programmers Guide for more information.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_singleapp_large_encr_nvs.csv
- Factory app, two OTA definitions, encrypted NVS (CONFIG_PARTITION_TABLE_TWO_OTA_ENCRYPTED_NVS)
  This is a variation of the "Factory app, two OTA definitions" partition table that supports encrypted NVS when using flash encryption. See the Flash Encryption section in the ESP-IDF Programmers Guide for more information.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_two_ota_encr_nvs.csv

CONFIG_PARTITION_TABLE_CUSTOM_FILENAME

Custom partition CSV file

Found in: Partition Table

Name of the custom partition CSV filename. This path is evaluated relative to the project root directory by default. However, if the absolute path for the CSV file is provided, then the absolute path is configured.

Default value:

- "partitions.csv"

CONFIG_PARTITION_TABLE_OFFSET

Offset of partition table

Found in: Partition Table

The address of partition table (by default 0x8000). Allows you to move the partition table, it gives more space for the bootloader. Note that the bootloader and app will both need to be compiled with the same PARTITION_TABLE_OFFSET value.

This number should be a multiple of 0x1000.

Note that partition offsets in the partition table CSV file may need to be changed if this value is set to a higher value. To have each partition offset adapt to the configured partition table offset, leave all partition

offsets blank in the CSV file.

Default value:

- ”0x8000”

CONFIG_PARTITION_TABLE_MD5

Generate an MD5 checksum for the partition table

Found in: Partition Table

Generate an MD5 checksum for the partition table for protecting the integrity of the table. The generation should be turned off for legacy bootloaders which cannot recognize the MD5 checksum in the partition table.

Default value:

- Yes (enabled)

Compiler options

Contains:

- *CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL*
- *CONFIG_COMPILER_FLOAT_LIB_FROM*
- *CONFIG_COMPILER_RT_LIB*
- *CONFIG_COMPILER_NO_MERGE_CONSTANTS*
- *CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT*
- *CONFIG_COMPILER_DISABLE_GCC12_WARNINGS*
- *CONFIG_COMPILER_DISABLE_GCC13_WARNINGS*
- *CONFIG_COMPILER_DUMP_RTL_FILES*
- *CONFIG_COMPILER_SAVE_RESTORE_LIBCALLS*
- *CONFIG_COMPILER_WARN_WRITE_STRINGS*
- *CONFIG_COMPILER_CXX_EXCEPTIONS*
- *CONFIG_COMPILER_CXX_RTTI*
- *CONFIG_COMPILER_OPTIMIZATION*
- *CONFIG_COMPILER_ORPHAN_SECTIONS*
- *CONFIG_COMPILER_HIDE_PATHS_MACROS*
- *CONFIG_COMPILER_STACK_CHECK_MODE*

CONFIG_COMPILER_OPTIMIZATION

Optimization Level

Found in: Compiler options

This option sets compiler optimization level (gcc -O argument) for the app.

- The ”Debug” setting will add the -0g flag to CFLAGS.
- The ”Size” setting will add the -0s flag to CFLAGS.
- The ”Performance” setting will add the -O2 flag to CFLAGS.
- The ”None” setting will add the -O0 flag to CFLAGS.

The ”Size” setting cause the compiled code to be smaller and faster, but may lead to difficulties of correlating code addresses to source file lines when debugging.

The ”Performance” setting causes the compiled code to be larger and faster, but will be easier to correlated code addresses to source file lines.

”None” with -O0 produces compiled code without optimization.

Note that custom optimization levels may be unsupported.

Compiler optimization for the IDF bootloader is set separately, see the BOOTLOADER_COMPILER_OPTIMIZATION setting.

Available options:

- Debug (-Og) (CONFIG_COMPILER_OPTIMIZATION_DEBUG)
- Optimize for size (-Os) (CONFIG_COMPILER_OPTIMIZATION_SIZE)
- Optimize for performance (-O2) (CONFIG_COMPILER_OPTIMIZATION_PERF)
- Debug without optimization (-O0) (CONFIG_COMPILER_OPTIMIZATION_NONE)

CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL

Assertion level

Found in: Compiler options

Assertions can be:

- Enabled. Failure will print verbose assertion details. This is the default.
- Set to "silent" to save code size (failed assertions will abort() but user needs to use the aborting address to find the line number with the failed assertion.)
- Disabled entirely (not recommended for most configurations.) -DNDEBUG is added to CPPFLAGS in this case.

Available options:

- Enabled (CONFIG_COMPILER_OPTIMIZATION_ASSERTIONS_ENABLE)
  Enable assertions. Assertion content and line number will be printed on failure.
- Silent (saves code size) (CONFIG_COMPILER_OPTIMIZATION_ASSERTIONS_SILENT)
  Enable silent assertions. Failed assertions will abort(), user needs to use the aborting address to find the line number with the failed assertion.
- Disabled (sets -DNDEBUG) (CONFIG_COMPILER_OPTIMIZATION_ASSERTIONS_DISABLE)
  If assertions are disabled, -DNDEBUG is added to CPPFLAGS.

CONFIG_COMPILER_FLOAT_LIB_FROM

Compiler float lib source

Found in: Compiler options

In the soft-fp part of libgcc, riscv version is written in C, and handles all edge cases in IEEE754, which makes it larger and performance is slow.

RVfplib is an optimized RISC-V library for FP arithmetic on 32-bit integer processors, for single and double-precision FP. RVfplib is "fast", but it has a few exceptions from IEEE 754 compliance.

Available options:

- libgcc (CONFIG_COMPILER_FLOAT_LIB_FROM_GCCLIB)
- librvfp (CONFIG_COMPILER_FLOAT_LIB_FROM_RVFPLIB)

CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT

Disable messages in ESP_RETURN_ON_* and ESP_EXIT_ON_* macros

Found in: Compiler options

If enabled, the error messages will be discarded in following check macros: - ESP_RETURN_ON_ERROR - ESP_EXIT_ON_ERROR - ESP_RETURN_ON_FALSE - ESP_EXIT_ON_FALSE

Default value:

- No (disabled)

CONFIG_COMPILER_HIDE_PATHS_MACROS

Replace ESP-IDF and project paths in binaries

Found in: Compiler options

When expanding the __FILE__ and __BASE_FILE__ macros, replace paths inside ESP-IDF with paths relative to the placeholder string "IDF", and convert paths inside the project directory to relative paths.

This allows building the project with assertions or other code that embeds file paths, without the binary containing the exact path to the IDF or project directories.

This option passes -fmacro-prefix-map options to the GCC command line. To replace additional paths in your binaries, modify the project CMakeLists.txt file to pass custom -fmacro-prefix-map or -ffile-prefix-map arguments.

Default value:

- Yes (enabled)

CONFIG_COMPILER_CXX_EXCEPTIONS

Enable C++ exceptions

Found in: Compiler options

Enabling this option compiles all IDF C++ files with exception support enabled.

Disabling this option disables C++ exception support in all compiled files, and any libstdc++ code which throws an exception will abort instead.

Enabling this option currently adds an additional ~500 bytes of heap overhead when an exception is thrown in user code for the first time.

Default value:

- No (disabled)

Contains:

- *CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE*

CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE

Emergency Pool Size

Found in: Compiler options > CONFIG_COMPILER_CXX_EXCEPTIONS

Size (in bytes) of the emergency memory pool for C++ exceptions. This pool will be used to allocate memory for thrown exceptions when there is not enough memory on the heap.

Default value:

- 0 if *CONFIG_COMPILER_CXX_EXCEPTIONS*

CONFIG_COMPILER_CXX_RTTI

Enable C++ run-time type info (RTTI)

Found in: Compiler options

Enabling this option compiles all C++ files with RTTI support enabled. This increases binary size (typically by tens of kB) but allows using dynamic_cast conversion and typeid operator.

Default value:

- No (disabled)

CONFIG_COMPILER_STACK_CHECK_MODE

Stack smashing protection mode

Found in: Compiler options

Stack smashing protection mode. Emit extra code to check for buffer overflows, such as stack smashing attacks. This is done by adding a guard variable to functions with vulnerable objects. The guards are initialized when a function is entered and then checked when the function exits. If a guard check fails, program is halted. Protection has the following modes:

- In NORMAL mode (GCC flag: -fstack-protector) only functions that call alloca, and functions with buffers larger than 8 bytes are protected.
- STRONG mode (GCC flag: -fstack-protector-strong) is like NORMAL, but includes additional functions to be protected -- those that have local array definitions, or have references to local frame addresses.
- In OVERALL mode (GCC flag: -fstack-protector-all) all functions are protected.

Modes have the following impact on code performance and coverage:

- performance: NORMAL > STRONG > OVERALL
- coverage: NORMAL < STRONG < OVERALL

The performance impact includes increasing the amount of stack memory required for each task.

Available options:

- None (CONFIG_COMPILER_STACK_CHECK_MODE_NONE)
- Normal (CONFIG_COMPILER_STACK_CHECK_MODE_NORM)
- Strong (CONFIG_COMPILER_STACK_CHECK_MODE_STRONG)
- Overall (CONFIG_COMPILER_STACK_CHECK_MODE_ALL)

CONFIG_COMPILER_NO_MERGE_CONSTANTS

Disable merging const sections

Found in: Compiler options

Disable merging identical constants (string/floating-point) across compilation units. This helps in better size analysis of the application binary as the rodata section distribution is more uniform across libraries. On downside, it may increase the binary size and hence should be used during development phase only.

CONFIG_COMPILER_WARN_WRITE_STRINGS

Enable -Wwrite-strings warning flag

Found in: Compiler options

Adds -Wwrite-strings flag for the C/C++ compilers.

For C, this gives string constants the type `const char[]` so that copying the address of one into a non-const `char \*` pointer produces a warning. This warning helps to find at compile time code that tries to write into a string constant.

For C++, this warns about the deprecated conversion from string literals to `char \*`.

Default value:

- No (disabled)

CONFIG_COMPILER_SAVE_RESTORE_LIBCALLS

Enable -msave-restore flag to reduce code size

Found in: Compiler options

Adds -msave-restore to C/C++ compilation flags.

When this flag is enabled, compiler will call library functions to save/restore registers in function prologues/epilogues. This results in lower overall code size, at the expense of slightly reduced performance.

This option can be enabled for RISC-V targets only.

CONFIG_COMPILER_DISABLE_GCC12_WARNINGS

Disable new warnings introduced in GCC 12

Found in: Compiler options

Enable this option if use GCC 12 or newer, and want to disable warnings which don't appear with GCC 11.

Default value:

- No (disabled)

CONFIG_COMPILER_DISABLE_GCC13_WARNINGS

Disable new warnings introduced in GCC 13

Found in: Compiler options

Enable this option if use GCC 13 or newer, and want to disable warnings which don't appear with GCC 12.

Default value:

- No (disabled)

CONFIG_COMPILER_DUMP_RTL_FILES

Dump RTL files during compilation

Found in: Compiler options

If enabled, RTL files will be produced during compilation. These files can be used by other tools, for example to calculate call graphs.

CONFIG_COMPILER_RT_LIB

Compiler runtime library

Found in: Compiler options

Select runtime library to be used by compiler. - GCC toolchain supports libgcc only. - Clang allows to choose between libgcc or libclang_rt. - For host builds ("linux" target), uses the default library.

Available options:

- libgcc (CONFIG_COMPILER_RT_LIB_GCCLIB)
- libclang_rt (CONFIG_COMPILER_RT_LIB_CLANGRT)
- Host (CONFIG_COMPILER_RT_LIB_HOST)

CONFIG_COMPILER_ORPHAN_SECTIONS

Orphan sections handling

Found in: Compiler options

If the linker finds orphan sections, it attempts to place orphan sections after sections of the same attribute such as code vs data, loadable vs non-loadable, etc. That means that orphan sections could placed between sections defined in IDF linker scripts. This could lead to corruption of the binary image. Configure the linker action here.

Available options:

- Place with warning (CONFIG_COMPILER_ORPHAN_SECTIONS_WARNING)
  Places orphan sections with a warning message.
- Place silently (CONFIG_COMPILER_ORPHAN_SECTIONS_PLACE)
  Places orphan sections without a warning/error message.

Component config

Contains:

- *ADC and ADC Calibration*
- *Application Level Tracing*
- *Bluetooth*
- *Common ESP-related*
- *Console Library*
- *Core dump*
- *Driver Configurations*
- *eFuse Bit Manager*
- *CONFIG_BLE_MESH*
- *ESP HTTP client*
- *ESP HTTPS OTA*
- *ESP HTTPS server*
- *ESP NETIF Adapter*
- *ESP PSRAM*
- *ESP Ringbuf*
- *ESP System Settings*
- *ESP Timer (High Resolution Timer)*
- *ESP-Driver:Analog Comparator Configurations*
- *ESP-Driver:Camera Controller Configurations*
- *ESP-Driver:DAC Configurations*
- *ESP-Driver:GPIO Configurations*
- *ESP-Driver:GPTimer Configurations*
- *ESP-Driver:I2C Configurations*
- *ESP-Driver:I2S Configurations*
- *ESP-Driver:ISP Configurations*
- *ESP-Driver:JPEG-Codec Configurations*
- *ESP-Driver:LEDC Configurations*
- *ESP-Driver:MCPWM Configurations*
- *ESP-Driver:Parallel IO Configurations*
- *ESP-Driver:PCNT Configurations*
- *ESP-Driver:RMT Configurations*
- *ESP-Driver:Sigma Delta Modulator Configurations*
- *ESP-Driver:SPI Configurations*
- *ESP-Driver:Temperature Sensor Configurations*
- *ESP-Driver:Touch Sensor Configurations*
- *ESP-Driver:UART Configurations*
- *ESP-Driver:USB Serial/JTAG Configuration*
- *ESP-MQTT Configurations*
- *ESP-TLS*
- *Ethernet*
- *Event Loop Library*

- *FAT Filesystem support*
- *FreeRTOS*
- *GDB Stub*
- *Hardware Abstraction Layer (HAL) and Low Level (LL)*
- *Hardware Settings*
- *Heap memory debugging*
- *HTTP Server*
- *IEEE 802.15.4*
- *IPC (Inter-Processor Call)*
- *LCD and Touch Panel*
- *Log output*
- *LWIP*
- *Main Flash configuration*
- *mbedTLS*
- *Newlib*
- *NVS*
- *NVS Security Provider*
- *OpenThread*
- *Partition API Configuration*
- *Power Management*
- *Protocomm*
- *PThreads*
- *SoC Settings*
- *SPI Flash driver*
- *SPIFFS Configuration*
- *TCP Transport*
- *Ultra Low Power (ULP) Co-processor*
- *Unity unit testing library*
- *USB-OTG*
- *Virtual file system*
- *Wear Levelling*
- *Wi-Fi*
- *Wi-Fi Provisioning Manager*
- *Wireless Coexistence*

Application Level Tracing Contains:

- *CONFIG_APPTRACE_DESTINATION1*
- *CONFIG_APPTRACE_DESTINATION2*
- *FreeRTOS SystemView Tracing*
- *CONFIG_APPTRACE_GCOV_ENABLE*
- *CONFIG_APPTRACE_BUF_SIZE*
- *CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX*
- *CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH*
- *CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO*
- *CONFIG_APPTRACE_UART_BAUDRATE*
- *CONFIG_APPTRACE_UART_RX_GPIO*
- *CONFIG_APPTRACE_UART_RX_BUFF_SIZE*
- *CONFIG_APPTRACE_UART_TASK_PRIO*
- *CONFIG_APPTRACE_UART_TX_MSG_SIZE*
- *CONFIG_APPTRACE_UART_TX_GPIO*
- *CONFIG_APPTRACE_UART_TX_BUFF_SIZE*

CONFIG_APPTRACE_DESTINATION1

Data Destination 1

Found in: Component config > Application Level Tracing

Select destination for application trace: JTAG or none (to disable).

Available options:

- JTAG (CONFIG_APPTRACE_DEST_JTAG)
- None (CONFIG_APPTRACE_DEST_NONE)

CONFIG_APPTRACE_DESTINATION2

Data Destination 2

Found in: Component config > Application Level Tracing

Select destination for application trace: UART(XX) or none (to disable).

Available options:

- UART0 (CONFIG_APPTRACE_DEST_UART0)
- UART1 (CONFIG_APPTRACE_DEST_UART1)
- UART2 (CONFIG_APPTRACE_DEST_UART2)
- USB_CDC (CONFIG_APPTRACE_DEST_USB_CDC)
- None (CONFIG_APPTRACE_DEST_UART_NONE)

CONFIG_APPTRACE_UART_TX_GPIO

UART TX on GPIO#

Found in: Component config > Application Level Tracing

This GPIO is used for UART TX pin.

CONFIG_APPTRACE_UART_RX_GPIO

UART RX on GPIO#

Found in: Component config > Application Level Tracing

This GPIO is used for UART RX pin.

CONFIG_APPTRACE_UART_BAUDRATE

UART baud rate

Found in: Component config > Application Level Tracing

This baud rate is used for UART.

The app's maximum baud rate depends on the UART clock source. If Power Management is disabled, the UART clock source is the APB clock and all baud rates in the available range will be sufficiently accurate. If Power Management is enabled, REF_TICK clock source is used so the baud rate is divided from 1MHz. Baud rates above 1Mbps are not possible and values between 500Kbps and 1Mbps may not be accurate.

CONFIG_APPTRACE_UART_RX_BUFF_SIZE

UART RX ring buffer size

Found in: Component config > Application Level Tracing

Size of the UART input ring buffer. This size related to the baudrate, system tick frequency and amount of data to transfer. The data placed to this buffer before sent out to the interface.

CONFIG_APPTRACE_UART_TX_BUFF_SIZE

UART TX ring buffer size

Found in: Component config > Application Level Tracing

Size of the UART output ring buffer. This size related to the baudrate, system tick frequency and amount of data to transfer.

CONFIG_APPTRACE_UART_TX_MSG_SIZE

UART TX message size

Found in: Component config > Application Level Tracing

Maximum size of the single message to transfer.

CONFIG_APPTRACE_UART_TASK_PRIO

UART Task Priority

Found in: Component config > Application Level Tracing

UART task priority. In case of high events rate, this parameter could be changed up to (config-MAX_PRIORITIES-1).

Range:
- from 1 to 32

Default value:
- 1

CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO

Timeout for flushing last trace data to host on panic

Found in: Component config > Application Level Tracing

Timeout for flushing last trace data to host in case of panic. In ms. Use -1 to disable timeout and wait forever.

CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH

Threshold for flushing last trace data to host on panic

Found in: Component config > Application Level Tracing

Threshold for flushing last trace data to host on panic in post-mortem mode. This is minimal amount of data needed to perform flush. In bytes.

CONFIG_APPTRACE_BUF_SIZE

Size of the apptrace buffer

Found in: Component config > Application Level Tracing

Size of the memory buffer for trace data in bytes.

CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX

Size of the pending data buffer

Found in: Component config > Application Level Tracing

Size of the buffer for events in bytes. It is useful for buffering events from the time critical code (scheduler, ISRs etc). If this parameter is 0 then events will be discarded when main HW buffer is full.

FreeRTOS SystemView Tracing Contains:

- *CONFIG_APPTRACE_SV_CPU*
- *CONFIG_APPTRACE_SV_EVT_ISR_ENTER_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_ISR_EXIT_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_ISR_TO_SCHED_ENABLE*
- *CONFIG_APPTRACE_SV_MAX_TASKS*
- *CONFIG_APPTRACE_SV_EVT_IDLE_ENABLE*
- *CONFIG_APPTRACE_SV_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_TASK_CREATE_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_TASK_START_EXEC_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_TASK_START_READY_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_TASK_STOP_EXEC_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_TASK_STOP_READY_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_TASK_TERMINATE_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_TIMER_ENTER_ENABLE*
- *CONFIG_APPTRACE_SV_EVT_TIMER_EXIT_ENABLE*
- *CONFIG_APPTRACE_SV_TS_SOURCE*
- *CONFIG_APPTRACE_SV_EVT_OVERFLOW_ENABLE*
- *CONFIG_APPTRACE_SV_BUF_WAIT_TMO*

CONFIG_APPTRACE_SV_ENABLE

SystemView Tracing Enable

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables supporrt for SEGGER SystemView tracing functionality.

CONFIG_APPTRACE_SV_DEST

SystemView destination

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_APPTRACE_SV_ENABLE

SystemView witt transfer data trough defined interface.

Available options:

- Data destination JTAG (CONFIG_APPTRACE_SV_DEST_JTAG)
  Send SEGGER SystemView events through JTAG interface.
- Data destination UART (CONFIG_APPTRACE_SV_DEST_UART)
  Send SEGGER SystemView events through UART interface.

CONFIG_APPTRACE_SV_CPU

CPU to trace

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Define the CPU to trace by SystemView.

Available options:

- CPU0 (CONFIG_APPTRACE_SV_DEST_CPU_0)
  Send SEGGER SystemView events for Pro CPU.
- CPU1 (CONFIG_APPTRACE_SV_DEST_CPU_1)
  Send SEGGER SystemView events for App CPU.

CONFIG_APPTRACE_SV_TS_SOURCE

Timer to use as timestamp source

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

SystemView needs to use a hardware timer as the source of timestamps when tracing. This option selects the timer for it.

Available options:

- CPU cycle counter (CCOUNT) (CONFIG_APPTRACE_SV_TS_SOURCE_CCOUNT)
- General Purpose Timer (Timer Group) (CONFIG_APPTRACE_SV_TS_SOURCE_GPTIMER)
- esp_timer high resolution timer (CONFIG_APPTRACE_SV_TS_SOURCE_ESP_TIMER)

CONFIG_APPTRACE_SV_MAX_TASKS

Maximum supported tasks

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Configures maximum supported tasks in sysview debug

CONFIG_APPTRACE_SV_BUF_WAIT_TMO

Trace buffer wait timeout

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Configures timeout (in us) to wait for free space in trace buffer. Set to -1 to wait forever and avoid lost events.

CONFIG_APPTRACE_SV_EVT_OVERFLOW_ENABLE

Trace Buffer Overflow Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Trace Buffer Overflow” event.

CONFIG_APPTRACE_SV_EVT_ISR_ENTER_ENABLE

ISR Enter Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”ISR Enter” event.

CONFIG_APPTRACE_SV_EVT_ISR_EXIT_ENABLE

ISR Exit Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”ISR Exit” event.

CONFIG_APPTRACE_SV_EVT_ISR_TO_SCHED_ENABLE

ISR Exit to Scheduler Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”ISR to Scheduler” event.

CONFIG_APPTRACE_SV_EVT_TASK_START_EXEC_ENABLE

Task Start Execution Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Task Start Execution” event.

CONFIG_APPTRACE_SV_EVT_TASK_STOP_EXEC_ENABLE

Task Stop Execution Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Task Stop Execution” event.

CONFIG_APPTRACE_SV_EVT_TASK_START_READY_ENABLE

Task Start Ready State Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Task Start Ready State” event.

CONFIG_APPTRACE_SV_EVT_TASK_STOP_READY_ENABLE

Task Stop Ready State Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Task Stop Ready State” event.

CONFIG_APPTRACE_SV_EVT_TASK_CREATE_ENABLE

Task Create Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Task Create” event.

CONFIG_APPTRACE_SV_EVT_TASK_TERMINATE_ENABLE

Task Terminate Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Task Terminate” event.

CONFIG_APPTRACE_SV_EVT_IDLE_ENABLE

System Idle Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”System Idle” event.

CONFIG_APPTRACE_SV_EVT_TIMER_ENTER_ENABLE

Timer Enter Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Timer Enter” event.

CONFIG_APPTRACE_SV_EVT_TIMER_EXIT_ENABLE

Timer Exit Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables ”Timer Exit” event.

CONFIG_APPTRACE_GCOV_ENABLE

GCOV to Host Enable

Found in: Component config > Application Level Tracing

Enables support for GCOV data transfer to host.

CONFIG_APPTRACE_GCOV_DUMP_TASK_STACK_SIZE

Gcov dump task stack size

Found in: Component config > Application Level Tracing > CONFIG_APPTRACE_GCOV_ENABLE

Configures stack size of Gcov dump task

Default value:

- 2048 if *CONFIG_APPTRACE_GCOV_ENABLE*

Bluetooth Contains:

- *Bluedroid Options*
- *CONFIG_BT_ENABLED*
- *Common Options*
- *Controller Options*
- *CONFIG_BT_HCI_LOG_DEBUG_EN*
- *NimBLE Options*
- *CONFIG_BT_RELEASE_IRAM*

CONFIG_BT_ENABLED

Bluetooth

Found in: Component config > Bluetooth

Select this option to enable Bluetooth and show the submenu with Bluetooth configuration choices.

CONFIG_BT_HOST

Host

Found in: Component config > Bluetooth > CONFIG_BT_ENABLED

This helps to choose Bluetooth host stack

Available options:

- Bluedroid - Dual-mode (CONFIG_BT_BLUEDROID_ENABLED)
  This option is recommended for classic Bluetooth or for dual-mode usecases
- NimBLE - BLE only (CONFIG_BT_NIMBLE_ENABLED)
  This option is recommended for BLE only usecases to save on memory
- Disabled (CONFIG_BT_CONTROLLER_ONLY)
  This option is recommended when you want to communicate directly with the controller (without any host) or when you are using any other host stack not supported by Espressif (not mentioned here).

CONFIG_BT_CONTROLLER

Controller

Found in: Component config > Bluetooth > CONFIG_BT_ENABLED

This helps to choose Bluetooth controller stack

Available options:

- Enabled (CONFIG_BT_CONTROLLER_ENABLED)
  This option is recommended for Bluetooth controller usecases
- Disabled (CONFIG_BT_CONTROLLER_DISABLED)
  This option is recommended for Bluetooth Host only usecases

Bluedroid Options Contains:

- *CONFIG_BT_ABORT_WHEN_ALLOCATION_FAILS*
- *CONFIG_BT_BLE_HOST_QUEUE_CONG_CHECK*
- *CONFIG_BT_BLUEDROID_MEM_DEBUG*
- *CONFIG_BT_BTU_TASK_STACK_SIZE*
- *CONFIG_BT_BTC_TASK_STACK_SIZE*
- *CONFIG_BT_BLE_ENABLED*
- *BT DEBUG LOG LEVEL*
- *CONFIG_BT_ACL_CONNECTIONS*
- *CONFIG_BT_SMP_MAX_BONDS*
- *CONFIG_BT_ALLOCATION_FROM_SPIRAM_FIRST*
- *CONFIG_BT_CLASSIC_ENABLED*
- *CONFIG_BT_HID_ENABLED*
- *CONFIG_BT_STACK_NO_LOG*
- *CONFIG_BT_BLE_42_FEATURES_SUPPORTED*
- *CONFIG_BT_BLE_50_FEATURES_SUPPORTED*
- *CONFIG_BT_BLE_HIGH_DUTY_ADV_INTERVAL*
- *CONFIG_BT_MULTI_CONNECTION_ENBALE*
- *CONFIG_BT_BLE_FEAT_PERIODIC_ADV_SYNC_TRANSFER*
- *CONFIG_BT_BLE_FEAT_CREATE_SYNC_ENH*
- *CONFIG_BT_BLUEDROID_ESP_COEX_VSC*
- *CONFIG_BT_BLE_FEAT_PERIODIC_ADV_ENH*
- *CONFIG_BT_MAX_DEVICE_NAME_LEN*
- *CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN*
- *CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE*
- *CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT*
- *CONFIG_BT_BLE_RPA_TIMEOUT*
- *CONFIG_BT_BLE_RPA_SUPPORTED*
- *CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY*
- *CONFIG_BT_HFP_WBS_ENABLE*

CONFIG_BT_BTC_TASK_STACK_SIZE

Bluetooth event (callback to application) task stack size

Found in: Component config > Bluetooth > Bluedroid Options

This select btc task stack size

Default value:

- 3072 if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE

The cpu core which Bluedroid run

Found in: Component config > Bluetooth > Bluedroid Options

Which the cpu core to run Bluedroid. Can choose core0 and core1. Can not specify no-affinity.

Available options:

- Core 0 (PRO CPU) (CONFIG_BT_BLUEDROID_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (CONFIG_BT_BLUEDROID_PINNED_TO_CORE_1)

CONFIG_BT_BTU_TASK_STACK_SIZE

Bluetooth Bluedroid Host Stack task stack size

Found in: Component config > Bluetooth > Bluedroid Options

This select btu task stack size

Default value:

- 4352 if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLUEDROID_MEM_DEBUG

Bluedroid memory debug

Found in: Component config > Bluetooth > Bluedroid Options

Bluedroid memory debug

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLUEDROID_ESP_COEX_VSC

Enable Espressif Vendor-specific HCI commands for coexist status configuration

Found in: Component config > Bluetooth > Bluedroid Options

Enable Espressif Vendor-specific HCI commands for coexist status configuration

Default value:

- Yes (enabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_CLASSIC_ENABLED

Classic Bluetooth

Found in: Component config > Bluetooth > Bluedroid Options

For now this option needs ”SMP_ENABLE” to be set to yes

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && ((*CONFIG_BT_CONTROLLER_ENABLED* && SOC_BT_CLASSIC_SUPPORTED) || *CONFIG_BT_CONTROLLER_DISABLED*) && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_ENC_KEY_SIZE_CTRL_ENABLED

configure encryption key size

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

This chooses the support status of configuring encryption key size

Available options:

- Supported by standard HCI command (CONFIG_BT_ENC_KEY_SIZE_CTRL_STD)
- Supported by Vendor-specific HCI command (CONFIG_BT_ENC_KEY_SIZE_CTRL_VSC)
- Not supported (CONFIG_BT_ENC_KEY_SIZE_CTRL_NONE)

CONFIG_BT_CLASSIC_BQB_ENABLED

Host Qualitifcation support for Classic Bluetooth

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

This enables functionalities of Host qualification for Classic Bluetooth.

Default value:

- No (disabled) if *CONFIG_BT_CLASSIC_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_A2DP_ENABLE

A2DP

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

Advanced Audio Distribution Profile

Default value:

- No (disabled) if *CONFIG_BT_CLASSIC_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_SPP_ENABLED

SPP

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

This enables the Serial Port Profile

Default value:

- No (disabled) if *CONFIG_BT_CLASSIC_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_L2CAP_ENABLED

BT L2CAP

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

This enables the Logical Link Control and Adaptation Layer Protocol. Only supported classic bluetooth.

Default value:

- No (disabled) if *CONFIG_BT_CLASSIC_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_HFP_ENABLE

Hands Free/Handset Profile

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

Hands Free Unit and Audio Gateway can be included simultaneously but they cannot run simultaneously due to internal limitations.

Default value:

- No (disabled) if *CONFIG_BT_CLASSIC_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

Contains:

- *CONFIG_BT_HFP_AG_ENABLE*
- *CONFIG_BT_HFP_AUDIO_DATA_PATH*
- *CONFIG_BT_HFP_CLIENT_ENABLE*

CONFIG_BT_HFP_CLIENT_ENABLE

Hands Free Unit

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED > CONFIG_BT_HFP_ENABLE

Default value:

- Yes (enabled) if *CONFIG_BT_HFP_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_HFP_AG_ENABLE

Audio Gateway

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED > CONFIG_BT_HFP_ENABLE

Default value:

- Yes (enabled) if *CONFIG_BT_HFP_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_HFP_AUDIO_DATA_PATH

audio(SCO) data path

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED > CONFIG_BT_HFP_ENABLE

SCO data path, i.e. HCI or PCM. This option is set using API ”esp_bredr_sco_datapath_set” in Bluetooth host. Default SCO data path can also be set in Bluetooth Controller.

Available options:

- PCM (CONFIG_BT_HFP_AUDIO_DATA_PATH_PCM)
- HCI (CONFIG_BT_HFP_AUDIO_DATA_PATH_HCI)

CONFIG_BT_HFP_WBS_ENABLE

Wide Band Speech

Found in: Component config > Bluetooth > Bluedroid Options

This enables Wide Band Speech. Should disable it when SCO data path is PCM. Otherwise there will be no data transmitted via GPIOs.

Default value:

- Yes (enabled) if *CONFIG_BT_HFP_AUDIO_DATA_PATH_HCI* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_HID_ENABLED

Classic BT HID

Found in: Component config > Bluetooth > Bluedroid Options

This enables the BT HID Host

Default value:

- No (disabled) if *CONFIG_BT_CLASSIC_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

Contains:

- *CONFIG_BT_HID_DEVICE_ENABLED*
- *CONFIG_BT_HID_HOST_ENABLED*

CONFIG_BT_HID_HOST_ENABLED

Classic BT HID Host

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_HID_ENABLED

This enables the BT HID Host

Default value:

- No (disabled) if *CONFIG_BT_HID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_HID_DEVICE_ENABLED

Classic BT HID Device

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_HID_ENABLED

This enables the BT HID Device

CONFIG_BT_BLE_ENABLED

Bluetooth Low Energy

Found in: Component config > Bluetooth > Bluedroid Options

This enables Bluetooth Low Energy

Default value:

- Yes (enabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTS_ENABLE

Include GATT server module(GATTS)

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED

This option can be disabled when the app work only on gatt client mode

Default value:

- Yes (enabled) if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTS_PPCP_CHAR_GAP

Enable Peripheral Preferred Connection Parameters characteristic in GAP service

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

This enables "Peripheral Preferred Connection Parameters" characteristic (UUID: 0x2A04) in GAP service that has connection parameters like min/max connection interval, slave latency and supervision timeout multiplier

Default value:

- No (disabled) if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_BLUFI_ENABLE

Include blufi function

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

This option can be close when the app does not require blufi function.

Default value:

- No (disabled) if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATT_MAX_SR_PROFILES

Max GATT Server Profiles

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

Maximum GATT Server Profiles Count

Range:

- from 1 to 32 if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:

- 8 if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATT_MAX_SR_ATTRIBUTES

Max GATT Service Attributes

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

Maximum GATT Service Attributes Count

Range:

- from 1 to 500 if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:

- 100 if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTS_SEND_SERVICE_CHANGE_MODE

GATTS Service Change Mode

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

Service change indication mode for GATT Server.

Available options:

- GATTS manually send service change indication (CONFIG_BT_GATTS_SEND_SERVICE_CHANGE_MANUAL)
  Manually send service change indication through API esp_ble_gatts_send_service_change_indication()
- GATTS automatically send service change indication (CONFIG_BT_GATTS_SEND_SERVICE_CHANGE_AUTO)
  Let Bluedroid handle the service change indication internally

CONFIG_BT_GATTS_ROBUST_CACHING_ENABLED

Enable Robust Caching on Server Side

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

This option enables the GATT robust caching feature on the server. if turned on, the Client Supported Features characteristic, Database Hash characteristic, and Server Supported Features characteristic will be included in the GAP SERVICE.

Default value:

- No (disabled) if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTS_DEVICE_NAME_WRITABLE

Allow to write device name by GATT clients

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

Enabling this option allows remote GATT clients to write device name

Default value:

- No (disabled) if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTS_APPEARANCE_WRITABLE

Allow to write appearance by GATT clients

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

Enabling this option allows remote GATT clients to write appearance

Default value:

- No (disabled) if *CONFIG_BT_GATTS_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTC_ENABLE

Include GATT client module(GATTC)

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED

This option can be close when the app work only on gatt server mode

Default value:

- Yes (enabled) if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTC_MAX_CACHE_CHAR

Max gattc cache characteristic for discover

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTC_ENABLE

Maximum GATTC cache characteristic count

Range:
- from 1 to 500 if *CONFIG_BT_GATTC_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:
- 40 if *CONFIG_BT_GATTC_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTC_NOTIF_REG_MAX

Max gattc notify(indication) register number

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTC_ENABLE

Maximum GATTC notify(indication) register number

Range:
- from 1 to 64 if *CONFIG_BT_GATTC_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:
- 5 if *CONFIG_BT_GATTC_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTC_CACHE_NVS_FLASH

Save gattc cache data to nvs flash

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTC_ENABLE

This select can save gattc cache data to nvs flash

Default value:
- No (disabled) if *CONFIG_BT_GATTC_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_GATTC_CONNECT_RETRY_COUNT

The number of attempts to reconnect if the connection establishment failed

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTC_ENABLE

The number of attempts to reconnect if the connection establishment failed

Range:
- from 0 to 255 if *CONFIG_BT_GATTC_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:
- 3 if *CONFIG_BT_GATTC_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_SMP_ENABLE

Include BLE security module(SMP)

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED

This option can be close when the app not used the ble security connect.

Default value:
- Yes (enabled) if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_SMP_SLAVE_CON_PARAMS_UPD_ENABLE

Slave enable connection parameters update during pairing

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_BLE_SMP_ENABLE

In order to reduce the pairing time, slave actively initiates connection parameters update during pairing.

Default value:

- No (disabled) if *CONFIG_BT_BLE_SMP_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_SMP_ID_RESET_ENABLE

Reset device identity when all bonding records are deleted

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_BLE_SMP_ENABLE

There are tracking risks associated with using a fixed or static IRK. If enabled this option, Bluedroid will assign a new randomly-generated IRK when all pairing and bonding records are deleted. This would decrease the ability of a previously paired peer to be used to determine whether a device with which it previously shared an IRK is within range.

Default value:

- No (disabled) if *CONFIG_BT_BLE_SMP_ENABLE* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_STACK_NO_LOG

Disable BT debug logs (minimize bin size)

Found in: Component config > Bluetooth > Bluedroid Options

This select can save the rodata code size

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

BT DEBUG LOG LEVEL

Contains:

- *CONFIG_BT_LOG_A2D_TRACE_LEVEL*
- *CONFIG_BT_LOG_APPL_TRACE_LEVEL*
- *CONFIG_BT_LOG_AVCT_TRACE_LEVEL*
- *CONFIG_BT_LOG_AVDT_TRACE_LEVEL*
- *CONFIG_BT_LOG_AVRC_TRACE_LEVEL*
- *CONFIG_BT_LOG_BLUFI_TRACE_LEVEL*
- *CONFIG_BT_LOG_BNEP_TRACE_LEVEL*
- *CONFIG_BT_LOG_BTC_TRACE_LEVEL*
- *CONFIG_BT_LOG_BTIF_TRACE_LEVEL*
- *CONFIG_BT_LOG_BTM_TRACE_LEVEL*
- *CONFIG_BT_LOG_GAP_TRACE_LEVEL*
- *CONFIG_BT_LOG_GATT_TRACE_LEVEL*
- *CONFIG_BT_LOG_HCI_TRACE_LEVEL*
- *CONFIG_BT_LOG_HID_TRACE_LEVEL*
- *CONFIG_BT_LOG_L2CAP_TRACE_LEVEL*
- *CONFIG_BT_LOG_MCA_TRACE_LEVEL*
- *CONFIG_BT_LOG_OSI_TRACE_LEVEL*
- *CONFIG_BT_LOG_PAN_TRACE_LEVEL*
- *CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL*

- *CONFIG_BT_LOG_SDP_TRACE_LEVEL*
- *CONFIG_BT_LOG_SMP_TRACE_LEVEL*

CONFIG_BT_LOG_HCI_TRACE_LEVEL

HCI layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for HCI layer

Available options:

- NONE (CONFIG_BT_LOG_HCI_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_HCI_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_HCI_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_HCI_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_HCI_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_HCI_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_HCI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTM_TRACE_LEVEL

BTM layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BTM layer

Available options:

- NONE (CONFIG_BT_LOG_BTM_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_BTM_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_BTM_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_BTM_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_BTM_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_BTM_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_BTM_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_L2CAP_TRACE_LEVEL

L2CAP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for L2CAP layer

Available options:

- NONE (CONFIG_BT_LOG_L2CAP_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_L2CAP_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_L2CAP_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_L2CAP_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_L2CAP_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_L2CAP_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_L2CAP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL

RFCOMM layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for RFCOMM layer

Available options:

- NONE (CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_SDP_TRACE_LEVEL

SDP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for SDP layer

Available options:

- NONE (CONFIG_BT_LOG_SDP_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_SDP_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_SDP_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_SDP_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_SDP_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_SDP_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_SDP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_GAP_TRACE_LEVEL

GAP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for GAP layer

Available options:

- NONE (CONFIG_BT_LOG_GAP_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_GAP_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_GAP_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_GAP_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_GAP_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_GAP_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_GAP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BNEP_TRACE_LEVEL

BNEP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BNEP layer

Available options:

- NONE (CONFIG_BT_LOG_BNEP_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_BNEP_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_BNEP_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_BNEP_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_BNEP_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_BNEP_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_BNEP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_PAN_TRACE_LEVEL

PAN layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for PAN layer

Available options:

- NONE (CONFIG_BT_LOG_PAN_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_PAN_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_PAN_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_PAN_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_PAN_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_PAN_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_PAN_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_A2D_TRACE_LEVEL

A2D layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for A2D layer

Available options:

- NONE (CONFIG_BT_LOG_A2D_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_A2D_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_A2D_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_A2D_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_A2D_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_A2D_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_A2D_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_AVDT_TRACE_LEVEL

AVDT layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for AVDT layer

Available options:

- NONE (CONFIG_BT_LOG_AVDT_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_AVDT_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_AVDT_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_AVDT_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_AVDT_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_AVDT_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_AVDT_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_AVCT_TRACE_LEVEL

AVCT layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for AVCT layer

Available options:

- NONE (CONFIG_BT_LOG_AVCT_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_AVCT_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_AVCT_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_AVCT_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_AVCT_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_AVCT_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_AVCT_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_AVRC_TRACE_LEVEL

AVRC layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for AVRC layer

Available options:

- NONE (CONFIG_BT_LOG_AVRC_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_AVRC_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_AVRC_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_AVRC_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_AVRC_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_AVRC_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_AVRC_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_MCA_TRACE_LEVEL

MCA layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for MCA layer

Available options:

- NONE (CONFIG_BT_LOG_MCA_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_MCA_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_MCA_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_MCA_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_MCA_TRACE_LEVEL_EVENT)

- DEBUG (CONFIG_BT_LOG_MCA_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_MCA_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_HID_TRACE_LEVEL

HID layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for HID layer

Available options:

- NONE (CONFIG_BT_LOG_HID_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_HID_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_HID_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_HID_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_HID_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_HID_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_HID_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_APPL_TRACE_LEVEL

APPL layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for APPL layer

Available options:

- NONE (CONFIG_BT_LOG_APPL_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_APPL_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_APPL_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_APPL_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_APPL_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_APPL_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_APPL_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_GATT_TRACE_LEVEL

GATT layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for GATT layer

Available options:

- NONE (CONFIG_BT_LOG_GATT_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_GATT_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_GATT_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_GATT_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_GATT_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_GATT_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_GATT_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_SMP_TRACE_LEVEL

SMP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for SMP layer

Available options:

- NONE (CONFIG_BT_LOG_SMP_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_SMP_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_SMP_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_SMP_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_SMP_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_SMP_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_SMP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTIF_TRACE_LEVEL

BTIF layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BTIF layer

Available options:

- NONE (CONFIG_BT_LOG_BTIF_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_BTIF_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_BTIF_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_BTIF_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_BTIF_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_BTIF_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_BTIF_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTC_TRACE_LEVEL

BTC layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BTC layer

Available options:

- NONE (CONFIG_BT_LOG_BTC_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_BTC_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_BTC_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_BTC_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_BTC_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_BTC_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_BTC_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_OSI_TRACE_LEVEL

OSI layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for OSI layer

Available options:

- NONE (CONFIG_BT_LOG_OSI_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_OSI_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_OSI_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_OSI_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_OSI_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_OSI_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_OSI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BLUFI_TRACE_LEVEL

BLUFI layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BLUFI layer

Available options:

- NONE (CONFIG_BT_LOG_BLUFI_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BT_LOG_BLUFI_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BT_LOG_BLUFI_TRACE_LEVEL_WARNING)
- API (CONFIG_BT_LOG_BLUFI_TRACE_LEVEL_API)
- EVENT (CONFIG_BT_LOG_BLUFI_TRACE_LEVEL_EVENT)
- DEBUG (CONFIG_BT_LOG_BLUFI_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BT_LOG_BLUFI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_ACL_CONNECTIONS

BT/BLE MAX ACL CONNECTIONS(1~9)

Found in: Component config > Bluetooth > Bluedroid Options

Maximum BT/BLE connection count. The ESP32-C3/S3 chip supports a maximum of 10 instances, including ADV, SCAN and connections. The ESP32-C3/S3 chip can connect up to 9 devices if ADV or SCAN uses only one. If ADV and SCAN are both used, The ESP32-C3/S3 chip is connected to a maximum of 8 devices. Because Bluetooth cannot reclaim used instances once ADV or SCAN is used.

Range:

- from 1 to 9 if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:

- 4 if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_MULTI_CONNECTION_ENBALE

Enable BLE multi-connections

Found in: Component config > Bluetooth > Bluedroid Options

Enable this option if there are multiple connections

Default value:

- Yes (enabled) if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_ALLOCATION_FROM_SPIRAM_FIRST

BT/BLE will first malloc the memory from the PSRAM

Found in: Component config > Bluetooth > Bluedroid Options

This select can save the internal RAM if there have the PSRAM

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY

Use dynamic memory allocation in BT/BLE stack

Found in: Component config > Bluetooth > Bluedroid Options

This select can make the allocation of memory will become more flexible

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_HOST_QUEUE_CONG_CHECK

BLE queue congestion check

Found in: Component config > Bluetooth > Bluedroid Options

When scanning and scan duplicate is not enabled, if there are a lot of adv packets around or application layer handling adv packets is slow, it will cause the controller memory to run out. if enabled, adv packets will be lost when host queue is congested.

Default value:

- No (disabled) if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_SMP_MAX_BONDS

BT/BLE maximum bond device count

Found in: Component config > Bluetooth > Bluedroid Options

The number of security records for peer devices.

CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN

Report adv data and scan response individually when BLE active scan

Found in: Component config > Bluetooth > Bluedroid Options

Originally, when doing BLE active scan, Bluedroid will not report adv to application layer until receive scan response. This option is used to disable the behavior. When enable this option, Bluedroid will report adv data or scan response to application layer immediately.

Memory reserved at start of DRAM for Bluetooth stack

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT

Timeout of BLE connection establishment

Found in: Component config > Bluetooth > Bluedroid Options

Bluetooth Connection establishment maximum time, if connection time exceeds this value, the connection establishment fails, ESP_GATTC_OPEN_EVT or ESP_GATTS_OPEN_EVT is triggered.

Range:

- from 1 to 60 if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:

- 30 if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_MAX_DEVICE_NAME_LEN

length of bluetooth device name

Found in: Component config > Bluetooth > Bluedroid Options

Bluetooth Device name length shall be no larger than 248 octets, If the broadcast data cannot contain the complete device name, then only the shortname will be displayed, the rest parts that can't fit in will be truncated.

Range:

- from 32 to 248 if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:

- 32 if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_RPA_SUPPORTED

Update RPA to Controller

Found in: Component config > Bluetooth > Bluedroid Options

This enables controller RPA list function. For ESP32, ESP32 only support network privacy mode. If this option is enabled, ESP32 will only accept advertising packets from peer devices that contain private address, HW will not receive the advertising packets contain identity address after IRK changed. If this option is disabled, address resolution will be performed in the host, so the functions that require controller to resolve address in the white list cannot be used. This option is disabled by default on ESP32, please enable or disable this option according to your own needs.

For other BLE chips, devices support network privacy mode and device privacy mode, users can switch the two modes according to their own needs. So this option is enabled by default.

Default value:

- Yes (enabled) if *CONFIG_BT_CONTROLLER_DISABLED* && *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_CONTROLLER_DISABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_RPA_TIMEOUT

Timeout of resolvable private address

Found in: Component config > Bluetooth > Bluedroid Options

This set RPA timeout of Controller and Host. Default is 900 s (15 minutes). Range is 1 s to 1 hour (3600 s).

Range:

- from 1 to 3600 if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

Default value:

- 900 if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_50_FEATURES_SUPPORTED

Enable BLE 5.0 features

Found in: Component config > Bluetooth > Bluedroid Options

Enabling this option activates BLE 5.0 features. This option is universally supported in chips that support BLE, except for ESP32.

Default value:

- Yes (enabled) if *CONFIG_BT_BLE_ENABLED* && ((*CONFIG_BT_CONTROLLER_ENABLED* && SOC_BLE_50_SUPPORTED) || *CONFIG_BT_CONTROLLER_DISABLED*) && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_42_FEATURES_SUPPORTED

Enable BLE 4.2 features

Found in: Component config > Bluetooth > Bluedroid Options

This enables BLE 4.2 features.

Default value:

- No (disabled) if *CONFIG_BT_BLE_ENABLED* && ((*CONFIG_BT_CONTROLLER_ENABLED* && SOC_BLE_SUPPORTED) || *CONFIG_BT_CONTROLLER_DISABLED*) && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_FEAT_PERIODIC_ADV_SYNC_TRANSFER

Enable BLE periodic advertising sync transfer feature

Found in: Component config > Bluetooth > Bluedroid Options

This enables BLE periodic advertising sync transfer feature

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLE_50_FEATURES_SUPPORTED* && ((*CONFIG_BT_CONTROLLER_ENABLED* && SOC_ESP_NIMBLE_CONTROLLER) || *CONFIG_BT_CONTROLLER_DISABLED*) && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_FEAT_PERIODIC_ADV_ENH

Enable periodic adv enhancements(adi support)

Found in: Component config > Bluetooth > Bluedroid Options

Enable the periodic advertising enhancements

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLE_50_FEATURES_SUPPORTED* && ((*CONFIG_BT_CONTROLLER_ENABLED* && SOC_ESP_NIMBLE_CONTROLLER) || *CONFIG_BT_CONTROLLER_DISABLED*) && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_FEAT_CREATE_SYNC_ENH

Enable create sync enhancements(reporting disable and duplicate filtering enable support)

Found in: Component config > Bluetooth > Bluedroid Options

Enable the create sync enhancements

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLE_50_FEATURES_SUPPORTED* && ((*CONFIG_BT_CONTROLLER_ENABLED* && SOC_ESP_NIMBLE_CONTROLLER) || *CONFIG_BT_CONTROLLER_DISABLED*) && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_BLE_HIGH_DUTY_ADV_INTERVAL

Enable BLE high duty advertising interval feature

Found in: Component config > Bluetooth > Bluedroid Options

This enable BLE high duty advertising interval feature

Default value:

- No (disabled) if *CONFIG_BT_BLE_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

CONFIG_BT_ABORT_WHEN_ALLOCATION_FAILS

Abort when memory allocation fails in BT/BLE stack

Found in: Component config > Bluetooth > Bluedroid Options

This enables abort when memory allocation fails

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* && *CONFIG_BT_BLUEDROID_ENABLED*

NimBLE Options

Contains:

- *CONFIG_BT_NIMBLE_HOST_ALLOW_CONNECT_WITH_SCAN*
- *CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME*
- *CONFIG_BT_NIMBLE_HS_STOP_TIMEOUT_MS*
- *CONFIG_BT_NIMBLE_HOST_QUEUE_CONG_CHECK*
- *BLE Services*
- *CONFIG_BT_NIMBLE_WHITELIST_SIZE*
- *CONFIG_BT_NIMBLE_BLE_GATT_BLOB_TRANSFER*
- *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT*
- *CONFIG_BT_NIMBLE_ROLE_BROADCASTER*
- *CONFIG_BT_NIMBLE_ROLE_CENTRAL*
- *CONFIG_BT_NIMBLE_HIGH_DUTY_ADV_ITVL*
- *CONFIG_BT_NIMBLE_MESH*
- *CONFIG_BT_NIMBLE_ROLE_OBSERVER*
- *CONFIG_BT_NIMBLE_ROLE_PERIPHERAL*
- *CONFIG_BT_NIMBLE_SECURITY_ENABLE*
- *CONFIG_BT_NIMBLE_BLUFI_ENABLE*
- *CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT*
- *CONFIG_BT_NIMBLE_DYNAMIC_SERVICE*
- *CONFIG_BT_NIMBLE_USE_ESP_TIMER*
- *CONFIG_BT_NIMBLE_DEBUG*
- *CONFIG_BT_NIMBLE_HS_FLOW_CTRL*
- *CONFIG_BT_NIMBLE_VS_SUPPORT*
- *CONFIG_BT_NIMBLE_OPTIMIZE_MULTI_CONN*
- *CONFIG_BT_NIMBLE_ENC_ADV_DATA*
- *CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE*
- *GAP Service*
- *Host-controller Transport*
- *CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MAX_LEN*
- *CONFIG_BT_NIMBLE_MAX_BONDS*
- *CONFIG_BT_NIMBLE_MAX_CCCDS*

- *CONFIG_BT_NIMBLE_MAX_CONNECTIONS*
- *CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM*
- *CONFIG_BT_NIMBLE_GATT_MAX_PROCS*
- *CONFIG_BT_NIMBLE_MEM_ALLOC_MODE*
- *Memory Settings*
- *CONFIG_BT_NIMBLE_LOG_LEVEL*
- *CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE*
- *CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS*
- *CONFIG_BT_NIMBLE_NVS_PERSIST*
- *CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU*
- *CONFIG_BT_NIMBLE_SMP_ID_RESET*
- *CONFIG_BT_NIMBLE_RPA_TIMEOUT*
- *CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE*
- *CONFIG_BT_NIMBLE_TEST_THROUGHPUT_TEST*

CONFIG_BT_NIMBLE_MEM_ALLOC_MODE

Memory allocation strategy

Found in: Component config > Bluetooth > NimBLE Options

Allocation strategy for NimBLE host stack, essentially provides ability to allocate all required dynamic allocations from,

- Internal DRAM memory only
- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF
- Internal IRAM memory wherever applicable else internal DRAM

Available options:

- Internal memory (CONFIG_BT_NIMBLE_MEM_ALLOC_MODE_INTERNAL)
- External SPIRAM (CONFIG_BT_NIMBLE_MEM_ALLOC_MODE_EXTERNAL)
- Default alloc mode (CONFIG_BT_NIMBLE_MEM_ALLOC_MODE_DEFAULT)
- Internal IRAM (CONFIG_BT_NIMBLE_MEM_ALLOC_MODE_IRAM_8BIT)
  Allows to use IRAM memory region as 8bit accessible region.
  Every unaligned (8bit or 16bit) access will result in an exception and incur penalty of certain clock cycles per unaligned read/write.

CONFIG_BT_NIMBLE_LOG_LEVEL

NimBLE Host log verbosity

Found in: Component config > Bluetooth > NimBLE Options

Select NimBLE log level. Please make a note that the selected NimBLE log verbosity can not exceed the level set in "Component config --> Log output --> Default log verbosity".

Available options:

- No logs (CONFIG_BT_NIMBLE_LOG_LEVEL_NONE)
- Error logs (CONFIG_BT_NIMBLE_LOG_LEVEL_ERROR)
- Warning logs (CONFIG_BT_NIMBLE_LOG_LEVEL_WARNING)
- Info logs (CONFIG_BT_NIMBLE_LOG_LEVEL_INFO)
- Debug logs (CONFIG_BT_NIMBLE_LOG_LEVEL_DEBUG)

CONFIG_BT_NIMBLE_MAX_CONNECTIONS

Maximum number of concurrent connections

Found in: Component config > Bluetooth > NimBLE Options

Defines maximum number of concurrent BLE connections. For ESP32, user is expected to configure BTDM_CTRL_BLE_MAX_CONN from controller menu along with this option. Similarly for ESP32-C3 or ESP32-S3, user is expected to configure BT_CTRL_BLE_MAX_ACT from controller menu. For ESP32C2, ESP32C6 and ESP32H2, each connection will take about 1k DRAM.

Range:

- from 1 to 9 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 3 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MAX_BONDS

Maximum number of bonds to save across reboots

Found in: Component config > Bluetooth > NimBLE Options

Defines maximum number of bonds to save for peer security and our security

Default value:

- 3 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MAX_CCCDS

Maximum number of CCC descriptors to save across reboots

Found in: Component config > Bluetooth > NimBLE Options

Defines maximum number of CCC descriptors to save

Default value:

- 8 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM

Maximum number of connection oriented channels

Found in: Component config > Bluetooth > NimBLE Options

Defines maximum number of BLE Connection Oriented Channels. When set to (0), BLE COC is not compiled in

Range:

- from 0 to 9 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 0 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE

The CPU core on which NimBLE host will run

Found in: Component config > Bluetooth > NimBLE Options

The CPU core on which NimBLE host will run. You can choose Core 0 or Core 1. Cannot specify no-affinity

Available options:

- Core 0 (PRO CPU) (CONFIG_BT_NIMBLE_PINNED_TO_CORE_0)

- Core 1 (APP CPU) (CONFIG_BT_NIMBLE_PINNED_TO_CORE_1)

CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE

NimBLE Host task stack size

Found in: Component config > Bluetooth > NimBLE Options

This configures stack size of NimBLE host task

Default value:

- 5120 if *CONFIG_BLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*
- 4096 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ROLE_CENTRAL

Enable BLE Central role

Found in: Component config > Bluetooth > NimBLE Options

Enables central role

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ROLE_PERIPHERAL

Enable BLE Peripheral role

Found in: Component config > Bluetooth > NimBLE Options

Enable peripheral role

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ROLE_BROADCASTER

Enable BLE Broadcaster role

Found in: Component config > Bluetooth > NimBLE Options

Enables broadcaster role

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ROLE_OBSERVER

Enable BLE Observer role

Found in: Component config > Bluetooth > NimBLE Options

Enables observer role

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_NVS_PERSIST

Persist the BLE Bonding keys in NVS

Found in: Component config > Bluetooth > NimBLE Options

Enable this flag to make bonding persistent across device reboots

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SMP_ID_RESET

Reset device identity when all bonding records are deleted

Found in: Component config > Bluetooth > NimBLE Options

There are tracking risks associated with using a fixed or static IRK. If enabled this option, Bluedroid will assign a new randomly-generated IRK when all pairing and bonding records are deleted. This would decrease the ability of a previously paired peer to be used to determine whether a device with which it previously shared an IRK is within range.

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SECURITY_ENABLE

Enable BLE SM feature

Found in: Component config > Bluetooth > NimBLE Options

Enable BLE sm feature

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Contains:

- *CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_ENCRYPTION*
- *CONFIG_BT_NIMBLE_SM_LVL*
- *CONFIG_BT_NIMBLE_SM_LEGACY*
- *CONFIG_BT_NIMBLE_SM_SC*

CONFIG_BT_NIMBLE_SM_LEGACY

Security manager legacy pairing

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_SECURITY_ENABLE

Enable security manager legacy pairing

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_SECURITY_ENABLE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SM_SC

Security manager secure connections (4.2)

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_SECURITY_ENABLE

Enable security manager secure connections

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_SECURITY_ENABLE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SM_SC_DEBUG_KEYS

Use predefined public-private key pair

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_SECURITY_ENABLE > CONFIG_BT_NIMBLE_SM_SC

If this option is enabled, SM uses predefined DH key pair as described in Core Specification, Vol. 3, Part H, 2.3.5.6.1. This allows to decrypt air traffic easily and thus should only be used for debugging.

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_SECURITY_ENABLE* && *CONFIG_BT_NIMBLE_SM_SC* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_ENCRYPTION

Enable LE encryption

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_SECURITY_ENABLE

Enable encryption connection

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_SECURITY_ENABLE* && *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SM_LVL

Security level

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_SECURITY_ENABLE

LE Security Mode 1 Levels: 1. No Security 2. Unauthenticated pairing with encryption 3. Authenticated pairing with encryption 4. Authenticated LE Secure Connections pairing with encryption using a 128-bit strength encryption key.

Default value:

- 0 if *CONFIG_BT_NIMBLE_SECURITY_ENABLE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_DEBUG

Enable extra runtime asserts and host debugging

Found in: Component config > Bluetooth > NimBLE Options

This enables extra runtime asserts and host debugging

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_DYNAMIC_SERVICE

Enable dynamic services

Found in: Component config > Bluetooth > NimBLE Options

This enables user to add/remove Gatt services at runtime

CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME

BLE GAP default device name

Found in: Component config > Bluetooth > NimBLE Options

The Device Name characteristic shall contain the name of the device as an UTF-8 string. This name can be changed by using API ble_svc_gap_device_name_set()

Default value:

- ”nimble” if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MAX_LEN

Maximum length of BLE device name in octets

Found in: Component config > Bluetooth > NimBLE Options

Device Name characteristic value shall be 0 to 248 octets in length

Default value:

- 31 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU

Preferred MTU size in octets

Found in: Component config > Bluetooth > NimBLE Options

This is the default value of ATT MTU indicated by the device during an ATT MTU exchange. This value can be changed using API ble_att_set_preferred_mtu()

Default value:

- 256 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE

External appearance of the device

Found in: Component config > Bluetooth > NimBLE Options

Standard BLE GAP Appearance value in HEX format e.g. 0x02C0

Default value:

- 0 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Memory Settings

Contains:

- *CONFIG_BT_NIMBLE_TRANSPORT_ACL_FROM_LL_COUNT*
- *CONFIG_BT_NIMBLE_TRANSPORT_EVT_DISCARD_COUNT*
- *CONFIG_BT_NIMBLE_MSYS_BUF_FROM_HEAP*
- *CONFIG_BT_NIMBLE_L2CAP_COC_SDU_BUFF_COUNT*
- *CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT*
- *CONFIG_BT_NIMBLE_MSYS_1_BLOCK_SIZE*
- *CONFIG_BT_NIMBLE_MSYS_2_BLOCK_COUNT*
- *CONFIG_BT_NIMBLE_MSYS_2_BLOCK_SIZE*
- *CONFIG_BT_NIMBLE_TRANSPORT_ACL_SIZE*
- *CONFIG_BT_NIMBLE_TRANSPORT_EVT_COUNT*
- *CONFIG_BT_NIMBLE_TRANSPORT_EVT_SIZE*

CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT

MSYS_1 Block Count

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

MSYS is a system level mbuf registry. For prepare write & prepare responses MBUFs are allocated out of msys_1 pool. For NIMBLE_MESH enabled cases, this block count is increased by 8 than user defined count.

Default value:

- 24 if SOC_ESP_NIMBLE_CONTROLLER && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MSYS_1_BLOCK_SIZE

MSYS_1 Block Size

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

Dynamic memory size of block 1

Default value:

- 128 if SOC_ESP_NIMBLE_CONTROLLER && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MSYS_2_BLOCK_COUNT

MSYS_2 Block Count

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

Dynamic memory count

Default value:

- 24 if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MSYS_2_BLOCK_SIZE

MSYS_2 Block Size

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

Dynamic memory size of block 2

Default value:

- 320 if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MSYS_BUF_FROM_HEAP

Get Msys Mbuf from heap

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

This option sets the source of the shared msys mbuf memory between the Host and the Controller. Allocate the memory from the heap if this option is sets, from the mempool otherwise.

Default value:

- Yes (enabled) if BT_LE_MSYS_INIT_IN_CONTROLLER && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_TRANSPORT_ACL_FROM_LL_COUNT

ACL Buffer count

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

The number of ACL data buffers allocated for host.

Default value:

- 24 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_TRANSPORT_ACL_SIZE

Transport ACL Buffer size

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

This is the maximum size of the data portion of HCI ACL data packets. It does not include the HCI data header (of 4 bytes)

Default value:

- 255 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_TRANSPORT_EVT_SIZE

Transport Event Buffer size

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

This is the size of each HCI event buffer in bytes. In case of extended advertising, packets can be fragmented. 257 bytes is the maximum size of a packet.

Default value:

- 257 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*
- 70 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_TRANSPORT_EVT_COUNT

Transport Event Buffer count

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

This is the high priority HCI events' buffer size. High-priority event buffers are for everything except advertising reports. If there are no free high-priority event buffers then host will try to allocate a low-priority buffer instead

Default value:

- 30 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_TRANSPORT_EVT_DISCARD_COUNT

Discardable Transport Event Buffer count

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

This is the low priority HCI events' buffer size. Low-priority event buffers are only used for advertising reports. If there are no free low-priority event buffers, then an incoming advertising report will get dropped

Default value:

- 8 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_L2CAP_COC_SDU_BUFF_COUNT

L2cap coc Service Data Unit Buffer count

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

This is the service data unit buffer count for l2cap coc.

Default value:

- 1 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_GATT_MAX_PROCS

Maximum number of GATT client procedures

Found in: Component config > Bluetooth > NimBLE Options

Maximum number of GATT client procedures that can be executed.

Default value:

- 4 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_HS_FLOW_CTRL

Enable Host Flow control

Found in: Component config > Bluetooth > NimBLE Options

Enable Host Flow control

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_ITVL

Host Flow control interval

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL

Host flow control interval in msecs

Default value:

- 1000 if *CONFIG_BT_NIMBLE_HS_FLOW_CTRL* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_THRESH

Host Flow control threshold

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL

Host flow control threshold, if the number of free buffers are at or below this threshold, send an immediate number-of-completed-packets event

Default value:

- 2 if *CONFIG_BT_NIMBLE_HS_FLOW_CTRL* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT

Host Flow control on disconnect

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL

Enable this option to send number-of-completed-packets event to controller after disconnection

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_HS_FLOW_CTRL* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_RPA_TIMEOUT

RPA timeout in seconds

Found in: Component config > Bluetooth > NimBLE Options

Time interval between RPA address change.

Range:

- from 1 to 41400 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 900 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH

Enable BLE mesh functionality

Found in: Component config > Bluetooth > NimBLE Options

Enable BLE Mesh example present in upstream mynewt-nimble and not maintained by Espressif.

IDF maintains ESP-BLE-MESH as the official Mesh solution. Please refer to ESP-BLE-MESH guide at: *:doc:../esp32/api-guides/esp-ble-mesh/ble-mesh-index“*

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Contains:

- *CONFIG_BT_NIMBLE_MESH_PROVISIONER*
- *CONFIG_BT_NIMBLE_MESH_PROV*
- *CONFIG_BT_NIMBLE_MESH_GATT_PROXY*
- *CONFIG_BT_NIMBLE_MESH_FRIEND*
- *CONFIG_BT_NIMBLE_MESH_LOW_POWER*
- *CONFIG_BT_NIMBLE_MESH_PROXY*
- *CONFIG_BT_NIMBLE_MESH_RELAY*
- *CONFIG_BT_NIMBLE_MESH_DEVICE_NAME*
- *CONFIG_BT_NIMBLE_MESH_NODE_COUNT*

CONFIG_BT_NIMBLE_MESH_PROXY

Enable mesh proxy functionality

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable proxy. This is automatically set whenever NIMBLE_MESH_PB_GATT or NIMBLE_MESH_GATT_PROXY is set

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_PROV

Enable BLE mesh provisioning

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable mesh provisioning

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_PB_ADV

Enable mesh provisioning over advertising bearer

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH > CONFIG_BT_NIMBLE_MESH_PROV

Enable this option to allow the device to be provisioned over the advertising bearer

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_MESH_PROV* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_PB_GATT

Enable mesh provisioning over GATT bearer

Found in: Component config > *Bluetooth* > *NimBLE Options* > *CONFIG_BT_NIMBLE_MESH* > *CONFIG_BT_NIMBLE_MESH_PROV*

Enable this option to allow the device to be provisioned over the GATT bearer

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_MESH_PROV* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_GATT_PROXY

Enable GATT Proxy functionality

Found in: Component config > *Bluetooth* > *NimBLE Options* > *CONFIG_BT_NIMBLE_MESH*

This option enables support for the Mesh GATT Proxy Service, i.e. the ability to act as a proxy between a Mesh GATT Client and a Mesh network

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_RELAY

Enable mesh relay functionality

Found in: Component config > *Bluetooth* > *NimBLE Options* > *CONFIG_BT_NIMBLE_MESH*

Support for acting as a Mesh Relay Node

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_LOW_POWER

Enable mesh low power mode

Found in: Component config > *Bluetooth* > *NimBLE Options* > *CONFIG_BT_NIMBLE_MESH*

Enable this option to be able to act as a Low Power Node

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_FRIEND

Enable mesh friend functionality

Found in: Component config > *Bluetooth* > *NimBLE Options* > *CONFIG_BT_NIMBLE_MESH*

Enable this option to be able to act as a Friend Node

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_DEVICE_NAME

Set mesh device name

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

This value defines Bluetooth Mesh device/node name

Default value:

- "nimble-mesh-node" if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_NODE_COUNT

Set mesh node count

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Defines mesh node count.

Default value:

- 1 if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MESH_PROVISIONER

Enable BLE mesh provisioner

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable mesh provisioner.

Default value:

- 0 if *CONFIG_BT_NIMBLE_MESH* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS

Override TinyCrypt with mbedTLS for crypto computations

Found in: Component config > Bluetooth > NimBLE Options

Enable this option to choose mbedTLS instead of TinyCrypt for crypto computations.

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_HS_STOP_TIMEOUT_MS

BLE host stop timeout in msec

Found in: Component config > Bluetooth > NimBLE Options

BLE Host stop procedure timeout in milliseconds.

Default value:

- 2000 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT

Enable connection reattempts on connection establishment error

Found in: Component config > Bluetooth > NimBLE Options

Enable to make the NimBLE host to reattempt GAP connection on connection establishment failure.

Default value:

- Yes (enabled) if SOC_ESP_NIMBLE_CONTROLLER && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MAX_CONN_REATTEMPT

Maximum number connection reattempts

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT

Defines maximum number of connection reattempts.

Range:

- from 1 to 255 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 3 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable BLE 5 feature

Found in: Component config > Bluetooth > NimBLE Options

Enable BLE 5 feature

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED* && SOC_BLE_50_SUPPORTED && *CONFIG_BT_NIMBLE_ENABLED*

Contains:

- *CONFIG_BT_NIMBLE_AOA_AOD*
- *CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_2M_PHY*
- *CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_CODED_PHY*
- *CONFIG_BT_NIMBLE_EXT_ADV*
- *CONFIG_BT_NIMBLE_EXT_SCAN*
- *CONFIG_BT_NIMBLE_GATT_CACHING*
- *CONFIG_BT_NIMBLE_BLE_POWER_CONTROL*

CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_2M_PHY

Enable 2M Phy

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable 2M-PHY

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_CODED_PHY

Enable coded Phy

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable coded-PHY

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_EXT_ADV

Enable extended advertising

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable this option to do extended advertising. Extended advertising will be supported from BLE 5.0 onwards.

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MAX_EXT_ADV_INSTANCES

Maximum number of extended advertising instances.

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV

Change this option to set maximum number of extended advertising instances. Minimum there is always one instance of advertising. Enter how many more advertising instances you want. For ESP32C2, ESP32C6 and ESP32H2, each extended advertising instance will take about 0.5k DRAM.

Range:

- from 0 to 4 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 1 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*
- 0 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_EXT_ADV_MAX_SIZE

Maximum length of the advertising data.

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV

Defines the length of the extended adv data. The value should not exceed 1650.

Range:

- from 0 to 1650 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 1650 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*
- 0 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV

Enable periodic advertisement.

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV

Enable this option to start periodic advertisement.

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_PERIODIC_ADV_ENH

Periodic adv enhancements(adi support)

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV > CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV

Enable the periodic advertising enhancements

CONFIG_BT_NIMBLE_PERIODIC_ADV_SYNC_TRANSFER

Enable Transfer Sync Events

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV > CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV

This enables controller transfer periodic sync events to host

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_EXT_SCAN

Enable extended scanning

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable this option to do extended scanning.

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ROLE_OBSERVER* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ENABLE_PERIODIC_SYNC

Enable periodic sync

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_SCAN

Enable this option to receive periodic advertisement.

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_EXT_SCAN* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MAX_PERIODIC_SYNCS

Maximum number of periodic advertising syncs

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_SCAN > CONFIG_BT_NIMBLE_ENABLE_PERIODIC_SYNC

Set this option to set the upper limit for number of periodic sync connections. This should be less than maximum connections allowed by controller.

Range:

- from 0 to 8 if *CONFIG_BT_NIMBLE_ENABLE_PERIODIC_SYNC* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 1 if *CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV* && *CONFIG_BT_NIMBLE_ENABLE_PERIODIC_SYNC* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*
- 0 if *CONFIG_BT_NIMBLE_ENABLE_PERIODIC_SYNC* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_MAX_PERIODIC_ADVERTISER_LIST

Maximum number of periodic advertiser list

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_SCAN > CONFIG_BT_NIMBLE_ENABLE_PERIODIC_SYNC

Set this option to set the upper limit for number of periodic advertiser list.

Range:

- from 1 to 5 if SOC_ESP_NIMBLE_CONTROLLER && *CONFIG_BT_NIMBLE_ENABLE_PERIODIC_SYNC* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 5 if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && SOC_ESP_NIMBLE_CONTROLLER && *CONFIG_BT_NIMBLE_ENABLE_PERIODIC_SYNC* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_BLE_POWER_CONTROL

Enable support for BLE Power Control

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Set this option to enable the Power Control feature

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && SOC_BLE_POWER_CONTROL_SUPPORTED && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_AOA_AOD

Direction Finding

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable support for Connectionless and Connection Oriented Direction Finding

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && (*CONFIG_BT_CONTROLLER_DISABLED* || SOC_BLE_CTE_SUPPORTED) && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_GATT_CACHING

Enable GATT caching

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable GATT caching

Contains:

- *CONFIG_BT_NIMBLE_GATT_CACHING_MAX_CONNS*
- *CONFIG_BT_NIMBLE_GATT_CACHING_MAX_CHRS*
- *CONFIG_BT_NIMBLE_GATT_CACHING_MAX_DSCS*
- *CONFIG_BT_NIMBLE_GATT_CACHING_MAX_SVCS*

CONFIG_BT_NIMBLE_GATT_CACHING_MAX_CONNS

Maximum connections to be cached

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_GATT_CACHING

Set this option to set the upper limit on number of connections to be cached.

Default value:

- 1 if *CONFIG_BT_NIMBLE_GATT_CACHING* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_GATT_CACHING_MAX_SVCS

Maximum number of services per connection

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_GATT_CACHING

Set this option to set the upper limit on number of services per connection to be cached.

Default value:

- 64 if *CONFIG_BT_NIMBLE_GATT_CACHING* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_GATT_CACHING_MAX_CHRS

Maximum number of characteristics per connection

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_GATT_CACHING

Set this option to set the upper limit on number of characteristics per connection to be cached.

Default value:

- 64 if *CONFIG_BT_NIMBLE_GATT_CACHING* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_GATT_CACHING_MAX_DSCS

Maximum number of descriptors per connection

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_GATT_CACHING

Set this option to set the upper limit on number of descriptors per connection to be cached.

Default value:

- 64 if *CONFIG_BT_NIMBLE_GATT_CACHING* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_WHITELIST_SIZE

BLE white list size

Found in: Component config > Bluetooth > NimBLE Options

BLE list size

Range:

- from 1 to 15 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Default value:

- 12 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_TEST_THROUGHPUT_TEST

Throughput Test Mode enable

Found in: Component config > Bluetooth > NimBLE Options

Enable the throughput test mode

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_BLUFI_ENABLE

Enable blufi functionality

Found in: Component config > Bluetooth > NimBLE Options

Set this option to enable blufi functionality.

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_USE_ESP_TIMER

Enable Esp Timer for Nimble

Found in: Component config > Bluetooth > NimBLE Options

Set this option to use Esp Timer which has higher priority timer instead of FreeRTOS timer

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_BLE_GATT_BLOB_TRANSFER

Blob transfer

Found in: Component config > Bluetooth > NimBLE Options

This option is used when data to be sent is more than 512 bytes. For peripheral role, BT_NIMBLE_MSYS_1_BLOCK_COUNT needs to be increased according to the need.

GAP Service Contains:

- *GAP Appearance write permissions*
- *CONFIG_BT_NIMBLE_SVC_GAP_CENT_ADDR_RESOLUTION*
- *GAP device name write permissions*
- *CONFIG_BT_NIMBLE_SVC_GAP_PPCP_MAX_CONN_INTERVAL*
- *CONFIG_BT_NIMBLE_SVC_GAP_PPCP_MIN_CONN_INTERVAL*
- *CONFIG_BT_NIMBLE_SVC_GAP_PPCP_SLAVE_LATENCY*
- *CONFIG_BT_NIMBLE_SVC_GAP_PPCP_SUPERVISION_TMO*

GAP Appearance write permissions Contains:

- *CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE*

CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE

Write

Found in: Component config > Bluetooth > NimBLE Options > GAP Service > GAP Appearance write permissions

Enable write permission (BLE_GATT_CHR_F_WRITE)

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE_ENC

Write with encryption

Found in: Component config > Bluetooth > NimBLE Options > GAP Service > GAP Appearance write permissions > CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE

Enable write with encryption permission (BLE_GATT_CHR_F_WRITE_ENC)

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE_AUTHEN

Write with authentication

Found in: Component config > Bluetooth > NimBLE Options > GAP Service > GAP Appearance write permissions > CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE

Enable write with authentication permission (BLE_GATT_CHR_F_WRITE_AUTHEN)

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE_AUTHOR

Write with authorisation

Found in: Component config > Bluetooth > NimBLE Options > GAP Service > GAP Appearance write permissions > CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE

Enable write with authorisation permission (BLE_GATT_CHR_F_WRITE_AUTHOR)

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_SVC_GAP_APPEAR_WRITE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_CENT_ADDR_RESOLUTION

GAP Characteristic - Central Address Resolution

Found in: Component config > Bluetooth > NimBLE Options > GAP Service

Weather or not Central Address Resolution characteristic is supported on the device, and if supported, weather or not Central Address Resolution is supported.

- Central Address Resolution characteristic not supported
- Central Address Resolution not supported
- Central Address Resolution supported

Available options:

- Characteristic not supported (CONFIG_BT_NIMBLE_SVC_GAP_CAR_CHAR_NOT_SUPP)
- Central Address Resolution not supported (CONFIG_BT_NIMBLE_SVC_GAP_CAR_NOT_SUPP)
- Central Address Resolution supported (CONFIG_BT_NIMBLE_SVC_GAP_CAR_SUPP)

GAP device name write permissions Contains:

- *CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE*

CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE

Write

Found in: Component config > Bluetooth > NimBLE Options > GAP Service > GAP device name write permissions

Enable write permission (BLE_GATT_CHR_F_WRITE)

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE_ENC

Write with encryption

Found in: Component config > Bluetooth > NimBLE Options > GAP Service > GAP device name write permissions > CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE

Enable write with encryption permission (BLE_GATT_CHR_F_WRITE_ENC)

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE_AUTHEN

Write with authentication

Found in: Component config > Bluetooth > NimBLE Options > GAP Service > GAP device name write permissions > CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE

Enable write with authentication permission (BLE_GATT_CHR_F_WRITE_AUTHEN)

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE_AUTHOR

Write with authorisation

Found in: Component config > Bluetooth > NimBLE Options > GAP Service > GAP device name write permissions > CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE

Enable write with authorisation permission (BLE_GATT_CHR_F_WRITE_AUTHOR)

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_SVC_GAP_NAME_WRITE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_PPCP_MAX_CONN_INTERVAL

PPCP Connection Interval Max (Unit: 1.25 ms)

Found in: Component config > Bluetooth > NimBLE Options > GAP Service

Peripheral Preferred Connection Parameter: Connection Interval maximum value Interval Max = value * 1.25 ms

Default value:

- 0 if *CONFIG_BT_NIMBLE_ROLE_PERIPHERAL* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_PPCP_MIN_CONN_INTERVAL

PPCP Connection Interval Min (Unit: 1.25 ms)

Found in: Component config > Bluetooth > NimBLE Options > GAP Service

Peripheral Preferred Connection Parameter: Connection Interval minimum value Interval Min = value * 1.25 ms

Default value:

- 0 if *CONFIG_BT_NIMBLE_ROLE_PERIPHERAL* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_PPCP_SLAVE_LATENCY

PPCP Slave Latency

Found in: Component config > Bluetooth > NimBLE Options > GAP Service

Peripheral Preferred Connection Parameter: Slave Latency

Default value:

- 0 if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_GAP_PPCP_SUPERVISION_TMO

PPCP Supervision Timeout (Uint: 10 ms)

Found in: Component config > Bluetooth > NimBLE Options > GAP Service

Peripheral Preferred Connection Parameter: Supervision Timeout Timeout = Value * 10 ms

Default value:

- 0 if *CONFIG_BT_NIMBLE_ENABLED*

BLE Services Contains:

- *CONFIG_BT_NIMBLE_HID_SERVICE*

CONFIG_BT_NIMBLE_HID_SERVICE

HID service

Found in: Component config > Bluetooth > NimBLE Options > BLE Services

Enable HID service support

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Contains:

- *CONFIG_BT_NIMBLE_SVC_HID_MAX_RPTS*
- *CONFIG_BT_NIMBLE_SVC_HID_MAX_INSTANCES*

CONFIG_BT_NIMBLE_SVC_HID_MAX_INSTANCES

Maximum HID service instances

Found in: Component config > Bluetooth > NimBLE Options > BLE Services > CONFIG_BT_NIMBLE_HID_SERVICE

Defines maximum number of HID service instances

Default value:

- 2 if *CONFIG_BT_NIMBLE_HID_SERVICE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_SVC_HID_MAX_RPTS

Maximum HID Report characteristics per service instance

Found in: Component config > Bluetooth > NimBLE Options > BLE Services > CONFIG_BT_NIMBLE_HID_SERVICE

Defines maximum number of report characteristics per service instance

Default value:

- 3 if *CONFIG_BT_NIMBLE_HID_SERVICE* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_VS_SUPPORT

Enable support for VSC and VSE

Found in: Component config > Bluetooth > NimBLE Options

This option is used to enable support for sending Vendor Specific HCI commands and handling Vendor Specific HCI Events.

CONFIG_BT_NIMBLE_OPTIMIZE_MULTI_CONN

Enable the optimization of multi-connection

Found in: Component config > Bluetooth > NimBLE Options

This option enables the use of vendor-specific APIs for multi-connections, which can greatly enhance the stability of coexistence between numerous central and peripheral devices. It will prohibit the usage of standard APIs.

Default value:

- No (disabled) if SOC_BLE_MULTI_CONN_OPTIMIZATION && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_ENC_ADV_DATA

Encrypted Advertising Data

Found in: Component config > Bluetooth > NimBLE Options

This option is used to enable encrypted advertising data.

CONFIG_BT_NIMBLE_MAX_EADS

Maximum number of EAD devices to save across reboots

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_ENC_ADV_DATA

Defines maximum number of encrypted advertising data key material to save

Default value:

- 10 if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENC_ADV_DATA* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_HIGH_DUTY_ADV_ITVL

Enable BLE high duty advertising interval feature

Found in: Component config > Bluetooth > NimBLE Options

This enable BLE high duty advertising interval feature

CONFIG_BT_NIMBLE_HOST_ALLOW_CONNECT_WITH_SCAN

Allow Connections with scanning in progress

Found in: Component config > Bluetooth > NimBLE Options

This enables support for user to initiate a new connection with scan in progress

CONFIG_BT_NIMBLE_HOST_QUEUE_CONG_CHECK

BLE queue congestion check

Found in: Component config > Bluetooth > NimBLE Options

When scanning and scan duplicate is not enabled, if there are a lot of adv packets around or application layer handling adv packets is slow, it will cause the controller memory to run out. if enabled, adv packets will be lost when host queue is congested.

Default value:

- No (disabled) if *CONFIG_BT_NIMBLE_ENABLED* && *CONFIG_BT_NIMBLE_ENABLED*

Host-controller Transport Contains:

- *CONFIG_BT_NIMBLE_TRANSPORT_UART*
- *CONFIG_BT_NIMBLE_HCI_UART_CTS_PIN*
- *CONFIG_BT_NIMBLE_USE_HCI_UART_FLOW_CTRL*
- *CONFIG_BT_NIMBLE_HCI_UART_RTS_PIN*

CONFIG_BT_NIMBLE_TRANSPORT_UART

Enable Uart Transport

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport

Use UART transport

Default value:

- Yes (enabled) if *CONFIG_BT_CONTROLLER_DISABLED* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_TRANSPORT_UART_PORT

Uart port

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport > CONFIG_BT_NIMBLE_TRANSPORT_UART

Uart port

Default value:

- 1 if *CONFIG_BT_CONTROLLER_DISABLED* && *CONFIG_BT_NIMBLE_TRANSPORT_UART* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_HCI_USE_UART_BAUDRATE

Uart Hci Baud Rate

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport > CONFIG_BT_NIMBLE_TRANSPORT_UART

Uart Baud Rate

Available options:

- 115200 (CONFIG_UART_BAUDRATE_115200)
- 230400 (CONFIG_UART_BAUDRATE_230400)
- 460800 (CONFIG_UART_BAUDRATE_460800)
- 921600 (CONFIG_UART_BAUDRATE_921600)

CONFIG_BT_NIMBLE_USE_HCI_UART_PARITY

Uart PARITY

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport > CONFIG_BT_NIMBLE_TRANSPORT_UART

Uart Parity

Available options:

- None (CONFIG_UART_PARITY_NONE)
- Odd (CONFIG_UART_PARITY_ODD)
- Even (CONFIG_UART_PARITY_EVEN)

CONFIG_BT_NIMBLE_UART_RX_PIN

UART Rx pin

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport > CONFIG_BT_NIMBLE_TRANSPORT_UART

Rx pin for Nimble Transport

Default value:

- 5 if *CONFIG_BT_CONTROLLER_DISABLED* && *CONFIG_BT_NIMBLE_TRANSPORT_UART* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_UART_TX_PIN

UART Tx pin

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport > CONFIG_BT_NIMBLE_TRANSPORT_UART

Tx pin for Nimble Transport

Default value:

- 4 if *CONFIG_BT_CONTROLLER_DISABLED* && *CONFIG_BT_NIMBLE_TRANSPORT_UART* && *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_USE_HCI_UART_FLOW_CTRL

Uart Flow Control

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport

Uart Flow Control

Available options:

- Disable (CONFIG_UART_HW_FLOWCTRL_DISABLE)
- Enable hardware flow control (CONFIG_UART_HW_FLOWCTRL_CTS_RTS)

CONFIG_BT_NIMBLE_HCI_UART_RTS_PIN

UART Rts Pin

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport

UART HCI RTS pin

Default value:

- 19 if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_NIMBLE_HCI_UART_CTS_PIN

UART Cts Pin

Found in: Component config > Bluetooth > NimBLE Options > Host-controller Transport

UART HCI CTS pin

Default value:

- 23 if *CONFIG_BT_NIMBLE_ENABLED*

Controller Options

CONFIG_BT_RELEASE_IRAM

Release Bluetooth text (READ DOCS FIRST)

Found in: Component config > Bluetooth

This option release Bluetooth text section and merge Bluetooth data, bss & text into a large free heap region when esp_bt_mem_release is called, total saving ~21kB or more of IRAM. ESP32-C2 only 3 configurable PMP entries available, rest of them are hard-coded. We cannot split the memory into 3 different regions (IRAM, BLE-IRAM, DRAM). So this option will disable the PMP (ESP_SYSTEM_PMP_IDRAM_SPLIT)

Default value:

- No (disabled) if *CONFIG_BT_ENABLED* && BT_LE_RELEASE_IRAM_SUPPORTED

Common Options Contains:

- *CONFIG_BT_ALARM_MAX_NUM*

CONFIG_BT_ALARM_MAX_NUM

Maximum number of Bluetooth alarms

Found in: Component config > Bluetooth > Common Options

This option decides the maximum number of alarms which could be used by Bluetooth host.

Default value:

- 50

CONFIG_BT_HCI_LOG_DEBUG_EN

Enable Bluetooth HCI debug mode

Found in: Component config > Bluetooth

This option is used to enable bluetooth debug mode, which saves the hci layer data stream.

Default value:

- No (disabled) if *CONFIG_BT_BLUEDROID_ENABLED* || *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_BT_HCI_LOG_DATA_BUFFER_SIZE

Size of the cache used for HCI data in Bluetooth HCI debug mode (N*1024 bytes)

Found in: Component config > Bluetooth > CONFIG_BT_HCI_LOG_DEBUG_EN

This option is to configure the buffer size of the hci data steam cache in hci debug mode. This is a ring buffer, the new data will overwrite the oldest data if the buffer is full.

Range:

- from 1 to 100 if *CONFIG_BT_HCI_LOG_DEBUG_EN*

Default value:

- 5 if *CONFIG_BT_HCI_LOG_DEBUG_EN*

CONFIG_BT_HCI_LOG_ADV_BUFFER_SIZE

Size of the cache used for adv report in Bluetooth HCI debug mode (N*1024 bytes)

Found in: Component config > Bluetooth > CONFIG_BT_HCI_LOG_DEBUG_EN

This option is to configure the buffer size of the hci adv report cache in hci debug mode. This is a ring buffer, the new data will overwrite the oldest data if the buffer is full.

Range:

- from 1 to 100 if *CONFIG_BT_HCI_LOG_DEBUG_EN*

Default value:

- 8 if *CONFIG_BT_HCI_LOG_DEBUG_EN*

CONFIG_BLE_MESH

ESP BLE Mesh Support

Found in: Component config

This option enables ESP BLE Mesh support. The specific features that are available may depend on other features that have been enabled in the stack, such as Bluetooth Support, Bluedroid Support & GATT support.

Contains:

- *BLE Mesh and BLE coexistence support*
- *CONFIG_BLE_MESH_GATT_PROXY_CLIENT*
- *CONFIG_BLE_MESH_GATT_PROXY_SERVER*
- *BLE Mesh NET BUF DEBUG LOG LEVEL*
- *CONFIG_BLE_MESH_PROV*
- *CONFIG_BLE_MESH_PROXY*
- *BLE Mesh specific test option*
- *BLE Mesh STACK DEBUG LOG LEVEL*
- *CONFIG_BLE_MESH_NO_LOG*
- *CONFIG_BLE_MESH_IVU_DIVIDER*
- *CONFIG_BLE_MESH_FAST_PROV*
- *CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC*
- *CONFIG_BLE_MESH_EXPERIMENTAL*
- *CONFIG_BLE_MESH_CRPL*
- *CONFIG_BLE_MESH_RX_SDU_MAX*
- *CONFIG_BLE_MESH_MODEL_KEY_COUNT*
- *CONFIG_BLE_MESH_APP_KEY_COUNT*
- *CONFIG_BLE_MESH_MODEL_GROUP_COUNT*
- *CONFIG_BLE_MESH_LABEL_COUNT*
- *CONFIG_BLE_MESH_SUBNET_COUNT*
- *CONFIG_BLE_MESH_TX_SEG_MAX*
- *CONFIG_BLE_MESH_RX_SEG_MSG_COUNT*
- *CONFIG_BLE_MESH_TX_SEG_MSG_COUNT*
- *CONFIG_BLE_MESH_MEM_ALLOC_MODE*
- *CONFIG_BLE_MESH_MSG_CACHE_SIZE*
- *CONFIG_BLE_MESH_NOT_RELAY_REPLAY_MSG*
- *CONFIG_BLE_MESH_ADV_BUF_COUNT*
- *CONFIG_BLE_MESH_PB_GATT*
- *CONFIG_BLE_MESH_PB_ADV*
- *CONFIG_BLE_MESH_IVU_RECOVERY_IVI*
- *CONFIG_BLE_MESH_RELAY*
- *CONFIG_BLE_MESH_SAR_ENHANCEMENT*
- *CONFIG_BLE_MESH_SETTINGS*
- *CONFIG_BLE_MESH_ACTIVE_SCAN*
- *CONFIG_BLE_MESH_DEINIT*
- *CONFIG_BLE_MESH_USE_DUPLICATE_SCAN*
- *CONFIG_BLE_MESH_V11_SUPPORT*
- *Support for BLE Mesh Client/Server models*
- *Support for BLE Mesh Foundation models*
- *CONFIG_BLE_MESH_NODE*
- *CONFIG_BLE_MESH_PROVISIONER*
- *CONFIG_BLE_MESH_FRIEND*
- *CONFIG_BLE_MESH_LOW_POWER*
- *CONFIG_BLE_MESH_HCI_5_0*

- *CONFIG_BLE_MESH_RANDOM_ADV_INTERVAL*
- *CONFIG_BLE_MESH_IV_UPDATE_TEST*
- *CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT*

CONFIG_BLE_MESH_HCI_5_0

Support sending 20ms non-connectable adv packets

Found in: Component config > CONFIG_BLE_MESH

It is a temporary solution and needs further modifications.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_V11_SUPPORT

Support ESP BLE Mesh v1.1 features (Preview)

Found in: Component config > CONFIG_BLE_MESH

Support BLE Mesh v1.1 features

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RANDOM_ADV_INTERVAL

Support using random adv interval for mesh packets

Found in: Component config > CONFIG_BLE_MESH

Enable this option to allow using random advertising interval for mesh packets. And this could help avoid collision of advertising packets.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_USE_DUPLICATE_SCAN

Support Duplicate Scan in BLE Mesh

Found in: Component config > CONFIG_BLE_MESH

Enable this option to allow using specific duplicate scan filter in BLE Mesh, and Scan Duplicate Type must be set by choosing the option in the Bluetooth Controller section in menuconfig, which is ”Scan Duplicate By Device Address and Advertising Data”.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_ACTIVE_SCAN

Support Active Scan in BLE Mesh

Found in: Component config > CONFIG_BLE_MESH

Enable this option to allow using BLE Active Scan for BLE Mesh.

CONFIG_BLE_MESH_MEM_ALLOC_MODE

Memory allocation strategy

Found in: Component config > CONFIG_BLE_MESH

Allocation strategy for BLE Mesh stack, essentially provides ability to allocate all required dynamic allocations from,

- Internal DRAM memory only
- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF
- Internal IRAM memory wherever applicable else internal DRAM

Recommended mode here is always internal (*), since that is most preferred from security perspective. But if application requirement does not allow sufficient free internal memory then alternate mode can be selected.

(*) In case of ESP32-S2/ESP32-S3, hardware allows encryption of external SPIRAM contents provided hardware flash encryption feature is enabled. In that case, using external SPIRAM allocation strategy is also safe choice from security perspective.

Available options:

- Internal DRAM (CONFIG_BLE_MESH_MEM_ALLOC_MODE_INTERNAL)
- External SPIRAM (CONFIG_BLE_MESH_MEM_ALLOC_MODE_EXTERNAL)
- Default alloc mode (CONFIG_BLE_MESH_MEM_ALLOC_MODE_DEFAULT)
  Enable this option to use the default memory allocation strategy when external SPIRAM is enabled. See the SPIRAM options for more details.
- Internal IRAM (CONFIG_BLE_MESH_MEM_ALLOC_MODE_IRAM_8BIT)
  Allows to use IRAM memory region as 8bit accessible region. Every unaligned (8bit or 16bit) access will result in an exception and incur penalty of certain clock cycles per unaligned read/write.

CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC

Enable FreeRTOS static allocation

Found in: Component config > CONFIG_BLE_MESH

Enable this option to use FreeRTOS static allocation APIs for BLE Mesh, which provides the ability to use different dynamic memory (i.e. SPIRAM or IRAM) for FreeRTOS objects. If this option is disabled, the FreeRTOS static allocation APIs will not be used, and internal DRAM will be allocated for FreeRTOS objects.

Default value:

- No (disabled) if ESP32_IRAM_AS_8BIT_ACCESSIBLE_MEMORY && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC_MODE

Memory allocation for FreeRTOS objects

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC

Choose the memory to be used for FreeRTOS objects.

Available options:

- External SPIRAM (CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC_EXTERNAL)
  If enabled, BLE Mesh allocates dynamic memory from external SPIRAM for FreeRTOS objects, i.e. mutex, queue, and task stack. External SPIRAM can only be used for task stack when SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY is enabled. See the SPIRAM options for more details.
- Internal IRAM (CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC_IRAM_8BIT)
  If enabled, BLE Mesh allocates dynamic memory from internal IRAM for FreeRTOS objects, i.e. mutex, queue. Note: IRAM region cannot be used as task stack.

CONFIG_BLE_MESH_DEINIT

Support de-initialize BLE Mesh stack

Found in: Component config > CONFIG_BLE_MESH

If enabled, users can use the function esp_ble_mesh_deinit() to de-initialize the whole BLE Mesh stack.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

BLE Mesh and BLE coexistence support Contains:

- *CONFIG_BLE_MESH_SUPPORT_BLE_SCAN*
- *CONFIG_BLE_MESH_SUPPORT_BLE_ADV*

CONFIG_BLE_MESH_SUPPORT_BLE_ADV

Support sending normal BLE advertising packets

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh and BLE coexistence support

When selected, users can send normal BLE advertising packets with specific API.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_BLE_ADV_BUF_COUNT

Number of advertising buffers for BLE advertising packets

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh and BLE coexistence support > CONFIG_BLE_MESH_SUPPORT_BLE_ADV

Number of advertising buffers for BLE packets available.

Range:

- from 1 to 255 if *CONFIG_BLE_MESH_SUPPORT_BLE_ADV* && *CONFIG_BLE_MESH*

Default value:

- 3 if *CONFIG_BLE_MESH_SUPPORT_BLE_ADV* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SUPPORT_BLE_SCAN

Support scanning normal BLE advertising packets

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh and BLE coexistence support

When selected, users can register a callback and receive normal BLE advertising packets in the application layer.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_FAST_PROV

Enable BLE Mesh Fast Provisioning

Found in: Component config > CONFIG_BLE_MESH

Enable this option to allow BLE Mesh fast provisioning solution to be used. When there are multiple unprovisioned devices around, fast provisioning can greatly reduce the time consumption of the whole provisioning process. When this option is enabled, and after an unprovisioned device is provisioned into a node successfully, it can be changed to a temporary Provisioner.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_NODE

Support for BLE Mesh Node

Found in: Component config > CONFIG_BLE_MESH

Enable the device to be provisioned into a node. This option should be enabled when an unprovisioned device is going to be provisioned into a node and communicate with other nodes in the BLE Mesh network.

CONFIG_BLE_MESH_PROVISIONER

Support for BLE Mesh Provisioner

Found in: Component config > CONFIG_BLE_MESH

Enable the device to be a Provisioner. The option should be enabled when a device is going to act as a Provisioner and provision unprovisioned devices into the BLE Mesh network.

CONFIG_BLE_MESH_WAIT_FOR_PROV_MAX_DEV_NUM

Maximum number of unprovisioned devices that can be added to device queue

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

This option specifies how many unprovisioned devices can be added to device queue for provisioning. Users can use this option to define the size of the queue in the bottom layer which is used to store unprovisioned device information (e.g. Device UUID, address).

Range:

- from 1 to 100 if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

Default value:

- 10 if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MAX_PROV_NODES

Maximum number of devices that can be provisioned by Provisioner

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

This option specifies how many devices can be provisioned by a Provisioner. This value indicates the maximum number of unprovisioned devices which can be provisioned by a Provisioner. For instance, if the value is 6, it means the Provisioner can provision up to 6 unprovisioned devices. Theoretically a Provisioner without the limitation of its memory can provision up to 32766 unprovisioned devices, here we limit the maximum number to 100 just to limit the memory used by a Provisioner. The bigger the value is, the more memory it will cost by a Provisioner to store the information of nodes.

Range:

- from 1 to 1000 if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

Default value:

- 10 if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PBA_SAME_TIME

Maximum number of PB-ADV running at the same time by Provisioner

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many devices can be provisioned at the same time using PB-ADV. For examples, if the value is 2, it means a Provisioner can provision two unprovisioned devices with PB-ADV at the same time.

Range:

- from 1 to 10 if *CONFIG_BLE_MESH_PB_ADV* && *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

Default value:

- 2 if *CONFIG_BLE_MESH_PB_ADV* && *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PBG_SAME_TIME

Maximum number of PB-GATT running at the same time by Provisioner

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many devices can be provisioned at the same time using PB-GATT. For example, if the value is 2, it means a Provisioner can provision two unprovisioned devices with PB-GATT at the same time.

Range:

- from 1 to 5 if *CONFIG_BLE_MESH_PB_GATT* && *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

Default value:

- 1 if *CONFIG_BLE_MESH_PB_GATT* && *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROVISIONER_SUBNET_COUNT

Maximum number of mesh subnets that can be created by Provisioner

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many subnets per network a Provisioner can create. Indeed, this value decides the number of network keys which can be added by a Provisioner.

Range:

- from 1 to 4096 if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

Default value:

- 3 if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROVISIONER_APP_KEY_COUNT

Maximum number of application keys that can be owned by Provisioner

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many application keys the Provisioner can have. Indeed, this value decides the number of the application keys which can be added by a Provisioner.

Range:

- from 1 to 4096 if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

Default value:

- 3 if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROVISIONER_RECV_HB

Support receiving Heartbeat messages

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

When this option is enabled, Provisioner can call specific functions to enable or disable receiving Heartbeat messages and notify them to the application layer.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROVISIONER_RECV_HB_FILTER_SIZE

Maximum number of filter entries for receiving Heartbeat messages

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER > CONFIG_BLE_MESH_PROVISIONER_RECV_HB

This option specifies how many heartbeat filter entries Provisioner supports. The heartbeat filter (acceptlist or rejectlist) entries are used to store a list of SRC and DST which can be used to decide if a heartbeat message will be processed and notified to the application layer by Provisioner. Note: The filter is an empty rejectlist by default.

Range:

- from 1 to 1000 if *CONFIG_BLE_MESH_PROVISIONER_RECV_HB* && *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

Default value:

- 3 if *CONFIG_BLE_MESH_PROVISIONER_RECV_HB* && *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROV

BLE Mesh Provisioning support

Found in: Component config > CONFIG_BLE_MESH

Enable this option to support BLE Mesh Provisioning functionality. For BLE Mesh, this option should be always enabled.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROV_EPA

BLE Mesh enhanced provisioning authentication

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROV

Enable this option to support BLE Mesh enhanced provisioning authentication functionality. This option can increase the security level of provisioning. It is recommended to enable this option.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH_PROV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_CERT_BASED_PROV

Support Certificate-based provisioning

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROV

Enable this option to support BLE Mesh Certificate-Based Provisioning.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_PROV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RECORD_FRAG_MAX_SIZE

Maximum size of the provisioning record fragment that Provisioner can receive

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_PROV* > *CONFIG_BLE_MESH_CERT_BASED_PROV*

This option sets the maximum size of the provisioning record fragment that the Provisioner can receive. The range depends on provisioning bearer.

Range:

- from 1 to 57 if *CONFIG_BLE_MESH_CERT_BASED_PROV* && *CONFIG_BLE_MESH*

Default value:

- 56 if *CONFIG_BLE_MESH_CERT_BASED_PROV* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PB_ADV

Provisioning support using the advertising bearer (PB-ADV)

Found in: Component config > *CONFIG_BLE_MESH*

Enable this option to allow the device to be provisioned over the advertising bearer. This option should be enabled if PB-ADV is going to be used during provisioning procedure.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_UNPROVISIONED_BEACON_INTERVAL

Interval between two consecutive Unprovisioned Device Beacon

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_PB_ADV*

This option specifies the interval of sending two consecutive unprovisioned device beacon, users can use this option to change the frequency of sending unprovisioned device beacon. For example, if the value is 5, it means the unprovisioned device beacon will send every 5 seconds. When the option of BLE_MESH_FAST_PROV is selected, the value is better to be 3 seconds, or less.

Range:

- from 1 to 100 if *CONFIG_BLE_MESH_NODE* && *CONFIG_BLE_MESH_PB_ADV* && *CONFIG_BLE_MESH*

Default value:

- 5 if *CONFIG_BLE_MESH_NODE* && *CONFIG_BLE_MESH_PB_ADV* && *CONFIG_BLE_MESH*
- 3 if *CONFIG_BLE_MESH_FAST_PROV* && *CONFIG_BLE_MESH_NODE* && *CONFIG_BLE_MESH_PB_ADV* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PB_GATT

Provisioning support using GATT (PB-GATT)

Found in: Component config > *CONFIG_BLE_MESH*

Enable this option to allow the device to be provisioned over GATT. This option should be enabled if PB-GATT is going to be used during provisioning procedure.

Virtual option enabled whenever any Proxy protocol is needed

CONFIG_BLE_MESH_PROXY

BLE Mesh Proxy protocol support

Found in: Component config > CONFIG_BLE_MESH

Enable this option to support BLE Mesh Proxy protocol used by PB-GATT and other proxy pdu transmission.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_GATT_PROXY_SERVER

BLE Mesh GATT Proxy Server

Found in: Component config > CONFIG_BLE_MESH

This option enables support for Mesh GATT Proxy Service, i.e. the ability to act as a proxy between a Mesh GATT Client and a Mesh network. This option should be enabled if a node is going to be a Proxy Server.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH_NODE* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_NODE_ID_TIMEOUT

Node Identity advertising timeout

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_GATT_PROXY_SERVER

This option determines for how long the local node advertises using Node Identity. The given value is in seconds. The specification limits this to 60 seconds and lists it as the recommended value as well. So leaving the default value is the safest option. When an unprovisioned device is provisioned successfully and becomes a node, it will start to advertise using Node Identity during the time set by this option. And after that, Network ID will be advertised.

Range:

- from 1 to 60 if *CONFIG_BLE_MESH_GATT_PROXY_SERVER* && *CONFIG_BLE_MESH*

Default value:

- 60 if *CONFIG_BLE_MESH_GATT_PROXY_SERVER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROXY_FILTER_SIZE

Maximum number of filter entries per Proxy Client

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_GATT_PROXY_SERVER

This option specifies how many Proxy Filter entries the local node supports. The entries of Proxy filter (whitelist or blacklist) are used to store a list of addresses which can be used to decide which messages will be forwarded to the Proxy Client by the Proxy Server.

Range:

- from 1 to 32767 if *CONFIG_BLE_MESH_GATT_PROXY_SERVER* && *CONFIG_BLE_MESH*

Default value:

- 4 if *CONFIG_BLE_MESH_GATT_PROXY_SERVER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROXY_PRIVACY

Support Proxy Privacy

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_GATT_PROXY_SERVER

The Proxy Privacy parameter controls the privacy of the Proxy Server over the connection. The value of the Proxy Privacy parameter is controlled by the type of proxy connection, which is dependent on the bearer used by the proxy connection.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH_PRB_SRV* && *CONFIG_BLE_MESH_GATT_PROXY_SERVER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROXY_SOLIC_PDU_RX

Support receiving Proxy Solicitation PDU

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_GATT_PROXY_SERVER*

Enable this option to support receiving Proxy Solicitation PDU.

CONFIG_BLE_MESH_PROXY_SOLIC_RX_CRPL

Maximum capacity of solicitation replay protection list

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_GATT_PROXY_SERVER* > *CONFIG_BLE_MESH_PROXY_SOLIC_PDU_RX*

This option specifies the maximum capacity of the solicitation replay protection list. The solicitation replay protection list is used to reject Solicitation PDUs that were already processed by a node, which will store the solicitation src and solicitation sequence number of the received Solicitation PDU message.

Range:

- from 1 to 255 if *CONFIG_BLE_MESH_PROXY_SOLIC_PDU_RX* && *CONFIG_BLE_MESH*

Default value:

- 2 if *CONFIG_BLE_MESH_PROXY_SOLIC_PDU_RX* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_GATT_PROXY_CLIENT

BLE Mesh GATT Proxy Client

Found in: Component config > *CONFIG_BLE_MESH*

This option enables support for Mesh GATT Proxy Client. The Proxy Client can use the GATT bearer to send mesh messages to a node that supports the advertising bearer.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PROXY_SOLIC_PDU_TX

Support sending Proxy Solicitation PDU

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_GATT_PROXY_CLIENT*

Enable this option to support sending Proxy Solicitation PDU.

CONFIG_BLE_MESH_PROXY_SOLIC_TX_SRC_COUNT

Maximum number of SSRC that can be used by Proxy Client

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_GATT_PROXY_CLIENT* > *CONFIG_BLE_MESH_PROXY_SOLIC_PDU_TX*

This option specifies the maximum number of Solicitation Source (SSRC) that can be used by Proxy Client for sending a Solicitation PDU. A Proxy Client may use the primary address or any of the secondary addresses as the SSRC for a Solicitation PDU. So for a Proxy Client, it's better to choose the value based on its own element count.

Range:
- from 1 to 16 if *CONFIG_BLE_MESH_PROXY_SOLIC_PDU_TX* && *CONFIG_BLE_MESH*

Default value:
- 2 if *CONFIG_BLE_MESH_PROXY_SOLIC_PDU_TX* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SETTINGS

Store BLE Mesh configuration persistently

Found in: Component config > CONFIG_BLE_MESH

When selected, the BLE Mesh stack will take care of storing/restoring the BLE Mesh configuration persistently in flash. If the device is a BLE Mesh node, when this option is enabled, the configuration of the device will be stored persistently, including unicast address, NetKey, AppKey, etc. And if the device is a BLE Mesh Provisioner, the information of the device will be stored persistently, including the information of provisioned nodes, NetKey, AppKey, etc.

Default value:
- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_STORE_TIMEOUT

Delay (in seconds) before storing anything persistently

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

This value defines in seconds how soon any pending changes are actually written into persistent storage (flash) after a change occurs. The option allows nodes to delay a certain period of time to save proper information to flash. The default value is 0, which means information will be stored immediately once there are updates.

Range:
- from 0 to 1000000 if *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

Default value:
- 0 if *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SEQ_STORE_RATE

How often the sequence number gets updated in storage

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

This value defines how often the local sequence number gets updated in persistent storage (i.e. flash). e.g. a value of 100 means that the sequence number will be stored to flash on every 100th increment. If the node sends messages very frequently a higher value makes more sense, whereas if the node sends infrequently a value as low as 0 (update storage for every increment) can make sense. When the stack gets initialized it will add sequence number to the last stored one, so that it starts off with a value that's guaranteed to be larger than the last one used before power off.

Range:
- from 0 to 1000000 if *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

Default value:
- 0 if *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RPL_STORE_TIMEOUT

Minimum frequency that the RPL gets updated in storage

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

This value defines in seconds how soon the RPL (Replay Protection List) gets written to persistent storage after a change occurs. If the node receives messages frequently, then a large value is recommended. If the node receives messages rarely, then the value can be as low as 0 (which means the RPL is written

into the storage immediately). Note that if the node operates in a security-sensitive case, and there is a risk of sudden power-off, then a value of 0 is strongly recommended. Otherwise, a power loss before RPL being written into the storage may introduce message replay attacks and system security will be in a vulnerable state.

Range:

- from 0 to 1000000 if *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

Default value:

- 0 if *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SETTINGS_BACKWARD_COMPATIBILITY

A specific option for settings backward compatibility

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_SETTINGS*

This option is created to solve the issue of failure in recovering node information after mesh stack updates. In the old version mesh stack, there is no key of ”mesh/role” in nvs. In the new version mesh stack, key of ”mesh/role” is added in nvs, recovering node information needs to check ”mesh/role” key in nvs and implements selective recovery of mesh node information. Therefore, there may be failure in recovering node information during node restarting after OTA.

The new version mesh stack adds the option of ”mesh/role” because we have added the support of storing Provisioner information, while the old version only supports storing node information.

If users are updating their nodes from old version to new version, we recommend enabling this option, so that system could set the flag in advance before recovering node information and make sure the node information recovering could work as expected.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_NODE* && *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SPECIFIC_PARTITION

Use a specific NVS partition for BLE Mesh

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_SETTINGS*

When selected, the mesh stack will use a specified NVS partition instead of default NVS partition. Note that the specified partition must be registered with NVS using nvs_flash_init_partition() API, and the partition must exists in the csv file. When Provisioner needs to store a large amount of nodes' information in the flash (e.g. more than 20), this option is recommended to be enabled.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PARTITION_NAME

Name of the NVS partition for BLE Mesh

Found in: Component config > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_SETTINGS* > *CONFIG_BLE_MESH_SPECIFIC_PARTITION*

This value defines the name of the specified NVS partition used by the mesh stack.

Default value:

- ”ble_mesh” if *CONFIG_BLE_MESH_SPECIFIC_PARTITION* && *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_USE_MULTIPLE_NAMESPACE

Support using multiple NVS namespaces by Provisioner

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

When selected, Provisioner can use different NVS namespaces to store different instances of mesh information. For example, if in the first room, Provisioner uses NetKey A, AppKey A and provisions three devices, these information will be treated as mesh information instance A. When the Provisioner moves to the second room, it uses NetKey B, AppKey B and provisions two devices, then the information will be treated as mesh information instance B. Here instance A and instance B will be stored in different namespaces. With this option enabled, Provisioner needs to use specific functions to open the corresponding NVS namespace, restore the mesh information, release the mesh information or erase the mesh information.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_PROVISIONER* && *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MAX_NVS_NAMESPACE

Maximum number of NVS namespaces

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS > CONFIG_BLE_MESH_USE_MULTIPLE_NAMESPACE

This option specifies the maximum NVS namespaces supported by Provisioner.

Range:

- from 1 to 255 if *CONFIG_BLE_MESH_USE_MULTIPLE_NAMESPACE* && *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

Default value:

- 2 if *CONFIG_BLE_MESH_USE_MULTIPLE_NAMESPACE* && *CONFIG_BLE_MESH_SETTINGS* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SUBNET_COUNT

Maximum number of mesh subnets per network

Found in: Component config > CONFIG_BLE_MESH

This option specifies how many subnets a Mesh network can have at the same time. Indeed, this value decides the number of the network keys which can be owned by a node.

Range:

- from 1 to 4096 if *CONFIG_BLE_MESH*

Default value:

- 3 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_APP_KEY_COUNT

Maximum number of application keys per network

Found in: Component config > CONFIG_BLE_MESH

This option specifies how many application keys the device can store per network. Indeed, this value decides the number of the application keys which can be owned by a node.

Range:

- from 1 to 4096 if *CONFIG_BLE_MESH*

Default value:

- 3 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MODEL_KEY_COUNT

Maximum number of application keys per model

Found in: Component config > CONFIG_BLE_MESH

This option specifies the maximum number of application keys to which each model can be bound.

Range:
- from 1 to 4096 if *CONFIG_BLE_MESH*

Default value:
- 3 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MODEL_GROUP_COUNT

Maximum number of group address subscriptions per model

Found in: Component config > CONFIG_BLE_MESH

This option specifies the maximum number of addresses to which each model can be subscribed.

Range:
- from 1 to 4096 if *CONFIG_BLE_MESH*

Default value:
- 3 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LABEL_COUNT

Maximum number of Label UUIDs used for Virtual Addresses

Found in: Component config > CONFIG_BLE_MESH

This option specifies how many Label UUIDs can be stored. Indeed, this value decides the number of the Virtual Addresses can be supported by a node.

Range:
- from 0 to 4096 if *CONFIG_BLE_MESH*

Default value:
- 3 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_CRPL

Maximum capacity of the replay protection list

Found in: Component config > CONFIG_BLE_MESH

This option specifies the maximum capacity of the replay protection list. It is similar to Network message cache size, but has a different purpose. The replay protection list is used to prevent a node from replay attack, which will store the source address and sequence number of the received mesh messages. For Provisioner, the replay protection list size should not be smaller than the maximum number of nodes whose information can be stored. And the element number of each node should also be taken into consideration. For example, if Provisioner can provision up to 20 nodes and each node contains two elements, then the replay protection list size of Provisioner should be at least 40.

Range:
- from 2 to 65535 if *CONFIG_BLE_MESH*

Default value:
- 10 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_NOT_RELAY_REPLAY_MSG

Not relay replayed messages in a mesh network

Found in: Component config > CONFIG_BLE_MESH

There may be many expired messages in a complex mesh network that would be considered replayed messages. Enable this option will refuse to relay such messages, which could help to reduce invalid packets in the mesh network. However, it should be noted that enabling this option may result in packet loss in certain environments. Therefore, users need to decide whether to enable this option according to the actual usage situation.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_EXPERIMENTAL* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MSG_CACHE_SIZE

Network message cache size

Found in: Component config > CONFIG_BLE_MESH

Number of messages that are cached for the network. This helps prevent unnecessary decryption operations and unnecessary relays. This option is similar to Replay protection list, but has a different purpose. A node is not required to cache the entire Network PDU and may cache only part of it for tracking, such as values for SRC/SEQ or others.

Range:

- from 2 to 65535 if *CONFIG_BLE_MESH*

Default value:

- 10 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_ADV_BUF_COUNT

Number of advertising buffers

Found in: Component config > CONFIG_BLE_MESH

Number of advertising buffers available. The transport layer reserves ADV_BUF_COUNT - 3 buffers for outgoing segments. The maximum outgoing SDU size is 12 times this value (out of which 4 or 8 bytes are used for the Transport Layer MIC). For example, 5 segments means the maximum SDU size is 60 bytes, which leaves 56 bytes for application layer data using a 4-byte MIC, or 52 bytes using an 8-byte MIC.

Range:

- from 6 to 256 if *CONFIG_BLE_MESH*

Default value:

- 60 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_IVU_DIVIDER

Divider for IV Update state refresh timer

Found in: Component config > CONFIG_BLE_MESH

When the IV Update state enters Normal operation or IV Update in Progress, we need to keep track of how many hours has passed in the state, since the specification requires us to remain in the state at least for 96 hours (Update in Progress has an additional upper limit of 144 hours).

In order to fulfill the above requirement, even if the node might be powered off once in a while, we need to store persistently how many hours the node has been in the state. This doesn't necessarily need to happen every hour (thanks to the flexible duration range). The exact cadence will depend a lot on the ways that the node will be used and what kind of power source it has.

Since there is no single optimal answer, this configuration option allows specifying a divider, i.e. how many intervals the 96 hour minimum gets split into. After each interval the duration that the node has

been in the current state gets stored to flash. E.g. the default value of 4 means that the state is saved every 24 hours (96 / 4).

Range:
- from 2 to 96 if *CONFIG_BLE_MESH*

Default value:
- 4 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_IVU_RECOVERY_IVI

Recovery the IV index when the latest whole IV update procedure is missed

Found in: Component config > CONFIG_BLE_MESH

According to Section 3.10.5 of Mesh Specification v1.0.1. If a node in Normal Operation receives a Secure Network beacon with an IV index equal to the last known IV index+1 and the IV Update Flag set to 0, the node may update its IV without going to the IV Update in Progress state, or it may initiate an IV Index Recovery procedure (Section 3.10.6), or it may ignore the Secure Network beacon. The node makes the choice depending on the time since last IV update and the likelihood that the node has missed the Secure Network beacons with the IV update Flag. When the above situation is encountered, this option can be used to decide whether to perform the IV index recovery procedure.

Default value:
- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SAR_ENHANCEMENT

Segmentation and reassembly enhancement

Found in: Component config > CONFIG_BLE_MESH

Enable this option to use the enhanced segmentation and reassembly mechanism introduced in Bluetooth Mesh Protocol 1.1.

Default value:
- No (disabled) if *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_TX_SEG_MSG_COUNT

Maximum number of simultaneous outgoing segmented messages

Found in: Component config > CONFIG_BLE_MESH

Maximum number of simultaneous outgoing multi-segment and/or reliable messages. The default value is 1, which means the device can only send one segmented message at a time. And if another segmented message is going to be sent, it should wait for the completion of the previous one. If users are going to send multiple segmented messages at the same time, this value should be configured properly.

Range:
- from 1 to if *CONFIG_BLE_MESH*

Default value:
- 1 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RX_SEG_MSG_COUNT

Maximum number of simultaneous incoming segmented messages

Found in: Component config > CONFIG_BLE_MESH

Maximum number of simultaneous incoming multi-segment and/or reliable messages. The default value is 1, which means the device can only receive one segmented message at a time. And if another segmented message is going to be received, it should wait for the completion of the previous one. If users

are going to receive multiple segmented messages at the same time, this value should be configured properly.

Range:
- from 1 to 255 if *CONFIG_BLE_MESH*

Default value:
- 1 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RX_SDU_MAX

Maximum incoming Upper Transport Access PDU length

Found in: Component config > CONFIG_BLE_MESH

Maximum incoming Upper Transport Access PDU length. Leave this to the default value, unless you really need to optimize memory usage.

Range:
- from 36 to 384 if *CONFIG_BLE_MESH*

Default value:
- 384 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_TX_SEG_MAX

Maximum number of segments in outgoing messages

Found in: Component config > CONFIG_BLE_MESH

Maximum number of segments supported for outgoing messages. This value should typically be fine-tuned based on what models the local node supports, i.e. what's the largest message payload that the node needs to be able to send. This value affects memory and call stack consumption, which is why the default is lower than the maximum that the specification would allow (32 segments).

The maximum outgoing SDU size is 12 times this number (out of which 4 or 8 bytes is used for the Transport Layer MIC). For example, 5 segments means the maximum SDU size is 60 bytes, which leaves 56 bytes for application layer data using a 4-byte MIC and 52 bytes using an 8-byte MIC.

Be sure to specify a sufficient number of advertising buffers when setting this option to a higher value. There must be at least three more advertising buffers (BLE_MESH_ADV_BUF_COUNT) as there are outgoing segments.

Range:
- from 2 to 32 if *CONFIG_BLE_MESH*

Default value:
- 32 if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RELAY

Relay support

Found in: Component config > CONFIG_BLE_MESH

Support for acting as a Mesh Relay Node. Enabling this option will allow a node to support the Relay feature, and the Relay feature can still be enabled or disabled by proper configuration messages. Disabling this option will let a node not support the Relay feature.

Default value:
- Yes (enabled) if *CONFIG_BLE_MESH_NODE* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RELAY_ADV_BUF

Use separate advertising buffers for relay packets

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_RELAY

When selected, self-send packets will be put in a high-priority queue and relay packets will be put in a low-priority queue.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_RELAY* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RELAY_ADV_BUF_COUNT

Number of advertising buffers for relay packets

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_RELAY > CONFIG_BLE_MESH_RELAY_ADV_BUF

Number of advertising buffers for relay packets available.

Range:

- from 6 to 256 if *CONFIG_BLE_MESH_RELAY_ADV_BUF* && *CONFIG_BLE_MESH_RELAY* && *CONFIG_BLE_MESH*

Default value:

- 60 if *CONFIG_BLE_MESH_RELAY_ADV_BUF* && *CONFIG_BLE_MESH_RELAY* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LOW_POWER

Support for Low Power features

Found in: Component config > CONFIG_BLE_MESH

Enable this option to operate as a Low Power Node. If low power consumption is required by a node, this option should be enabled. And once the node enters the mesh network, it will try to find a Friend node and establish a friendship.

CONFIG_BLE_MESH_LPN_ESTABLISHMENT

Perform Friendship establishment using low power

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Perform the Friendship establishment using low power with the help of a reduced scan duty cycle. The downside of this is that the node may miss out on messages intended for it until it has successfully set up Friendship with a Friend node. When this option is enabled, the node will stop scanning for a period of time after a Friend Request or Friend Poll is sent, so as to reduce more power consumption.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_AUTO

Automatically start looking for Friend nodes once provisioned

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Once provisioned, automatically enable LPN functionality and start looking for Friend nodes. If this option is disabled LPN mode needs to be manually enabled by calling bt_mesh_lpn_set(true). When an unprovisioned device is provisioned successfully and becomes a node, enabling this option will trigger the node starts to send Friend Request at a certain period until it finds a proper Friend node.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_AUTO_TIMEOUT

Time from last received message before going to LPN mode

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER > CONFIG_BLE_MESH_LPN_AUTO

Time in seconds from the last received message, that the node waits out before starting to look for Friend nodes.

Range:

- from 0 to 3600 if *CONFIG_BLE_MESH_LPN_AUTO* && *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:

- 15 if *CONFIG_BLE_MESH_LPN_AUTO* && *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_RETRY_TIMEOUT

Retry timeout for Friend requests

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Time in seconds between Friend Requests, if a previous Friend Request did not yield any acceptable Friend Offers.

Range:

- from 1 to 3600 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:

- 6 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_RSSI_FACTOR

RSSIFactor, used in Friend Offer Delay calculation

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The contribution of the RSSI, measured by the Friend node, used in Friend Offer Delay calculations. 0 = 1, 1 = 1.5, 2 = 2, 3 = 2.5. RSSIFactor, one of the parameters carried by Friend Request sent by Low Power node, which is used to calculate the Friend Offer Delay.

Range:

- from 0 to 3 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:

- 0 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_RECV_WIN_FACTOR

ReceiveWindowFactor, used in Friend Offer Delay calculation

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The contribution of the supported Receive Window used in Friend Offer Delay calculations. 0 = 1, 1 = 1.5, 2 = 2, 3 = 2.5. ReceiveWindowFactor, one of the parameters carried by Friend Request sent by Low Power node, which is used to calculate the Friend Offer Delay.

Range:

- from 0 to 3 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:

- 0 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_MIN_QUEUE_SIZE

Minimum size of the acceptable friend queue (MinQueueSizeLog)

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The MinQueueSizeLog field is defined as log_2(N), where N is the minimum number of maximum size Lower Transport PDUs that the Friend node can store in its Friend Queue. As an example, MinQueueSizeLog value 1 gives N = 2, and value 7 gives N = 128.

Range:
- from 1 to 7 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:
- 1 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_RECV_DELAY

Receive delay requested by the local node

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The ReceiveDelay is the time between the Low Power node sending a request and listening for a response. This delay allows the Friend node time to prepare the response. The value is in units of milliseconds.

Range:
- from 10 to 255 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:
- 100 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_POLL_TIMEOUT

The value of the PollTimeout timer

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

PollTimeout timer is used to measure time between two consecutive requests sent by a Low Power node. If no requests are received the Friend node before the PollTimeout timer expires, then the friendship is considered terminated. The value is in units of 100 milliseconds, so e.g. a value of 300 means 30 seconds. The smaller the value, the faster the Low Power node tries to get messages from corresponding Friend node and vice versa.

Range:
- from 10 to 244735 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:
- 300 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_INIT_POLL_TIMEOUT

The starting value of the PollTimeout timer

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The initial value of the PollTimeout timer when Friendship is to be established for the first time. After this, the timeout gradually grows toward the actual PollTimeout, doubling in value for each iteration. The value is in units of 100 milliseconds, so e.g. a value of 300 means 30 seconds.

Range:
- from 10 to if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:
- if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_SCAN_LATENCY

Latency for enabling scanning

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Latency (in milliseconds) is the time it takes to enable scanning. In practice, it means how much time in advance of the Receive Window, the request to enable scanning is made.

Range:
- from 0 to 50 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:
- 10 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_GROUPS

Number of groups the LPN can subscribe to

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Maximum number of groups to which the LPN can subscribe.

Range:
- from 0 to 16384 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

Default value:
- 8 if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LPN_SUB_ALL_NODES_ADDR

Automatically subscribe all nodes address

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Automatically subscribe all nodes address when friendship established.

Default value:
- No (disabled) if *CONFIG_BLE_MESH_LOW_POWER* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_FRIEND

Support for Friend feature

Found in: Component config > CONFIG_BLE_MESH

Enable this option to be able to act as a Friend Node.

CONFIG_BLE_MESH_FRIEND_RECV_WIN

Friend Receive Window

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND

Receive Window in milliseconds supported by the Friend node.

Range:
- from 1 to 255 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

Default value:
- 255 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_FRIEND_QUEUE_SIZE

Minimum number of buffers supported per Friend Queue

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND

Minimum number of buffers available to be stored for each local Friend Queue. This option decides the size of each buffer which can be used by a Friend node to store messages for each Low Power node.

Range:
- from 2 to 65536 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

Default value:
- 16 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_FRIEND_SUB_LIST_SIZE

Friend Subscription List Size

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND

Size of the Subscription List that can be supported by a Friend node for a Low Power node. And Low Power node can send Friend Subscription List Add or Friend Subscription List Remove messages to the Friend node to add or remove subscription addresses.

Range:
- from 0 to 1023 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

Default value:
- 3 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_FRIEND_LPN_COUNT

Number of supported LPN nodes

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND

Number of Low Power Nodes with which a Friend can have Friendship simultaneously. A Friend node can have friendship with multiple Low Power nodes at the same time, while a Low Power node can only establish friendship with only one Friend node at the same time.

Range:
- from 1 to 1000 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

Default value:
- 2 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_FRIEND_SEG_RX

Number of incomplete segment lists per LPN

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND

Number of incomplete segment lists tracked for each Friends' LPN. In other words, this determines from how many elements can segmented messages destined for the Friend queue be received simultaneously.

Range:
- from 1 to 1000 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

Default value:
- 1 if *CONFIG_BLE_MESH_FRIEND* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_NO_LOG

Disable BLE Mesh debug logs (minimize bin size)

Found in: Component config > CONFIG_BLE_MESH

Select this to save the BLE Mesh related rodata code size. Enabling this option will disable the output of BLE Mesh debug log.

Default value:

- No (disabled) if *CONFIG_BLE_MESH* && *CONFIG_BLE_MESH*

BLE Mesh STACK DEBUG LOG LEVEL Contains:

- *CONFIG_BLE_MESH_STACK_TRACE_LEVEL*

CONFIG_BLE_MESH_STACK_TRACE_LEVEL

BLE_MESH_STACK

Found in: Component config > *CONFIG_BLE_MESH* > *BLE Mesh STACK DEBUG LOG LEVEL*

Define BLE Mesh trace level for BLE Mesh stack.

Available options:

- NONE (CONFIG_BLE_MESH_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BLE_MESH_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BLE_MESH_TRACE_LEVEL_WARNING)
- INFO (CONFIG_BLE_MESH_TRACE_LEVEL_INFO)
- DEBUG (CONFIG_BLE_MESH_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BLE_MESH_TRACE_LEVEL_VERBOSE)

BLE Mesh NET BUF DEBUG LOG LEVEL Contains:

- *CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL*

CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL

BLE_MESH_NET_BUF

Found in: Component config > *CONFIG_BLE_MESH* > *BLE Mesh NET BUF DEBUG LOG LEVEL*

Define BLE Mesh trace level for BLE Mesh net buffer.

Available options:

- NONE (CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL_NONE)
- ERROR (CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL_ERROR)
- WARNING (CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL_WARNING)
- INFO (CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL_INFO)
- DEBUG (CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL_DEBUG)
- VERBOSE (CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL_VERBOSE)

CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT

Timeout(ms) for client message response

Found in: Component config > *CONFIG_BLE_MESH*

Timeout value used by the node to get response of the acknowledged message which is sent by the client model. This value indicates the maximum time that a client model waits for the response of the sent acknowledged messages. If a client model uses 0 as the timeout value when sending acknowledged messages, then the default value will be used which is four seconds.

Range:

- from 100 to 1200000 if *CONFIG_BLE_MESH*

Default value:

- 4000 if *CONFIG_BLE_MESH*

Support for BLE Mesh Foundation models Contains:

- *CONFIG_BLE_MESH_BRC_CLI*
- *CONFIG_BLE_MESH_BRC_SRV*
- *CONFIG_BLE_MESH_CFG_CLI*
- *CONFIG_BLE_MESH_DF_CLI*
- *CONFIG_BLE_MESH_DF_SRV*
- *CONFIG_BLE_MESH_HEALTH_CLI*
- *CONFIG_BLE_MESH_HEALTH_SRV*
- *CONFIG_BLE_MESH_LCD_CLI*
- *CONFIG_BLE_MESH_LCD_SRV*
- *CONFIG_BLE_MESH_PRB_CLI*
- *CONFIG_BLE_MESH_PRB_SRV*
- *CONFIG_BLE_MESH_ODP_CLI*
- *CONFIG_BLE_MESH_ODP_SRV*
- *CONFIG_BLE_MESH_AGG_CLI*
- *CONFIG_BLE_MESH_AGG_SRV*
- *CONFIG_BLE_MESH_RPR_CLI*
- *CONFIG_BLE_MESH_RPR_SRV*
- *CONFIG_BLE_MESH_SAR_CLI*
- *CONFIG_BLE_MESH_SAR_SRV*
- *CONFIG_BLE_MESH_SRPL_CLI*
- *CONFIG_BLE_MESH_SRPL_SRV*
- *CONFIG_BLE_MESH_COMP_DATA_1*
- *CONFIG_BLE_MESH_COMP_DATA_128*
- *CONFIG_BLE_MESH_MODELS_METADATA_0*

CONFIG_BLE_MESH_CFG_CLI

Configuration Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Configuration Client model.

CONFIG_BLE_MESH_HEALTH_CLI

Health Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Health Client model.

CONFIG_BLE_MESH_HEALTH_SRV

Health Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Health Server model.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_BRC_CLI

Bridge Configuration Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Bridge Configuration Client model.

CONFIG_BLE_MESH_BRC_SRV

Bridge Configuration Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Bridge Configuration Server model.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MAX_BRIDGING_TABLE_ENTRY_COUNT

Maximum number of Bridging Table entries

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models > CONFIG_BLE_MESH_BRC_SRV

Maximum number of Bridging Table entries that the Bridge Configuration Server can support.

Range:

- from 16 to 65535 if *CONFIG_BLE_MESH_BRC_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Default value:

- 16 if *CONFIG_BLE_MESH_BRC_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_BRIDGE_CRPL

Maximum capacity of bridge replay protection list

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models > CONFIG_BLE_MESH_BRC_SRV

This option specifies the maximum capacity of the bridge replay protection list. The bridge replay protection list is used to prevent a bridged subnet from replay attack, which will store the source address and sequence number of the received bridge messages.

Range:

- from 1 to 255 if *CONFIG_BLE_MESH_BRC_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Default value:

- 5 if *CONFIG_BLE_MESH_BRC_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PRB_CLI

Mesh Private Beacon Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Mesh Private Beacon Client model.

CONFIG_BLE_MESH_PRB_SRV

Mesh Private Beacon Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Mesh Private Beacon Server model.

CONFIG_BLE_MESH_ODP_CLI

On-Demand Private Proxy Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for On-Demand Private Proxy Client model.

CONFIG_BLE_MESH_ODP_SRV

On-Demand Private Proxy Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for On-Demand Private Proxy Server model.

CONFIG_BLE_MESH_SRPL_CLI

Solicitation PDU RPL Configuration Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Solicitation PDU RPL Configuration Client model.

CONFIG_BLE_MESH_SRPL_SRV

Solicitation PDU RPL Configuration Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Solicitation PDU RPL Configuration Server model. Note: This option depends on the functionality of receiving Solicitation PDU. If the device doesn't support receiving Solicitation PDU, then there is no need to enable this server model.

CONFIG_BLE_MESH_AGG_CLI

Opcodes Aggregator Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Opcodes Aggregator Client model.

CONFIG_BLE_MESH_AGG_SRV

Opcodes Aggregator Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Opcodes Aggregator Server model.

CONFIG_BLE_MESH_SAR_CLI

SAR Configuration Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for SAR Configuration Client model.

CONFIG_BLE_MESH_SAR_SRV

SAR Configuration Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for SAR Configuration Server model.

CONFIG_BLE_MESH_COMP_DATA_1

Support Composition Data Page 1

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Composition Data Page 1 contains information about the relationships among models. Each model either can be a root model or can extend other models.

CONFIG_BLE_MESH_COMP_DATA_128

Support Composition Data Page 128

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Composition Data Page 128 is used to indicate the structure of elements, features, and models of a node after the successful execution of the Node Address Refresh procedure or the Node Composition Refresh procedure, or after the execution of the Node Removal procedure followed by the provisioning process. Composition Data Page 128 shall be present if the node supports the Remote Provisioning Server model; otherwise it is optional.

CONFIG_BLE_MESH_MODELS_METADATA_0

Support Models Metadata Page 0

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

The Models Metadata state contains metadata of a node’ s models. The Models Metadata state is composed of a number of pages of information. Models Metadata Page 0 shall be present if the node supports the Large Composition Data Server model.

CONFIG_BLE_MESH_MODELS_METADATA_128

Support Models Metadata Page 128

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models > CONFIG_BLE_MESH_MODELS_METADATA_0

The Models Metadata state contains metadata of a node’ s models. The Models Metadata state is composed of a number of pages of information. Models Metadata Page 128 contains metadata for the node’ s models after the successful execution of the Node Address Refresh procedure or the Node Composition Refresh procedure, or after the execution of the Node Removal procedure followed by the provisioning process. Models Metadata Page 128 shall be present if the node supports the Remote Provisioning Server model and the node supports the Large Composition Data Server model.

CONFIG_BLE_MESH_LCD_CLI

Large Composition Data Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Large Composition Data Client model.

CONFIG_BLE_MESH_LCD_SRV

Large Composition Data Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Large Composition Data Server model.

CONFIG_BLE_MESH_RPR_CLI

Remote Provisioning Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Remote Provisioning Client model

CONFIG_BLE_MESH_RPR_CLI_PROV_SAME_TIME

Maximum number of PB-Remote running at the same time by Provisioner

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models > CONFIG_BLE_MESH_RPR_CLI

This option specifies how many devices can be provisioned at the same time using PB-REMOTE. For example, if the value is 2, it means a Provisioner can provision two unprovisioned devices with PB-REMOTE at the same time.

Range:

- from 1 to 5 if *CONFIG_BLE_MESH_RPR_CLI* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Default value:

- 2 if *CONFIG_BLE_MESH_RPR_CLI* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RPR_SRV

Remote Provisioning Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Remote Provisioning Server model

CONFIG_BLE_MESH_RPR_SRV_MAX_SCANNED_ITEMS

Maximum number of device information can be scanned

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models > CONFIG_BLE_MESH_RPR_SRV

This option specifies how many device information can a Remote Provisioning Server store each time while scanning.

Range:

- from 4 to 255 if *CONFIG_BLE_MESH_RPR_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Default value:

- 10 if *CONFIG_BLE_MESH_RPR_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_RPR_SRV_ACTIVE_SCAN

Support Active Scan for remote provisioning

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models > CONFIG_BLE_MESH_RPR_SRV

Enable this option to support Active Scan for remote provisioning.

CONFIG_BLE_MESH_RPR_SRV_MAX_EXT_SCAN

Maximum number of extended scan procedures

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models > CONFIG_BLE_MESH_RPR_SRV

This option specifies how many extended scan procedures can be started by the Remote Provisioning Server.

Range:

- from 1 to 10 if *CONFIG_BLE_MESH_RPR_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Default value:

- 1 if *CONFIG_BLE_MESH_RPR_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_DF_CLI

Directed Forwarding Configuration Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Directed Forwarding Configuration Client model.

CONFIG_BLE_MESH_DF_SRV

Directed Forwarding Configuration Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Directed Forwarding Configuration Server model.

CONFIG_BLE_MESH_MAX_DISC_TABLE_ENTRY_COUNT

Maximum number of discovery table entries in a given subnet

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models > CONFIG_BLE_MESH_DF_SRV

Maximum number of Discovery Table entries supported by the node in a given subnet.

Range:

- from 2 to 255 if *CONFIG_BLE_MESH_DF_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Default value:

- 2 if *CONFIG_BLE_MESH_DF_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MAX_FORWARD_TABLE_ENTRY_COUNT

Maximum number of forward table entries in a given subnet

Found in: *Component config* > *CONFIG_BLE_MESH* > *Support for BLE Mesh Foundation models* > *CONFIG_BLE_MESH_DF_SRV*

Maximum number of Forward Table entries supported by the node in a given subnet.

Range:

- from 2 to 64 if *CONFIG_BLE_MESH_DF_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Default value:

- 2 if *CONFIG_BLE_MESH_DF_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MAX_DEPS_NODES_PER_PATH

Maximum number of dependent nodes per path

Found in: *Component config* > *CONFIG_BLE_MESH* > *Support for BLE Mesh Foundation models* > *CONFIG_BLE_MESH_DF_SRV*

Maximum size of dependent nodes list supported by each forward table entry.

Range:

- from 2 to 64 if *CONFIG_BLE_MESH_DF_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Default value:

- 2 if *CONFIG_BLE_MESH_DF_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_PATH_MONITOR_TEST

Enable Path Monitoring test mode

Found in: *Component config* > *CONFIG_BLE_MESH* > *Support for BLE Mesh Foundation models* > *CONFIG_BLE_MESH_DF_SRV*

The option only removes the Path Use timer; all other behavior of the device is not changed. If Path Monitoring test mode is going to be used, this option should be enabled.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_DF_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SUPPORT_DIRECTED_PROXY

Enable Directed Proxy functionality

Found in: *Component config* > *CONFIG_BLE_MESH* > *Support for BLE Mesh Foundation models* > *CONFIG_BLE_MESH_DF_SRV*

Support Directed Proxy functionality.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH_GATT_PROXY_SERVER* && *CONFIG_BLE_MESH_DF_SRV* && *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

Support for BLE Mesh Client/Server models Contains:

- *CONFIG_BLE_MESH_MBT_CLI*
- *CONFIG_BLE_MESH_MBT_SRV*
- *CONFIG_BLE_MESH_GENERIC_BATTERY_CLI*
- *CONFIG_BLE_MESH_GENERIC_DEF_TRANS_TIME_CLI*
- *CONFIG_BLE_MESH_GENERIC_LEVEL_CLI*
- *CONFIG_BLE_MESH_GENERIC_LOCATION_CLI*
- *CONFIG_BLE_MESH_GENERIC_ONOFF_CLI*
- *CONFIG_BLE_MESH_GENERIC_POWER_LEVEL_CLI*
- *CONFIG_BLE_MESH_GENERIC_POWER_ONOFF_CLI*
- *CONFIG_BLE_MESH_GENERIC_PROPERTY_CLI*
- *CONFIG_BLE_MESH_GENERIC_SERVER*
- *CONFIG_BLE_MESH_LIGHT_CTL_CLI*
- *CONFIG_BLE_MESH_LIGHT_HSL_CLI*
- *CONFIG_BLE_MESH_LIGHT_LC_CLI*
- *CONFIG_BLE_MESH_LIGHT_LIGHTNESS_CLI*
- *CONFIG_BLE_MESH_LIGHT_XYL_CLI*
- *CONFIG_BLE_MESH_LIGHTING_SERVER*
- *CONFIG_BLE_MESH_SCENE_CLI*
- *CONFIG_BLE_MESH_SCHEDULER_CLI*
- *CONFIG_BLE_MESH_SENSOR_CLI*
- *CONFIG_BLE_MESH_SENSOR_SERVER*
- *CONFIG_BLE_MESH_TIME_SCENE_SERVER*
- *CONFIG_BLE_MESH_TIME_CLI*

CONFIG_BLE_MESH_GENERIC_ONOFF_CLI

Generic OnOff Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic OnOff Client model.

CONFIG_BLE_MESH_GENERIC_LEVEL_CLI

Generic Level Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Level Client model.

CONFIG_BLE_MESH_GENERIC_DEF_TRANS_TIME_CLI

Generic Default Transition Time Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Default Transition Time Client model.

CONFIG_BLE_MESH_GENERIC_POWER_ONOFF_CLI

Generic Power OnOff Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Power OnOff Client model.

CONFIG_BLE_MESH_GENERIC_POWER_LEVEL_CLI

Generic Power Level Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Power Level Client model.

CONFIG_BLE_MESH_GENERIC_BATTERY_CLI

Generic Battery Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Battery Client model.

CONFIG_BLE_MESH_GENERIC_LOCATION_CLI

Generic Location Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Location Client model.

CONFIG_BLE_MESH_GENERIC_PROPERTY_CLI

Generic Property Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Property Client model.

CONFIG_BLE_MESH_SENSOR_CLI

Sensor Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Sensor Client model.

CONFIG_BLE_MESH_TIME_CLI

Time Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Time Client model.

CONFIG_BLE_MESH_SCENE_CLI

Scene Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Scene Client model.

CONFIG_BLE_MESH_SCHEDULER_CLI

Scheduler Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Scheduler Client model.

CONFIG_BLE_MESH_LIGHT_LIGHTNESS_CLI

Light Lightness Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Light Lightness Client model.

CONFIG_BLE_MESH_LIGHT_CTL_CLI

Light CTL Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Light CTL Client model.

CONFIG_BLE_MESH_LIGHT_HSL_CLI

Light HSL Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Light HSL Client model.

CONFIG_BLE_MESH_LIGHT_XYL_CLI

Light XYL Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Light XYL Client model.

CONFIG_BLE_MESH_LIGHT_LC_CLI

Light LC Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Light LC Client model.

CONFIG_BLE_MESH_GENERIC_SERVER

Generic server models

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic server models.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SENSOR_SERVER

Sensor server models

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Sensor server models.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_TIME_SCENE_SERVER

Time and Scenes server models

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Time and Scenes server models.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_LIGHTING_SERVER

Lighting server models

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Lighting server models.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MBT_CLI

BLOB Transfer Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for BLOB Transfer Client model.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MAX_BLOB_RECEIVERS

Maximum number of simultaneous blob receivers

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models > CONFIG_BLE_MESH_MBT_CLI

Maximum number of BLOB Transfer Server models that can participating in the BLOB transfer with a BLOB Transfer Client model.

Range:

- from 1 to 255 if *CONFIG_BLE_MESH_MBT_CLI* && *CONFIG_BLE_MESH*

Default value:

- 2 if *CONFIG_BLE_MESH_MBT_CLI* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_MBT_SRV

BLOB Transfer Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for BLOB Transfer Server model.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_V11_SUPPORT* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_IV_UPDATE_TEST

Test the IV Update Procedure

Found in: Component config > CONFIG_BLE_MESH

This option removes the 96 hour limit of the IV Update Procedure and lets the state to be changed at any time. If IV Update test mode is going to be used, this option should be enabled.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

BLE Mesh specific test option Contains:

- *CONFIG_BLE_MESH_DEBUG*
- *CONFIG_BLE_MESH_SHELL*
- *CONFIG_BLE_MESH_BQB_TEST*
- *CONFIG_BLE_MESH_SELF_TEST*
- *CONFIG_BLE_MESH_TEST_AUTO_ENTER_NETWORK*
- *CONFIG_BLE_MESH_TEST_USE_WHITE_LIST*

CONFIG_BLE_MESH_SELF_TEST

Perform BLE Mesh self-tests

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

This option adds extra self-tests which are run every time BLE Mesh networking is initialized.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_BQB_TEST

Enable BLE Mesh specific internal test

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

This option is used to enable some internal functions for auto-pts test.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_TEST_AUTO_ENTER_NETWORK

Unprovisioned device enters mesh network automatically

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

With this option enabled, an unprovisioned device can automatically enters mesh network using a specific test function without the pro- visioning procedure. And on the Provisioner side, a test function needs to be invoked to add the node information into the mesh stack.

Default value:

- Yes (enabled) if *CONFIG_BLE_MESH_SELF_TEST* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_TEST_USE_WHITE_LIST

Use white list to filter mesh advertising packets

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

With this option enabled, users can use white list to filter mesh advertising packets while scanning.

Default value:

- No (disabled) if *CONFIG_BLE_MESH_SELF_TEST* && *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_SHELL

Enable BLE Mesh shell

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

Activate shell module that provides BLE Mesh commands to the console.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_DEBUG

Enable BLE Mesh debug logs

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

Enable debug logs for the BLE Mesh functionality.

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

CONFIG_BLE_MESH_DEBUG_NET

Network layer debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Network layer debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_TRANS

Transport layer debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Transport layer debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_BEACON

Beacon debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Beacon-related debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_CRYPTO

Crypto debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable cryptographic debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_PROV

Provisioning debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Provisioning debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_ACCESS

Access layer debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Access layer debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_MODEL

Foundation model debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Foundation Models debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_ADV

Advertising debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable advertising debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_LOW_POWER

Low Power debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Low Power debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_FRIEND

Friend debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Friend debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_PROXY

Proxy debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Proxy protocol debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_EXPERIMENTAL

Make BLE Mesh experimental features visible

Found in: Component config > CONFIG_BLE_MESH

Make BLE Mesh Experimental features visible. Experimental features list: - CONFIG_BLE_MESH_NOT_RELAY_REPLAY_MSG

Default value:

- No (disabled) if *CONFIG_BLE_MESH*

Console Library Contains:

- *CONFIG_CONSOLE_SORTED_HELP*

CONFIG_CONSOLE_SORTED_HELP

Enable sorted help

Found in: Component config > Console Library

Instead of listing the commands in the order of registration, the help command lists the available commands in sorted order, if this option is enabled.

Default value:

- No (disabled)

Driver Configurations Contains:

- *Legacy ADC Driver Configuration*
- *Legacy DAC Driver Configurations*
- *Legacy I2S Driver Configurations*
- *Legacy MCPWM Driver Configurations*
- *Legacy PCNT Driver Configurations*
- *Legacy RMT Driver Configurations*
- *Legacy SDM Driver Configurations*
- *Legacy Temperature Sensor Driver Configurations*
- *Legacy Timer Group Driver Configurations*
- *TWAI Configuration*

TWAI Configuration Contains:

- *CONFIG_TWAI_ISR_IN_IRAM*

CONFIG_TWAI_ISR_IN_IRAM

Place TWAI ISR function into IRAM

Found in: Component config > Driver Configurations > TWAI Configuration

Place the TWAI ISR in to IRAM. This will allow the ISR to avoid cache misses, and also be able to run whilst the cache is disabled (such as when writing to SPI Flash). Note that if this option is enabled: - Users should also set the ESP_INTR_FLAG_IRAM in the driver configuration structure when installing the driver (see docs for specifics). - Alert logging (i.e., setting of the TWAI_ALERT_AND_LOG flag) will have no effect.

Default value:

- No (disabled)

Legacy ADC Driver Configuration Contains:

- *CONFIG_ADC_DISABLE_DAC*
- *Legacy ADC Calibration Configuration*
- *CONFIG_ADC_SUPPRESS_DEPRECATE_WARN*

CONFIG_ADC_DISABLE_DAC

Disable DAC when ADC2 is used on GPIO 25 and 26

Found in: Component config > Driver Configurations > Legacy ADC Driver Configuration

If this is set, the ADC2 driver will disable the output of the DAC corresponding to the specified channel. This is the default value.

For testing, disable this option so that we can measure the output of DAC by internal ADC.

Default value:

- Yes (enabled) if SOC_DAC_SUPPORTED

CONFIG_ADC_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy ADC Driver Configuration

Whether to suppress the deprecation warnings when using legacy adc driver (driver/adc.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

Legacy ADC Calibration Configuration Contains:

- *CONFIG_ADC_CALI_SUPPRESS_DEPRECATE_WARN*

CONFIG_ADC_CALI_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy ADC Driver Configuration > Legacy ADC Calibration Configuration

Whether to suppress the deprecation warnings when using legacy adc calibration driver (esp_adc_cal.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

Legacy DAC Driver Configurations Contains:

- *CONFIG_DAC_SUPPRESS_DEPRECATE_WARN*

CONFIG_DAC_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy DAC Driver Configurations

Whether to suppress the deprecation warnings when using legacy dac driver (driver/dac.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled) if SOC_DAC_SUPPORTED

Legacy MCPWM Driver Configurations Contains:

- *CONFIG_MCPWM_SUPPRESS_DEPRECATE_WARN*

CONFIG_MCPWM_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy MCPWM Driver Configurations

Whether to suppress the deprecation warnings when using legacy MCPWM driver (driver/mcpwm.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

Legacy Timer Group Driver Configurations Contains:

- *CONFIG_GPTIMER_SUPPRESS_DEPRECATE_WARN*

CONFIG_GPTIMER_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy Timer Group Driver Configurations

Whether to suppress the deprecation warnings when using legacy timer group driver (driver/timer.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

Legacy RMT Driver Configurations Contains:

- *CONFIG_RMT_SUPPRESS_DEPRECATE_WARN*

CONFIG_RMT_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy RMT Driver Configurations

Whether to suppress the deprecation warnings when using legacy rmt driver (driver/rmt.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

Legacy I2S Driver Configurations Contains:

- *CONFIG_I2S_SUPPRESS_DEPRECATE_WARN*

CONFIG_I2S_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy I2S Driver Configurations

Whether to suppress the deprecation warnings when using legacy i2s driver (driver/i2s.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

Legacy PCNT Driver Configurations Contains:

- *CONFIG_PCNT_SUPPRESS_DEPRECATE_WARN*

CONFIG_PCNT_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy PCNT Driver Configurations

whether to suppress the deprecation warnings when using legacy PCNT driver (driver/pcnt.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

Legacy SDM Driver Configurations Contains:

- *CONFIG_SDM_SUPPRESS_DEPRECATE_WARN*

CONFIG_SDM_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy SDM Driver Configurations

whether to suppress the deprecation warnings when using legacy SDM driver (driver/sigmadelta.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

Legacy Temperature Sensor Driver Configurations Contains:

- *CONFIG_TEMP_SENSOR_SUPPRESS_DEPRECATE_WARN*

CONFIG_TEMP_SENSOR_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Legacy Temperature Sensor Driver Configurations

whether to suppress the deprecation warnings when using legacy temperature sensor driver (driver/temp_sensor.h). If you want to continue using the legacy driver, and don't want to see related deprecation warnings, you can enable this option.

Default value:

- No (disabled)

eFuse Bit Manager Contains:

- *CONFIG_EFUSE_VIRTUAL*
- *CONFIG_EFUSE_CUSTOM_TABLE*

CONFIG_EFUSE_CUSTOM_TABLE

Use custom eFuse table

Found in: Component config > eFuse Bit Manager

Allows to generate a structure for eFuse from the CSV file.

Default value:

- No (disabled)

CONFIG_EFUSE_CUSTOM_TABLE_FILENAME

Custom eFuse CSV file

Found in: Component config > eFuse Bit Manager > CONFIG_EFUSE_CUSTOM_TABLE

Name of the custom eFuse CSV filename. This path is evaluated relative to the project root directory.

Default value:

- ”main/esp_efuse_custom_table.csv” if *CONFIG_EFUSE_CUSTOM_TABLE*

CONFIG_EFUSE_VIRTUAL

Simulate eFuse operations in RAM

Found in: Component config > eFuse Bit Manager

If ”n” - No virtual mode. All eFuse operations are real and use eFuse registers. If ”y” - The virtual mode is enabled and all eFuse operations (read and write) are redirected to RAM instead of eFuse registers, all permanent changes (via eFuse) are disabled. Log output will state changes that would be applied, but they will not be.

If it is ”y”, then SECURE_FLASH_ENCRYPTION_MODE_RELEASE cannot be used. Because the EFUSE VIRT mode is for testing only.

During startup, the eFuses are copied into RAM. This mode is useful for fast tests.

Default value:

- No (disabled)

CONFIG_EFUSE_VIRTUAL_KEEP_IN_FLASH

Keep eFuses in flash

Found in: Component config > eFuse Bit Manager > CONFIG_EFUSE_VIRTUAL

In addition to the ”Simulate eFuse operations in RAM” option, this option just adds a feature to keep eFuses after reboots in flash memory. To use this mode the partition_table should have the *efuse* partition. partition.csv: ”efuse_em, data, efuse, , 0x2000,”

During startup, the eFuses are copied from flash or, in case if flash is empty, from real eFuse to RAM and then update flash. This mode is useful when need to keep changes after reboot (testing secure_boot and flash_encryption).

CONFIG_EFUSE_VIRTUAL_LOG_ALL_WRITES

Log all virtual writes

Found in: Component config > eFuse Bit Manager > CONFIG_EFUSE_VIRTUAL

If enabled, log efuse burns. This shows changes that would be made.

ESP-TLS Contains:

- *CONFIG_ESP_TLS_INSECURE*
- *CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK*
- *CONFIG_ESP_TLS_LIBRARY_CHOOSE*
- *CONFIG_ESP_TLS_CLIENT_SESSION_TICKETS*
- *CONFIG_ESP_DEBUG_WOLFSSL*
- *CONFIG_ESP_TLS_PSK_VERIFICATION*
- *CONFIG_ESP_TLS_SERVER_SESSION_TICKETS*
- *CONFIG_ESP_WOLFSSL_SMALL_CERT_VERIFY*
- *CONFIG_ESP_TLS_SERVER_MIN_AUTH_MODE_OPTIONAL*
- *CONFIG_ESP_TLS_USE_DS_PERIPHERAL*

CONFIG_ESP_TLS_LIBRARY_CHOOSE

Choose SSL/TLS library for ESP-TLS (See help for more Info)

Found in: Component config > ESP-TLS

The ESP-TLS APIs support multiple backend TLS libraries. Currently mbedTLS and WolfSSL are supported. Different TLS libraries may support different features and have different resource usage. Consult the ESP-TLS documentation in ESP-IDF Programming guide for more details.

Available options:

- mbedTLS (CONFIG_ESP_TLS_USING_MBEDTLS)
- wolfSSL (License info in wolfSSL directory README) (CONFIG_ESP_TLS_USING_WOLFSSL)

CONFIG_ESP_TLS_USE_DS_PERIPHERAL

Use Digital Signature (DS) Peripheral with ESP-TLS

Found in: Component config > ESP-TLS

Enable use of the Digital Signature Peripheral for ESP-TLS.The DS peripheral can only be used when it is appropriately configured for TLS. Consult the ESP-TLS documentation in ESP-IDF Programming Guide for more details.

Default value:

- Yes (enabled)

CONFIG_ESP_TLS_CLIENT_SESSION_TICKETS

Enable client session tickets

Found in: Component config > ESP-TLS

Enable session ticket support as specified in RFC5077.

CONFIG_ESP_TLS_SERVER_SESSION_TICKETS

Enable server session tickets

Found in: Component config > ESP-TLS

Enable session ticket support as specified in RFC5077

CONFIG_ESP_TLS_SERVER_SESSION_TICKET_TIMEOUT

Server session ticket timeout in seconds

Found in: Component config > ESP-TLS > CONFIG_ESP_TLS_SERVER_SESSION_TICKETS

Sets the session ticket timeout used in the tls server.

Default value:

- 86400 if *CONFIG_ESP_TLS_SERVER_SESSION_TICKETS*

CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK

Certificate selection hook

Found in: Component config > ESP-TLS

Ability to configure and use a certificate selection callback during server handshake, to select a certificate to present to the client based on the TLS extensions supplied in the client hello (alpn, sni, etc).

CONFIG_ESP_TLS_SERVER_MIN_AUTH_MODE_OPTIONAL

ESP-TLS Server: Set minimum Certificate Verification mode to Optional

Found in: Component config > ESP-TLS

When this option is enabled, the peer (here, the client) certificate is checked by the server, however the handshake continues even if verification failed. By default, the peer certificate is not checked and ignored by the server.

mbedtls_ssl_get_verify_result() can be called after the handshake is complete to retrieve status of verification.

CONFIG_ESP_TLS_PSK_VERIFICATION

Enable PSK verification

Found in: Component config > ESP-TLS

Enable support for pre shared key ciphers, supported for both mbedTLS as well as wolfSSL TLS library.

CONFIG_ESP_TLS_INSECURE

Allow potentially insecure options

Found in: Component config > ESP-TLS

You can enable some potentially insecure options. These options should only be used for testing pusposes. Only enable these options if you are very sure.

CONFIG_ESP_TLS_SKIP_SERVER_CERT_VERIFY

Skip server certificate verification by default (WARNING: ONLY FOR TESTING PURPOSE, READ HELP)

Found in: Component config > ESP-TLS > CONFIG_ESP_TLS_INSECURE

After enabling this option the esp-tls client will skip the server certificate verification by default. Note that this option will only modify the default behaviour of esp-tls client regarding server cert verification. The default behaviour should only be applicable when no other option regarding the server cert verification is opted in the esp-tls config (e.g. crt_bundle_attach, use_global_ca_store etc.). WARNING : Enabling this option comes with a potential risk of establishing a TLS connection with a server which has a fake identity, provided that the server certificate is not provided either through API or other mechanism like ca_store etc.

CONFIG_ESP_WOLFSSL_SMALL_CERT_VERIFY

Enable SMALL_CERT_VERIFY

Found in: Component config > ESP-TLS

Enables server verification with Intermediate CA cert, does not authenticate full chain of trust upto the root CA cert (After Enabling this option client only needs to have Intermediate CA certificate of the server to authenticate server, root CA cert is not necessary).

Default value:

- Yes (enabled) if *CONFIG_ESP_TLS_USING_WOLFSSL*

CONFIG_ESP_DEBUG_WOLFSSL

Enable debug logs for wolfSSL

Found in: Component config > ESP-TLS

Enable detailed debug prints for wolfSSL SSL library.

ADC and ADC Calibration

Contains:

- *ADC Calibration Configurations*
- *CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE*
- *CONFIG_ADC_DISABLE_DAC_OUTPUT*
- *CONFIG_ADC_ENABLE_DEBUG_LOG*
- *CONFIG_ADC_ONESHOT_CTRL_FUNC_IN_IRAM*

CONFIG_ADC_ONESHOT_CTRL_FUNC_IN_IRAM

Place ISR version ADC oneshot mode read function into IRAM

Found in: Component config > ADC and ADC Calibration

Place ISR version ADC oneshot mode read function into IRAM.

Default value:

- No (disabled)

CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE

ADC continuous mode driver ISR IRAM-Safe

Found in: Component config > ADC and ADC Calibration

Ensure the ADC continuous mode ISR is IRAM-Safe. When enabled, the ISR handler will be available when the cache is disabled.

Default value:

- No (disabled)

ADC Calibration Configurations

CONFIG_ADC_DISABLE_DAC_OUTPUT

Disable DAC when ADC2 is in use

Found in: Component config > ADC and ADC Calibration

By default, this is set. The ADC oneshot driver will disable the output of the corresponding DAC channels: ESP32: IO25 and IO26 ESP32S2: IO17 and IO18

Disable this option so as to measure the output of DAC by internal ADC, for test usage.

Default value:

- Yes (enabled) if SOC_DAC_SUPPORTED

CONFIG_ADC_ENABLE_DEBUG_LOG

Enable ADC debug log

Found in: Component config > ADC and ADC Calibration

whether to enable the debug log message for ADC driver. Note that this option only controls the ADC driver log, will not affect other drivers.

note: This cannot be used in the ADC legacy driver.

Default value:

- No (disabled)

Wireless Coexistence Contains:

- *CONFIG_ESP_COEX_EXTERNAL_COEXIST_ENABLE*
- *CONFIG_ESP_COEX_GPIO_DEBUG*
- *CONFIG_ESP_COEX_SW_COEXIST_ENABLE*
- *CONFIG_ESP_COEX_POWER_MANAGEMENT*

CONFIG_ESP_COEX_SW_COEXIST_ENABLE

Software controls WiFi/Bluetooth coexistence

Found in: Component config > Wireless Coexistence

If enabled, WiFi & Bluetooth coexistence is controlled by software rather than hardware. Recommended for heavy traffic scenarios. Both coexistence configuration options are automatically managed, no user intervention is required. If only Bluetooth is used, it is recommended to disable this option to reduce binary file size.

CONFIG_ESP_COEX_EXTERNAL_COEXIST_ENABLE

External Coexistence

Found in: Component config > Wireless Coexistence

If enabled, HW External coexistence arbitration is managed by GPIO pins. It can support three types of wired combinations so far which are 1-wired/2-wired/3-wired. User can select GPIO pins in application code with configure interfaces.

This function depends on BT-off because currently we do not support external coex and internal coex simultaneously.

CONFIG_ESP_COEX_POWER_MANAGEMENT

Support power management under coexistence

Found in: Component config > Wireless Coexistence

If enabled, coexist power management will be enabled.

CONFIG_ESP_COEX_GPIO_DEBUG

GPIO debugging for coexistence

Found in: Component config > Wireless Coexistence

Support coexistence GPIO debugging

CONFIG_ESP_COEX_GPIO_DEBUG_DIAG

Debugging Diagram

Found in: Component config > Wireless Coexistence > CONFIG_ESP_COEX_GPIO_DEBUG

Select type of debugging diagram

Available options:

- General (CONFIG_ESP_COEX_GPIO_DEBUG_DIAG_GENERAL)
- Wi-Fi (CONFIG_ESP_COEX_GPIO_DEBUG_DIAG_WIFI)

CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT

Max number of debugging GPIOs

Found in: Component config > Wireless Coexistence > CONFIG_ESP_COEX_GPIO_DEBUG

Range:

- from 0 to 12 if *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 12 if *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX0

Actual IO num for Debug IO ID0

Found in: Component config > Wireless Coexistence > CONFIG_ESP_COEX_GPIO_DEBUG

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 0 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 1 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 0 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX1

Actual IO num for Debug IO ID1

Found in: Component config > Wireless Coexistence > CONFIG_ESP_COEX_GPIO_DEBUG

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 1 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 2 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 1 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX2

Actual IO num for Debug IO ID2

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 2 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 3 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 2 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX3

Actual IO num for Debug IO ID3

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 3 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 4 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 3 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX4

Actual IO num for Debug IO ID4

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 4 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 5 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 4 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX5

Actual IO num for Debug IO ID5

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 5 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 6 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 5 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX6

Actual IO num for Debug IO ID6

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 6 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 7 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 6 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX7

Actual IO num for Debug IO ID7

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 7 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 8 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 7 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX8

Actual IO num for Debug IO ID8

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 8 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 9 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 8 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX9

Actual IO num for Debug IO ID9

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 9 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 10 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 9 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX10

Actual IO num for Debug IO ID10

Found in: Component config > *Wireless Coexistence* > *CONFIG_ESP_COEX_GPIO_DEBUG*

Range:

- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 10 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:

- 11 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 10 && *CONFIG_ESP_COEX_GPIO_DEBUG*

CONFIG_ESP_COEX_GPIO_DEBUG_IO_IDX11

Actual IO num for Debug IO ID11

Found in: Component config > Wireless Coexistence > CONFIG_ESP_COEX_GPIO_DEBUG

Range:
- from 0 to 54 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 11 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Default value:
- 12 if *CONFIG_ESP_COEX_GPIO_DEBUG_IO_COUNT* > 11 && *CONFIG_ESP_COEX_GPIO_DEBUG*

Common ESP-related Contains:

- *CONFIG_ESP_ERR_TO_NAME_LOOKUP*

CONFIG_ESP_ERR_TO_NAME_LOOKUP

Enable lookup of error code strings

Found in: Component config > Common ESP-related

Functions esp_err_to_name() and esp_err_to_name_r() return string representations of error codes from a pre-generated lookup table. This option can be used to turn off the use of the look-up table in order to save memory but this comes at the price of sacrificing distinguishable (meaningful) output string representations.

Default value:
- Yes (enabled)

ESP-Driver:Analog Comparator Configurations Contains:

- *CONFIG_ANA_CMPR_ISR_IRAM_SAFE*
- *CONFIG_ANA_CMPR_ENABLE_DEBUG_LOG*
- *CONFIG_ANA_CMPR_CTRL_FUNC_IN_IRAM*

CONFIG_ANA_CMPR_ISR_IRAM_SAFE

Analog comparator ISR IRAM-Safe

Found in: Component config > ESP-Driver:Analog Comparator Configurations

Ensure the Analog Comparator interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:
- No (disabled)

CONFIG_ANA_CMPR_CTRL_FUNC_IN_IRAM

Place Analog Comparator control functions into IRAM

Found in: Component config > ESP-Driver:Analog Comparator Configurations

Place Analog Comparator control functions (like ana_cmpr_set_internal_reference) into IRAM, so that these functions can be IRAM-safe and able to be called in an IRAM interrupt context. Enabling this option can improve driver performance as well.

Default value:
- No (disabled)

CONFIG_ANA_CMPR_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:Analog Comparator Configurations

whether to enable the debug log message for Analog Comparator driver. Note that, this option only controls the Analog Comparator driver log, won't affect other drivers.

Default value:

- No (disabled)

ESP-Driver:Camera Controller Configurations Contains:

- *CONFIG_CAM_CTLR_MIPI_CSI_ISR_IRAM_SAFE*
- *CONFIG_CAM_CTLR_DVP_CAM_ISR_IRAM_SAFE*
- *CONFIG_CAM_CTLR_ISP_DVP_ISR_IRAM_SAFE*

CONFIG_CAM_CTLR_MIPI_CSI_ISR_IRAM_SAFE

CSI ISR IRAM-Safe

Found in: Component config > ESP-Driver:Camera Controller Configurations

Ensure the CSI driver ISR is IRAM-Safe. When enabled, the ISR handler will be available when the cache is disabled.

Default value:

- No (disabled)

CONFIG_CAM_CTLR_ISP_DVP_ISR_IRAM_SAFE

ISP_DVP ISR IRAM-Safe

Found in: Component config > ESP-Driver:Camera Controller Configurations

Ensure the ISP_DVP driver ISR is IRAM-Safe. When enabled, the ISR handler will be available when the cache is disabled.

Default value:

- No (disabled)

CONFIG_CAM_CTLR_DVP_CAM_ISR_IRAM_SAFE

DVP ISR IRAM-Safe

Found in: Component config > ESP-Driver:Camera Controller Configurations

Ensure the DVP driver ISR is IRAM-Safe. When enabled, the ISR handler will be available when the cache is disabled.

Default value:

- No (disabled)

ESP-Driver:DAC Configurations Contains:

- *CONFIG_DAC_DMA_AUTO_16BIT_ALIGN*
- *CONFIG_DAC_ISR_IRAM_SAFE*
- *CONFIG_DAC_ENABLE_DEBUG_LOG*
- *CONFIG_DAC_CTRL_FUNC_IN_IRAM*

CONFIG_DAC_CTRL_FUNC_IN_IRAM

Place DAC control functions into IRAM

Found in: Component config > ESP-Driver:DAC Configurations

Place DAC control functions (e.g. 'dac_oneshot_output_voltage') into IRAM, so that this function can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

Default value:

- No (disabled) if SOC_DAC_SUPPORTED

CONFIG_DAC_ISR_IRAM_SAFE

DAC ISR IRAM-Safe

Found in: Component config > ESP-Driver:DAC Configurations

Ensure the DAC interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:

- No (disabled) if SOC_DAC_SUPPORTED

CONFIG_DAC_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:DAC Configurations

whether to enable the debug log message for DAC driver. Note that, this option only controls the DAC driver log, won't affect other drivers.

Default value:

- No (disabled) if SOC_DAC_SUPPORTED

CONFIG_DAC_DMA_AUTO_16BIT_ALIGN

Align the continuous data to 16 bit automatically

Found in: Component config > ESP-Driver:DAC Configurations

Whether to left shift the continuous data to align every bytes to 16 bits in the driver. On ESP32, although the DAC resolution is only 8 bits, the hardware requires 16 bits data in continuous mode. By enabling this option, the driver will left shift 8 bits for the input data automatically. Only disable this option when you decide to do this step by yourself. Note that the driver will allocate a new piece of memory to save the converted data.

Default value:

- Yes (enabled) if SOC_DAC_DMA_16BIT_ALIGN && SOC_DAC_SUPPORTED

ESP-Driver:GPIO Configurations

Contains:

- *CONFIG_GPIO_CTRL_FUNC_IN_IRAM*

CONFIG_GPIO_CTRL_FUNC_IN_IRAM

Place GPIO control functions into IRAM

Found in: Component config > ESP-Driver:GPIO Configurations

Place GPIO control functions (like intr_disable/set_level) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context.

Default value:

- No (disabled)

ESP-Driver:GPTimer Configurations Contains:

- *CONFIG_GPTIMER_ENABLE_DEBUG_LOG*
- *CONFIG_GPTIMER_ISR_IRAM_SAFE*
- *CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM*
- *CONFIG_GPTIMER_ISR_HANDLER_IN_IRAM*

CONFIG_GPTIMER_ISR_HANDLER_IN_IRAM

Place GPTimer ISR handler into IRAM

Found in: Component config > ESP-Driver:GPTimer Configurations

Place GPTimer ISR handler into IRAM for better performance and fewer cache misses.

Default value:

- Yes (enabled)

CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM

Place GPTimer control functions into IRAM

Found in: Component config > ESP-Driver:GPTimer Configurations

Place GPTimer control functions (like start/stop) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

Default value:

- No (disabled)

CONFIG_GPTIMER_ISR_IRAM_SAFE

GPTimer ISR IRAM-Safe

Found in: Component config > ESP-Driver:GPTimer Configurations

Ensure the GPTimer interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:

- No (disabled)

CONFIG_GPTIMER_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:GPTimer Configurations

whether to enable the debug log message for GPTimer driver. Note that, this option only controls the GPTimer driver log, won't affect other drivers.

Default value:

- No (disabled)

ESP-Driver:I2C Configurations Contains:

- *CONFIG_I2C_ENABLE_DEBUG_LOG*
- *CONFIG_I2C_ISR_IRAM_SAFE*

CONFIG_I2C_ISR_IRAM_SAFE

I2C ISR IRAM-Safe

Found in: Component config > ESP-Driver:I2C Configurations

Ensure the I2C interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write). note: This cannot be used in the I2C legacy driver.

Default value:

- No (disabled)

CONFIG_I2C_ENABLE_DEBUG_LOG

Enable I2C debug log

Found in: Component config > ESP-Driver:I2C Configurations

whether to enable the debug log message for I2C driver. Note that this option only controls the I2C driver log, will not affect other drivers.

note: This cannot be used in the I2C legacy driver.

Default value:

- No (disabled)

ESP-Driver:I2S Configurations

Contains:

- *CONFIG_I2S_ENABLE_DEBUG_LOG*
- *CONFIG_I2S_ISR_IRAM_SAFE*

CONFIG_I2S_ISR_IRAM_SAFE

I2S ISR IRAM-Safe

Found in: Component config > ESP-Driver:I2S Configurations

Ensure the I2S interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:

- No (disabled)

CONFIG_I2S_ENABLE_DEBUG_LOG

Enable I2S debug log

Found in: Component config > ESP-Driver:I2S Configurations

whether to enable the debug log message for I2S driver. Note that, this option only controls the I2S driver log, will not affect other drivers.

Default value:

- No (disabled)

ESP-Driver:ISP Configurations

Contains:

- *CONFIG_ISP_ISR_IRAM_SAFE*

CONFIG_ISP_ISR_IRAM_SAFE

ISP driver ISR IRAM-Safe

Found in: Component config > ESP-Driver:ISP Configurations

Ensure the ISP driver ISR is IRAM-Safe. When enabled, the ISR handler will be available when the cache is disabled.

Default value:

- No (disabled)

ESP-Driver:JPEG-Codec Configurations Contains:

- *CONFIG_JPEG_ENABLE_DEBUG_LOG*

CONFIG_JPEG_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:JPEG-Codec Configurations

whether to enable the debug log message for JPEG driver. Note that, this option only controls the JPEG driver log, won't affect other drivers. Please also note, enable this option will make jpeg codec process speed much slower.

Default value:

- No (disabled)

ESP-Driver:LEDC Configurations Contains:

- *CONFIG_LEDC_CTRL_FUNC_IN_IRAM*

CONFIG_LEDC_CTRL_FUNC_IN_IRAM

Place LEDC control functions into IRAM

Found in: Component config > ESP-Driver:LEDC Configurations

Place LEDC control functions (ledc_update_duty and ledc_stop) into IRAM, so that these functions can be IRAM-safe and able to be called in an IRAM context. Enabling this option can improve driver performance as well.

Default value:

- No (disabled)

ESP-Driver:MCPWM Configurations Contains:

- *CONFIG_MCPWM_ENABLE_DEBUG_LOG*
- *CONFIG_MCPWM_CTRL_FUNC_IN_IRAM*
- *CONFIG_MCPWM_ISR_IRAM_SAFE*

CONFIG_MCPWM_ISR_IRAM_SAFE

Place MCPWM ISR function into IRAM

Found in: Component config > ESP-Driver:MCPWM Configurations

This will ensure the MCPWM interrupt handle is IRAM-Safe, allow to avoid flash cache misses, and also be able to run whilst the cache is disabled. (e.g. SPI Flash write)

Default value:

- No (disabled)

CONFIG_MCPWM_CTRL_FUNC_IN_IRAM

Place MCPWM control functions into IRAM

Found in: Component config > ESP-Driver:MCPWM Configurations

Place MCPWM control functions (like set_compare_value) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

Default value:

- No (disabled)

CONFIG_MCPWM_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:MCPWM Configurations

whether to enable the debug log message for MCPWM driver. Note that, this option only controls the MCPWM driver log, won't affect other drivers.

Default value:

- No (disabled)

ESP-Driver:Parallel IO Configurations Contains:

- *CONFIG_PARLIO_ENABLE_DEBUG_LOG*
- *CONFIG_PARLIO_ISR_IRAM_SAFE*

CONFIG_PARLIO_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:Parallel IO Configurations

whether to enable the debug log message for parallel IO driver. Note that, this option only controls the parallel IO driver log, won't affect other drivers.

Default value:

- No (disabled)

CONFIG_PARLIO_ISR_IRAM_SAFE

Parallel IO ISR IRAM-Safe

Found in: Component config > ESP-Driver:Parallel IO Configurations

Ensure the Parallel IO interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:

- No (disabled)

ESP-Driver:PCNT Configurations Contains:

- *CONFIG_PCNT_ENABLE_DEBUG_LOG*
- *CONFIG_PCNT_ISR_IRAM_SAFE*
- *CONFIG_PCNT_CTRL_FUNC_IN_IRAM*

CONFIG_PCNT_CTRL_FUNC_IN_IRAM

Place PCNT control functions into IRAM

Found in: Component config > ESP-Driver:PCNT Configurations

Place PCNT control functions (like start/stop) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

Default value:

- No (disabled)

CONFIG_PCNT_ISR_IRAM_SAFE

PCNT ISR IRAM-Safe

Found in: Component config > ESP-Driver:PCNT Configurations

Ensure the PCNT interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:

- No (disabled)

CONFIG_PCNT_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:PCNT Configurations

whether to enable the debug log message for PCNT driver. Note that, this option only controls the PCNT driver log, won't affect other drivers.

Default value:

- No (disabled)

ESP-Driver:RMT Configurations Contains:

- *CONFIG_RMT_ENABLE_DEBUG_LOG*
- *CONFIG_RMT_RECV_FUNC_IN_IRAM*
- *CONFIG_RMT_ISR_IRAM_SAFE*

CONFIG_RMT_ISR_IRAM_SAFE

RMT ISR IRAM-Safe

Found in: Component config > ESP-Driver:RMT Configurations

Ensure the RMT interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:

- No (disabled)

CONFIG_RMT_RECV_FUNC_IN_IRAM

Place RMT receive function into IRAM

Found in: Component config > ESP-Driver:RMT Configurations

Place RMT receive function into IRAM, so that the receive function can be IRAM-safe and able to be called when the flash cache is disabled. Enabling this option can improve driver performance as well.

Default value:

- No (disabled)

CONFIG_RMT_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:RMT Configurations

whether to enable the debug log message for RMT driver. Note that, this option only controls the RMT driver log, won't affect other drivers.

Default value:

- No (disabled)

ESP-Driver:Sigma Delta Modulator Configurations Contains:

- *CONFIG_SDM_ENABLE_DEBUG_LOG*
- *CONFIG_SDM_CTRL_FUNC_IN_IRAM*

CONFIG_SDM_CTRL_FUNC_IN_IRAM

Place SDM control functions into IRAM

Found in: Component config > ESP-Driver:Sigma Delta Modulator Configurations

Place SDM control functions (like set_duty) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

Default value:

- No (disabled)

CONFIG_SDM_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:Sigma Delta Modulator Configurations

whether to enable the debug log message for SDM driver. Note that, this option only controls the SDM driver log, won't affect other drivers.

Default value:

- No (disabled)

ESP-Driver:SPI Configurations Contains:

- *CONFIG_SPI_MASTER_ISR_IN_IRAM*
- *CONFIG_SPI_SLAVE_ISR_IN_IRAM*
- *CONFIG_SPI_MASTER_IN_IRAM*
- *CONFIG_SPI_SLAVE_IN_IRAM*

CONFIG_SPI_MASTER_IN_IRAM

Place transmitting functions of SPI master into IRAM

Found in: Component config > ESP-Driver:SPI Configurations

Normally only the ISR of SPI master is placed in the IRAM, so that it can work without the flash when interrupt is triggered. For other functions, there's some possibility that the flash cache miss when running inside and out of SPI functions, which may increase the interval of SPI transactions. Enable this to put `queue\_trans`, `get\_trans\_result` and `transmit` functions into the IRAM to avoid possible cache miss.

This configuration won't be available if *CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH* is enabled.

During unit test, this is enabled to measure the ideal case of api.

CONFIG_SPI_MASTER_ISR_IN_IRAM

Place SPI master ISR function into IRAM

Found in: Component config > ESP-Driver:SPI Configurations

Place the SPI master ISR in to IRAM to avoid possible cache miss.

Enabling this configuration is possible only when HEAP_PLACE_FUNCTION_INTO_FLASH is disabled since the spi master uses can allocate transactions buffers into DMA memory section using the heap component API that ipso facto has to be placed in IRAM.

Also you can forbid the ISR being disabled during flash writing access, by add ESP_INTR_FLAG_IRAM when initializing the driver.

CONFIG_SPI_SLAVE_IN_IRAM

Place transmitting functions of SPI slave into IRAM

Found in: Component config > ESP-Driver:SPI Configurations

Normally only the ISR of SPI slave is placed in the IRAM, so that it can work without the flash when interrupt is triggered. For other functions, there's some possibility that the flash cache miss when running inside and out of SPI functions, which may increase the interval of SPI transactions. Enable this to put `queue\_trans`, `get\_trans\_result` and `transmit` functions into the IRAM to avoid possible cache miss.

Default value:

- No (disabled)

CONFIG_SPI_SLAVE_ISR_IN_IRAM

Place SPI slave ISR function into IRAM

Found in: Component config > ESP-Driver:SPI Configurations

Place the SPI slave ISR in to IRAM to avoid possible cache miss.

Also you can forbid the ISR being disabled during flash writing access, by add ESP_INTR_FLAG_IRAM when initializing the driver.

Default value:

- Yes (enabled)

ESP-Driver:Touch Sensor Configurations

Contains:

- *CONFIG_TOUCH_ENABLE_DEBUG_LOG*
- *CONFIG_TOUCH_CTRL_FUNC_IN_IRAM*
- *CONFIG_TOUCH_ISR_IRAM_SAFE*

CONFIG_TOUCH_CTRL_FUNC_IN_IRAM

Place touch sensor control functions into IRAM

Found in: Component config > ESP-Driver:Touch Sensor Configurations

Place touch sensor oneshot scanning and continuous scanning functions into IRAM, so that these function can be IRAM-safe and able to be called when the flash cache is disabled. Enabling this option can improve driver performance as well.

Default value:

- No (disabled)

CONFIG_TOUCH_ISR_IRAM_SAFE

Touch sensor ISR IRAM-Safe

Found in: Component config > ESP-Driver:Touch Sensor Configurations

Ensure the touch sensor interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:

- No (disabled)

CONFIG_TOUCH_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:Touch Sensor Configurations

Whether to enable the debug log message for touch driver. Note that, this option only controls the touch driver log, won't affect other drivers.

Default value:

- No (disabled)

ESP-Driver:Temperature Sensor Configurations Contains:

- *CONFIG_TEMP_SENSOR_ENABLE_DEBUG_LOG*
- *CONFIG_TEMP_SENSOR_ISR_IRAM_SAFE*

CONFIG_TEMP_SENSOR_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > ESP-Driver:Temperature Sensor Configurations

whether to enable the debug log message for temperature sensor driver. Note that, this option only controls the temperature sensor driver log, won't affect other drivers.

Default value:

- No (disabled)

CONFIG_TEMP_SENSOR_ISR_IRAM_SAFE

Temperature sensor ISR IRAM-Safe

Found in: Component config > ESP-Driver:Temperature Sensor Configurations

Ensure the Temperature Sensor interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:

- No (disabled)

ESP-Driver:UART Configurations Contains:

- *CONFIG_UART_ISR_IN_IRAM*

CONFIG_UART_ISR_IN_IRAM

Place UART ISR function into IRAM

Found in: Component config > ESP-Driver:UART Configurations

If this option is not selected, UART interrupt will be disabled for a long time and may cause data lost when doing spi flash operation.

ESP-Driver:USB Serial/JTAG Configuration Contains:

- *CONFIG_USJ_ENABLE_USB_SERIAL_JTAG*

CONFIG_USJ_ENABLE_USB_SERIAL_JTAG

Enable USB-Serial-JTAG Module

Found in: Component config > ESP-Driver:USB Serial/JTAG Configuration

The USB-Serial-JTAG module on ESP chips is turned on by default after power-on. If your application does not need it and not rely on it to be used as system console or use the built-in JTAG for debugging, you can disable this option, then the clock of this module will be disabled at startup, which will save some power consumption.

Default value:

- Yes (enabled)

CONFIG_USJ_NO_AUTO_LS_ON_CONNECTION

Don't enter the automatic light sleep when USB Serial/JTAG port is connected

Found in: Component config > ESP-Driver:USB Serial/JTAG Configuration > CONFIG_USJ_ENABLE_USB_SERIAL_JTAG

If enabled, the chip will constantly monitor the connection status of the USB Serial/JTAG port. As long as the USB Serial/JTAG is connected, a ESP_PM_NO_LIGHT_SLEEP power management lock will be acquired to prevent the system from entering light sleep. This option can be useful if serial monitoring is needed via USB Serial/JTAG while power management is enabled, as the USB Serial/JTAG cannot work under light sleep and after waking up from light sleep. Note. This option can only control the automatic Light-Sleep behavior. If esp_light_sleep_start() is called manually from the program, enabling this option will not prevent light sleep entry even if the USB Serial/JTAG is in use.

Ethernet Contains:

- *CONFIG_ETH_TRANSMIT_MUTEX*
- *CONFIG_ETH_USE_ESP32_EMAC*
- *CONFIG_ETH_USE_OPENETH*
- *CONFIG_ETH_USE_SPI_ETHERNET*

CONFIG_ETH_USE_ESP32_EMAC

Support ESP32 internal EMAC controller

Found in: Component config > Ethernet

ESP32 integrates a 10/100M Ethernet MAC controller.

Default value:

- Yes (enabled)

Contains:

- *CONFIG_ETH_DMA_RX_BUFFER_NUM*
- *CONFIG_ETH_DMA_TX_BUFFER_NUM*

- *CONFIG_ETH_IRAM_OPTIMIZATION*
- *CONFIG_ETH_SOFT_FLOW_CONTROL*
- *CONFIG_ETH_DMA_BUFFER_SIZE*
- *CONFIG_ETH_PHY_INTERFACE*

CONFIG_ETH_PHY_INTERFACE

PHY interface

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Select the communication interface between MAC and PHY chip.

Available options:

- Reduced Media Independent Interface (RMII) (CONFIG_ETH_PHY_INTERFACE_RMII)

CONFIG_ETH_DMA_BUFFER_SIZE

Ethernet DMA buffer size (Byte)

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Set the size of each buffer used by Ethernet MAC DMA. !! Important !! Make sure it is 64B aligned for ESP32P4!

Range:
- from 256 to 1600

Default value:
- 512

CONFIG_ETH_DMA_RX_BUFFER_NUM

Amount of Ethernet DMA Rx buffers

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Number of DMA receive buffers. Each buffer's size is ETH_DMA_BUFFER_SIZE. Larger number of buffers could increase throughput somehow.

Range:
- from 3 to 30

Default value:
- 20

CONFIG_ETH_DMA_TX_BUFFER_NUM

Amount of Ethernet DMA Tx buffers

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Number of DMA transmit buffers. Each buffer's size is ETH_DMA_BUFFER_SIZE. Larger number of buffers could increase throughput somehow.

Range:
- from 3 to 30

Default value:
- 10

CONFIG_ETH_SOFT_FLOW_CONTROL

Enable software flow control

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Ethernet MAC engine on ESP32 doesn't feature a flow control logic. The MAC driver can perform a software flow control if you enable this option. Note that, if the RX buffer number is small, enabling software flow control will cause obvious performance loss.

Default value:

- No (disabled)

CONFIG_ETH_IRAM_OPTIMIZATION

Enable IRAM optimization

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

If enabled, functions related to RX/TX are placed into IRAM. It can improve Ethernet throughput. If disabled, all functions are placed into FLASH.

Default value:

- No (disabled)

CONFIG_ETH_USE_SPI_ETHERNET

Support SPI to Ethernet Module

Found in: Component config > Ethernet

ESP-IDF can also support some SPI-Ethernet modules.

Default value:

- Yes (enabled)

Contains:

- *CONFIG_ETH_SPI_ETHERNET_DM9051*
- *CONFIG_ETH_SPI_ETHERNET_KSZ8851SNL*
- *CONFIG_ETH_SPI_ETHERNET_W5500*

CONFIG_ETH_SPI_ETHERNET_DM9051

Use DM9051

Found in: Component config > Ethernet > CONFIG_ETH_USE_SPI_ETHERNET

DM9051 is a fast Ethernet controller with an SPI interface. It's also integrated with a 10/100M PHY and MAC. Select this to enable DM9051 driver.

CONFIG_ETH_SPI_ETHERNET_W5500

Use W5500 (MAC RAW)

Found in: Component config > Ethernet > CONFIG_ETH_USE_SPI_ETHERNET

W5500 is a HW TCP/IP embedded Ethernet controller. TCP/IP stack, 10/100 Ethernet MAC and PHY are embedded in a single chip. However the driver in ESP-IDF only enables the RAW MAC mode, making it compatible with the software TCP/IP stack. Say yes to enable W5500 driver.

CONFIG_ETH_SPI_ETHERNET_KSZ8851SNL

Use KSZ8851SNL

Found in: Component config > Ethernet > CONFIG_ETH_USE_SPI_ETHERNET

The KSZ8851SNL is a single-chip Fast Ethernet controller consisting of a 10/100 physical layer transceiver (PHY), a MAC, and a Serial Peripheral Interface (SPI). Select this to enable KSZ8851SNL driver.

CONFIG_ETH_USE_OPENETH

Support OpenCores Ethernet MAC (for use with QEMU)

Found in: Component config > Ethernet

OpenCores Ethernet MAC driver can be used when an ESP-IDF application is executed in QEMU. This driver is not supported when running on a real chip.

Default value:

- No (disabled)

Contains:

- *CONFIG_ETH_OPENETH_DMA_RX_BUFFER_NUM*
- *CONFIG_ETH_OPENETH_DMA_TX_BUFFER_NUM*

CONFIG_ETH_OPENETH_DMA_RX_BUFFER_NUM

Number of Ethernet DMA Rx buffers

Found in: Component config > Ethernet > CONFIG_ETH_USE_OPENETH

Number of DMA receive buffers, each buffer is 1600 bytes.

Range:

- from 1 to 64 if *CONFIG_ETH_USE_OPENETH*

Default value:

- 4 if *CONFIG_ETH_USE_OPENETH*

CONFIG_ETH_OPENETH_DMA_TX_BUFFER_NUM

Number of Ethernet DMA Tx buffers

Found in: Component config > Ethernet > CONFIG_ETH_USE_OPENETH

Number of DMA transmit buffers, each buffer is 1600 bytes.

Range:

- from 1 to 64 if *CONFIG_ETH_USE_OPENETH*

Default value:

- 1 if *CONFIG_ETH_USE_OPENETH*

CONFIG_ETH_TRANSMIT_MUTEX

Enable Transmit Mutex

Found in: Component config > Ethernet

Prevents multiple accesses when Ethernet interface is used as shared resource and multiple functionalities might try to access it at a time.

Default value:

- No (disabled)

Event Loop Library Contains:

- *CONFIG_ESP_EVENT_LOOP_PROFILING*
- *CONFIG_ESP_EVENT_POST_FROM_ISR*

CONFIG_ESP_EVENT_LOOP_PROFILING

Enable event loop profiling

Found in: Component config > Event Loop Library

Enables collections of statistics in the event loop library such as the number of events posted to/recieved by an event loop, number of callbacks involved, number of events dropped to to a full event loop queue, run time of event handlers, and number of times/run time of each event handler.

Default value:

- No (disabled)

CONFIG_ESP_EVENT_POST_FROM_ISR

Support posting events from ISRs

Found in: Component config > Event Loop Library

Enable posting events from interrupt handlers.

Default value:

- Yes (enabled)

CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR

Support posting events from ISRs placed in IRAM

Found in: Component config > Event Loop Library > CONFIG_ESP_EVENT_POST_FROM_ISR

Enable posting events from interrupt handlers placed in IRAM. Enabling this option places API functions esp_event_post and esp_event_post_to in IRAM.

Default value:

- Yes (enabled)

GDB Stub Contains:

- *CONFIG_ESP_GDBSTUB_SUPPORT_TASKS*
- *CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME*

CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME

GDBStub at runtime

Found in: Component config > GDB Stub

Enable builtin GDBStub. This allows to debug the target device using serial port: - Run 'idf.py monitor'. - Wait for the device to initialize. - Press Ctrl+C to interrupt the execution and enter GDB attached to your device for debugging. NOTE: all UART input will be handled by GDBStub.

CONFIG_ESP_GDBSTUB_SUPPORT_TASKS

Enable listing FreeRTOS tasks through GDB Stub

Found in: Component config > GDB Stub

If enabled, GDBStub can supply the list of FreeRTOS tasks to GDB. Thread list can be queried from GDB using 'info threads' command. Note that if GDB task lists were corrupted, this feature may not work. If GDBStub fails, try disabling this feature.

Default value:

- Yes (enabled)

CONFIG_ESP_GDBSTUB_MAX_TASKS

Maximum number of tasks supported by GDB Stub

Found in: Component config > GDB Stub > CONFIG_ESP_GDBSTUB_SUPPORT_TASKS

Set the number of tasks which GDB Stub will support.

Default value:

- 32

ESP HTTP client

Contains:

- *CONFIG_ESP_HTTP_CLIENT_ENABLE_CUSTOM_TRANSPORT*
- *CONFIG_ESP_HTTP_CLIENT_ENABLE_BASIC_AUTH*
- *CONFIG_ESP_HTTP_CLIENT_ENABLE_DIGEST_AUTH*
- *CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS*

CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS

Enable https

Found in: Component config > ESP HTTP client

This option will enable https protocol by linking esp-tls library and initializing SSL transport

Default value:

- Yes (enabled)

CONFIG_ESP_HTTP_CLIENT_ENABLE_BASIC_AUTH

Enable HTTP Basic Authentication

Found in: Component config > ESP HTTP client

This option will enable HTTP Basic Authentication. It is disabled by default as Basic auth uses unencrypted encoding, so it introduces a vulnerability when not using TLS

Default value:

- No (disabled)

CONFIG_ESP_HTTP_CLIENT_ENABLE_DIGEST_AUTH

Enable HTTP Digest Authentication

Found in: Component config > ESP HTTP client

This option will enable HTTP Digest Authentication. It is enabled by default, but use of this configuration is not recommended as the password can be derived from the exchange, so it introduces a vulnerability when not using TLS

Default value:

- No (disabled)

CONFIG_ESP_HTTP_CLIENT_ENABLE_CUSTOM_TRANSPORT

Enable custom transport

Found in: Component config > ESP HTTP client

This option will enable injection of a custom tcp_transport handle, so the http operation will be performed on top of the user defined transport abstraction (if configured)

Default value:

- No (disabled)

HTTP Server Contains:

- *CONFIG_HTTPD_QUEUE_WORK_BLOCKING*
- *CONFIG_HTTPD_PURGE_BUF_LEN*
- *CONFIG_HTTPD_LOG_PURGE_DATA*
- *CONFIG_HTTPD_MAX_REQ_HDR_LEN*
- *CONFIG_HTTPD_MAX_URI_LEN*
- *CONFIG_HTTPD_ERR_RESP_NO_DELAY*
- *CONFIG_HTTPD_WS_SUPPORT*

CONFIG_HTTPD_MAX_REQ_HDR_LEN

Max HTTP Request Header Length

Found in: Component config > HTTP Server

This sets the maximum supported size of headers section in HTTP request packet to be processed by the server

Default value:

- 512

CONFIG_HTTPD_MAX_URI_LEN

Max HTTP URI Length

Found in: Component config > HTTP Server

This sets the maximum supported size of HTTP request URI to be processed by the server

Default value:

- 512

CONFIG_HTTPD_ERR_RESP_NO_DELAY

Use TCP_NODELAY socket option when sending HTTP error responses

Found in: Component config > HTTP Server

Using TCP_NODEALY socket option ensures that HTTP error response reaches the client before the underlying socket is closed. Please note that turning this off may cause multiple test failures

Default value:

- Yes (enabled)

CONFIG_HTTPD_PURGE_BUF_LEN

Length of temporary buffer for purging data

Found in: Component config > HTTP Server

This sets the size of the temporary buffer used to receive and discard any remaining data that is received from the HTTP client in the request, but not processed as part of the server HTTP request handler.

If the remaining data is larger than the available buffer size, the buffer will be filled in multiple iterations. The buffer should be small enough to fit on the stack, but large enough to avoid excessive iterations.

Default value:

- 32

CONFIG_HTTPD_LOG_PURGE_DATA

Log purged content data at Debug level

Found in: Component config > HTTP Server

Enabling this will log discarded binary HTTP request data at Debug level. For large content data this may not be desirable as it will clutter the log.

Default value:

- No (disabled)

CONFIG_HTTPD_WS_SUPPORT

WebSocket server support

Found in: Component config > HTTP Server

This sets the WebSocket server support.

Default value:

- No (disabled)

CONFIG_HTTPD_QUEUE_WORK_BLOCKING

httpd_queue_work as blocking API

Found in: Component config > HTTP Server

This makes httpd_queue_work() API to wait until a message space is available on UDP control socket. It internally uses a counting semaphore with count set to *LWIP_UDP_RECVMBOX_SIZE* to achieve this. This config will slightly change API behavior to block until message gets delivered on control socket.

ESP HTTPS OTA

Contains:

- *CONFIG_ESP_HTTPS_OTA_ALLOW_HTTP*
- *CONFIG_ESP_HTTPS_OTA_DECRYPT_CB*

CONFIG_ESP_HTTPS_OTA_DECRYPT_CB

Provide decryption callback

Found in: Component config > ESP HTTPS OTA

Exposes an additional callback whereby firmware data could be decrypted before being processed by OTA update component. This can help to integrate external encryption related format and removal of such encapsulation layer from firmware image.

Default value:

- No (disabled)

CONFIG_ESP_HTTPS_OTA_ALLOW_HTTP

Allow HTTP for OTA (WARNING: ONLY FOR TESTING PURPOSE, READ HELP)

Found in: Component config > ESP HTTPS OTA

It is highly recommended to keep HTTPS (along with server certificate validation) enabled. Enabling this option comes with potential risk of: - Non-encrypted communication channel with server - Accepting firmware upgrade image from server with fake identity

Default value:

- No (disabled)

ESP HTTPS server Contains:

- *CONFIG_ESP_HTTPS_SERVER_ENABLE*

CONFIG_ESP_HTTPS_SERVER_ENABLE

Enable ESP_HTTPS_SERVER component

Found in: Component config > ESP HTTPS server

Enable ESP HTTPS server component

Hardware Settings Contains:

- *2D-DMA Configurations*
- *Chip revision*
- *Crypto DPA Protection*
- *DCDC Regulator Configurations*
- *DW_GDMA Configurations*
- *ETM Configuration*
- *GDMA Configurations*
- *LDO Regulator Configurations*
- *MAC Config*
- *Main XTAL Config*
- *Peripheral Control*
- *RTC Clock Config*
- *Sleep Config*

Chip revision Contains:

- *CONFIG_ESP_REV_NEW_CHIP_TEST*
- *CONFIG_ESP_EFUSE_BLOCK_REV_MIN_FULL*
- *CONFIG_ESP32P4_REV_MIN*

CONFIG_ESP32P4_REV_MIN

Minimum Supported ESP32-P4 Revision

Found in: Component config > Hardware Settings > Chip revision

Required minimum chip revision. ESP-IDF will check for it and reject to boot if the chip revision fails the check. This ensures the chip used will have some modifications (features, or bugfixes).

The complied binary will only support chips above this revision, this will also help to reduce binary size.

Available options:

- Rev v0.0 (CONFIG_ESP32P4_REV_MIN_0)

- Rev v0.1 (CONFIG_ESP32P4_REV_MIN_1)
- Rev v1.0 (CONFIG_ESP32P4_REV_MIN_100)

CONFIG_ESP_EFUSE_BLOCK_REV_MIN_FULL

Minimum Supported ESP32-P4 eFuse Block Revision

Found in: Component config > Hardware Settings > Chip revision

Required minimum eFuse Block revision. ESP-IDF will check it at the 2nd bootloader stage whether the current image can work correctly for this eFuse Block revision. So that to avoid running an incompatible image on a SoC that contains breaking change in the eFuse Block. If you want to update this value to run the image that not compatible with the current eFuse Block revision, please contact to Espressif's business team for details: https://www.espressif.com.cn/en/contact-us/sales-questions

Default value:

- 0

CONFIG_ESP_REV_NEW_CHIP_TEST

Internal test mode

Found in: Component config > Hardware Settings > Chip revision

For internal chip testing, a small number of new versions chips didn't update the version field in eFuse, you can enable this option to force the software recognize the chip version based on the rev selected in menuconfig.

Default value:

- No (disabled)

MAC Config

Contains:

- *CONFIG_ESP_MAC_USE_CUSTOM_MAC_AS_BASE_MAC*
- *CONFIG_ESP32P4_UNIVERSAL_MAC_ADDRESSES*

CONFIG_ESP32P4_UNIVERSAL_MAC_ADDRESSES

Number of universally administered (by IEEE) MAC address

Found in: Component config > Hardware Settings > MAC Config

Configure the number of universally administered (by IEEE) MAC addresses.

During initialization, MAC addresses for each network interface are generated or derived from a single base MAC address.

If the number of universal MAC addresses is one, only Ethernet interface receives a universally administered MAC address. It's generated by adding 0 to the base MAC address.

When using the default (Espressif-assigned) base MAC address, either setting can be used. When using a custom universal MAC address range, the correct setting will depend on the allocation of MAC addresses in this range.

Available options:

- One (CONFIG_ESP32P4_UNIVERSAL_MAC_ADDRESSES_ONE)

CONFIG_ESP_MAC_USE_CUSTOM_MAC_AS_BASE_MAC

Enable using custom mac as base mac

Found in: Component config > Hardware Settings > MAC Config

When this configuration is enabled, the user can invoke *esp_read_mac* to obtain the desired type of MAC using a custom MAC as the base MAC.

Default value:

- No (disabled)

Sleep Config Contains:

- *CONFIG_ESP_SLEEP_GPIO_ENABLE_INTERNAL_RESISTORS*
- *CONFIG_ESP_SLEEP_CACHE_SAFE_ASSERTION*
- *CONFIG_ESP_SLEEP_EVENT_CALLBACKS*
- *CONFIG_ESP_SLEEP_DEBUG*
- *CONFIG_ESP_SLEEP_WAIT_FLASH_READY_EXTRA_DELAY*
- *CONFIG_ESP_SLEEP_GPIO_RESET_WORKAROUND*
- *CONFIG_ESP_SLEEP_POWER_DOWN_FLASH*
- *CONFIG_ESP_SLEEP_MSPI_NEED_ALL_IO_PU*
- *CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND*
- *CONFIG_ESP_SLEEP_PSRAM_LEAKAGE_WORKAROUND*

CONFIG_ESP_SLEEP_POWER_DOWN_FLASH

Power down flash in light sleep when there is no SPIRAM or SPIRAM has independent power supply

Found in: Component config > Hardware Settings > Sleep Config

If enabled, chip will try to power down flash as part of esp_light_sleep_start(), which costs more time when chip wakes up. Can only be enabled if there is no SPIRAM configured.

This option will power down flash under a strict but relatively safe condition. Also, it is possible to power down flash under a relaxed condition by using esp_sleep_pd_config() to set ESP_PD_DOMAIN_VDDSDIO to ESP_PD_OPTION_OFF. It should be noted that there is a risk in powering down flash, you can refer *ESP-IDF Programming Guide/API Reference/System API/Sleep Modes/Power-down of Flash* for more details.

CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND

Pull-up Flash CS pin in light sleep

Found in: Component config > Hardware Settings > Sleep Config

All IOs will be set to isolate(floating) state by default during sleep. Since the power supply of SPI Flash is not lost during lightsleep, if its CS pin is recognized as low level(selected state) in the floating state, there will be a large current leakage, and the data in Flash may be corrupted by random signals on other SPI pins. Select this option will set the CS pin of Flash to PULL-UP state during sleep, but this will increase the sleep current about 10 uA. If you are developing with esp32xx modules, you must select this option, but if you are developing with chips, you can also pull up the CS pin of SPI Flash in the external circuit to save power consumption caused by internal pull-up during sleep. (!!! Don't deselect this option if you don't have external SPI Flash CS pin pullups.)

CONFIG_ESP_SLEEP_PSRAM_LEAKAGE_WORKAROUND

Pull-up PSRAM CS pin in light sleep

Found in: Component config > Hardware Settings > Sleep Config

All IOs will be set to isolate(floating) state by default during sleep. Since the power supply of PSRAM is not lost during lightsleep, if its CS pin is recognized as low level(selected state) in the floating state,

there will be a large current leakage, and the data in PSRAM may be corrupted by random signals on other SPI pins. Select this option will set the CS pin of PSRAM to PULL-UP state during sleep, but this will increase the sleep current about 10 uA. If you are developing with esp32xx modules, you must select this option, but if you are developing with chips, you can also pull up the CS pin of PSRAM in the external circuit to save power consumption caused by internal pull-up during sleep. (!!! Don't deselect this option if you don't have external PSRAM CS pin pullups.)

Default value:

- Yes (enabled) if *CONFIG_SPIRAM*

CONFIG_ESP_SLEEP_MSPI_NEED_ALL_IO_PU

Pull-up all SPI pins in light sleep

Found in: Component config > Hardware Settings > Sleep Config

To reduce leakage current, some types of SPI Flash/RAM only need to pull up the CS pin during light sleep. But there are also some kinds of SPI Flash/RAM that need to pull up all pins. It depends on the SPI Flash/RAM chip used.

CONFIG_ESP_SLEEP_GPIO_RESET_WORKAROUND

light sleep GPIO reset workaround

Found in: Component config > Hardware Settings > Sleep Config

esp32c2, esp32c3, esp32s3, esp32c6 and esp32h2 will reset at wake-up if GPIO is received a small electrostatic pulse during light sleep, with specific condition

- GPIO needs to be configured as input-mode only
- The pin receives a small electrostatic pulse, and reset occurs when the pulse voltage is higher than 6 V

For GPIO set to input mode only, it is not a good practice to leave it open/floating, The hardware design needs to controlled it with determined supply or ground voltage is necessary.

This option provides a software workaround for this issue. Configure to isolate all GPIO pins in sleep state.

CONFIG_ESP_SLEEP_WAIT_FLASH_READY_EXTRA_DELAY

Extra delay (in us) after flash powerdown sleep wakeup to wait flash ready

Found in: Component config > Hardware Settings > Sleep Config

When the chip exits sleep, the CPU and the flash chip are powered on at the same time. CPU will run rom code (deepsleep) or ram code (lightsleep) first, and then load or execute code from flash.

Some flash chips need sufficient time to pass between power on and first read operation. By default, without any extra delay, this time is approximately 900us, although some flash chip types need more than that.

(!!! Please adjust this value according to the Data Sheet of SPI Flash used in your project.) In Flash Data Sheet, the parameters that define the Flash ready timing after power-up (minimum time from Vcc(min) to CS activeare) usually named tVSL in ELECTRICAL CHARACTERISTICS chapter, and the configuration value here should be: ESP_SLEEP_WAIT_FLASH_READY_EXTRA_DELAY = tVSL - 900

For esp32 and esp32s3, the default extra delay is set to 2000us. When optimizing startup time for applications which require it, this value may be reduced.

If you are seeing "flash read err, 1000" message printed to the console after deep sleep reset on esp32, or triggered RTC_WDT/LP_WDT after lightsleep wakeup, try increasing this value. (For esp32, the delay will be executed in both deep sleep and light sleep wake up flow. For chips after esp32, the delay

will be executed only in light sleep flow, the delay controlled by the EFUSE_FLASH_TPUW in ROM will be executed in deepsleep wake up flow.)

Range:
- from 0 to 5000

Default value:
- 0

CONFIG_ESP_SLEEP_CACHE_SAFE_ASSERTION

Check the cache safety of the sleep wakeup code in sleep process

Found in: Component config > Hardware Settings > Sleep Config

Enabling it will check the cache safety of the code before the flash power is ready after light sleep wakeup, and check PM_SLP_IRAM_OPT related code cache safety. This option is only for code quality inspection. Enabling it will increase the time overhead of entering and exiting sleep. It is not recommended to enable it in the release version.

Default value:
- No (disabled)

CONFIG_ESP_SLEEP_DEBUG

esp sleep debug

Found in: Component config > Hardware Settings > Sleep Config

Enable esp sleep debug.

Default value:
- No (disabled)

CONFIG_ESP_SLEEP_GPIO_ENABLE_INTERNAL_RESISTORS

Allow to enable internal pull-up/downs for the Deep-Sleep wakeup IOs

Found in: Component config > Hardware Settings > Sleep Config

When using rtc gpio wakeup source during deepsleep without external pull-up/downs, you may want to make use of the internal ones.

Default value:
- Yes (enabled)

CONFIG_ESP_SLEEP_EVENT_CALLBACKS

Enable registration of sleep event callbacks

Found in: Component config > Hardware Settings > Sleep Config

If enabled, it allows user to register sleep event callbacks. It is primarily designed for internal developers and customers can use PM_LIGHT_SLEEP_CALLBACKS as an alternative.

NOTE: These callbacks are executed from the IDLE task context hence you cannot have any blocking calls in your callbacks.

NOTE: Enabling these callbacks may change sleep duration calculations based on time spent in callback and hence it is highly recommended to keep them as short as possible.

Default value:
- No (disabled) if *CONFIG_FREERTOS_USE_TICKLESS_IDLE*

RTC Clock Config Contains:

- *CONFIG_RTC_CLK_CAL_CYCLES*
- *CONFIG_RTC_CLK_SRC*

CONFIG_RTC_CLK_SRC

RTC clock source

Found in: Component config > Hardware Settings > RTC Clock Config

Choose which clock is used as RTC clock source.

Available options:

- Internal 150 kHz RC oscillator (CONFIG_RTC_CLK_SRC_INT_RC)
- External 32kHz crystal (CONFIG_RTC_CLK_SRC_EXT_CRYS)
- Internal 32kHz RC oscillator (CONFIG_RTC_CLK_SRC_INT_RC32K)
  Internal RC32K clock is unstable at extreme temperatures and is not recommended for use. This option will be removed in IDF v6.0.

CONFIG_RTC_CLK_CAL_CYCLES

Number of cycles for RTC_SLOW_CLK calibration

Found in: Component config > Hardware Settings > RTC Clock Config

When the startup code initializes RTC_SLOW_CLK, it can perform calibration by comparing the RTC_SLOW_CLK frequency with main XTAL frequency. This option sets the number of RTC_SLOW_CLK cycles measured by the calibration routine. Higher numbers increase calibration precision, which may be important for applications which spend a lot of time in deep sleep. Lower numbers reduce startup time.

When this option is set to 0, clock calibration will not be performed at startup, and approximate clock frequencies will be assumed:

- 150000 Hz if internal RC oscillator is used as clock source. For this use value 1024.
- **32768 Hz if the 32k crystal oscillator is used. For this use value 3000 or more.** In case more value will help improve the definition of the launch of the crystal. If the crystal could not start, it will be switched to internal RC.

Range:

- from 0 to 8190 if *CONFIG_RTC_CLK_SRC_EXT_CRYS* || *CONFIG_RTC_CLK_SRC_INT_RC32K*
- from 0 to 32766

Default value:

- 3000 if *CONFIG_RTC_CLK_SRC_EXT_CRYS* || *CONFIG_RTC_CLK_SRC_INT_RC32K*
- 1024

Peripheral Control Contains:

- *CONFIG_PERIPH_CTRL_FUNC_IN_IRAM*

CONFIG_PERIPH_CTRL_FUNC_IN_IRAM

Place peripheral control functions into IRAM

Found in: Component config > Hardware Settings > Peripheral Control

Place peripheral control functions (e.g. periph_module_reset) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context.

Default value:

- No (disabled)

ETM Configuration Contains:

- *CONFIG_ETM_ENABLE_DEBUG_LOG*

CONFIG_ETM_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Hardware Settings > ETM Configuration

whether to enable the debug log message for ETM core driver. Note that, this option only controls the ETM related driver log, won't affect other drivers.

Default value:

- No (disabled)

GDMA Configurations Contains:

- *CONFIG_GDMA_ENABLE_DEBUG_LOG*
- *CONFIG_GDMA_ISR_IRAM_SAFE*
- *CONFIG_GDMA_CTRL_FUNC_IN_IRAM*

CONFIG_GDMA_CTRL_FUNC_IN_IRAM

Place GDMA control functions in IRAM

Found in: Component config > Hardware Settings > GDMA Configurations

Place GDMA control functions (like start/stop/append/reset) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context.

Default value:

- No (disabled)

CONFIG_GDMA_ISR_IRAM_SAFE

GDMA ISR IRAM-Safe

Found in: Component config > Hardware Settings > GDMA Configurations

This will ensure the GDMA interrupt handler is IRAM-Safe, allow to avoid flash cache misses, and also be able to run whilst the cache is disabled. (e.g. SPI Flash write).

Default value:

- No (disabled)

CONFIG_GDMA_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Hardware Settings > GDMA Configurations

Whether to enable the debug log message for GDMA driver. Note that, this option only controls the GDMA driver log, won't affect other drivers.

Default value:

- No (disabled)

DW_GDMA Configurations Contains:

- *CONFIG_DW_GDMA_ENABLE_DEBUG_LOG*

CONFIG_DW_GDMA_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Hardware Settings > DW_GDMA Configurations

Whether to enable the debug log message for DW_GDMA driver. Note that, this option only controls the DW_GDMA driver log, won't affect other drivers.

Default value:

- No (disabled)

2D-DMA Configurations Contains:

- *CONFIG_DMA2D_ISR_IRAM_SAFE*
- *CONFIG_DMA2D_OPERATION_FUNC_IN_IRAM*

CONFIG_DMA2D_OPERATION_FUNC_IN_IRAM

Place 2D-DMA operation functions into IRAM

Found in: Component config > Hardware Settings > 2D-DMA Configurations

Place 2D-DMA all operation functions, including control functions (e.g. start/stop/append/reset) and setter functions (e.g. connect/strategy/callback registration) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. It also helps optimizing the performance.

Default value:

- No (disabled)

CONFIG_DMA2D_ISR_IRAM_SAFE

2D-DMA ISR IRAM-Safe

Found in: Component config > Hardware Settings > 2D-DMA Configurations

This will ensure the 2D-DMA interrupt handler is IRAM-Safe, allow to avoid flash cache misses, and also be able to run whilst the cache is disabled. (e.g. SPI Flash write).

Default value:

- No (disabled)

Main XTAL Config Contains:

- *CONFIG_XTAL_FREQ_SEL*

CONFIG_XTAL_FREQ_SEL

Main XTAL frequency

Found in: Component config > Hardware Settings > Main XTAL Config

This option selects the operating frequency of the XTAL (crystal) clock used to drive the ESP target. The selected value MUST reflect the frequency of the given hardware.

Note: The XTAL_FREQ_AUTO option allows the ESP target to automatically estimating XTAL clock's operating frequency. However, this feature is only supported on the ESP32. The ESP32 uses the internal 8MHZ as a reference when estimating. Due to the internal oscillator's frequency being temperature

dependent, usage of the XTAL_FREQ_AUTO is not recommended in applications that operate in high ambient temperatures or use high-temperature qualified chips and modules.

Available options:

- 24 MHz (CONFIG_XTAL_FREQ_24)
- 26 MHz (CONFIG_XTAL_FREQ_26)
- 32 MHz (CONFIG_XTAL_FREQ_32)
- 40 MHz (CONFIG_XTAL_FREQ_40)
- 48 MHz (CONFIG_XTAL_FREQ_48)
- Autodetect (CONFIG_XTAL_FREQ_AUTO)

Crypto DPA Protection Contains:

- *CONFIG_ESP_CRYPTO_DPA_PROTECTION_AT_STARTUP*

CONFIG_ESP_CRYPTO_DPA_PROTECTION_AT_STARTUP

Enable crypto DPA protection at startup

Found in: Component config > Hardware Settings > Crypto DPA Protection

This config controls the DPA (Differential Power Analysis) protection knob for the crypto peripherals. DPA protection dynamically adjusts the clock frequency of the crypto peripheral. DPA protection helps to make it difficult to perform SCA attacks on the crypto peripherals. However, there is also associated performance impact based on the security level set. Please refer to the TRM for more details.

Default value:

- Yes (enabled) if SOC_CRYPTO_DPA_PROTECTION_SUPPORTED

CONFIG_ESP_CRYPTO_DPA_PROTECTION_LEVEL

DPA protection level

Found in: Component config > Hardware Settings > Crypto DPA Protection > CONFIG_ESP_CRYPTO_DPA_PROTECTION_AT_STARTUP

Configure the DPA protection security level

Available options:

- Security level low (CONFIG_ESP_CRYPTO_DPA_PROTECTION_LEVEL_LOW)
- Security level medium (CONFIG_ESP_CRYPTO_DPA_PROTECTION_LEVEL_MEDIUM)
- Security level high (CONFIG_ESP_CRYPTO_DPA_PROTECTION_LEVEL_HIGH)

DCDC Regulator Configurations Contains:

- *CONFIG_ESP_SLEEP_KEEP_DCDC_ALWAYS_ON*

CONFIG_ESP_SLEEP_KEEP_DCDC_ALWAYS_ON

Keep DC-DC power always on during light-sleep

Found in: Component config > Hardware Settings > DCDC Regulator Configurations

ESP32P4 will switch the power supply to LDO before sleeping, and switch to DCDC after waking up. These two processes take a long time and may bring some risks for some short duration light sleep. (DCDC -> LDO: 2.5ms (max), LDO -> DCDC: 1.2 ms) Enabling this option will make chip powered by DCDC during light sleep to reduce some power switch risks, this will also increase the power consumption during the light sleep.

DO NOT DISABLE UNLESS YOU KNOW WHAT YOU ARE DOING.

Default value:
- Yes (enabled)

CONFIG_ESP_SLEEP_DCM_VSET_VAL_IN_SLEEP

DCDC voltage parameter during sleep

Found in: Component config > Hardware Settings > DCDC Regulator Configurations > CONFIG_ESP_SLEEP_KEEP_DCDC_ALWAYS_ON

This value determines the voltage of the DCDC chip during sleep. The same parameter value may correspond to different voltage values on different models of DCDC chips. Please update this value according to the model of external DCDC selected in your hardware solution.

For the DCDC chip model recommended by ESP, the recommended configuration values are listed below:

- TI-TLV62569/TLV62569P: 14

Range:
- from 0 to 31

Default value:
- 14

LDO Regulator Configurations

Contains:

- *CONFIG_ESP_LDO_RESERVE_PSRAM*
- *CONFIG_ESP_LDO_RESERVE_SPI_NOR_FLASH*

CONFIG_ESP_LDO_RESERVE_SPI_NOR_FLASH

Reserve one LDO regulator channel for SPI NOR Flash (READ HELP)

Found in: Component config > Hardware Settings > LDO Regulator Configurations

The LDO channel 1 can be used to power the SPI Flash chip, because the channel 1 is enabled by default after power on reset. If your SPI flash chip is not powered by ESP internal LDO, you can disable this option. Then you will free up one LDO channel for other general purpose.

Default value:
- Yes (enabled)

CONFIG_ESP_LDO_CHAN_SPI_NOR_FLASH_DOMAIN

LDO regulator channel that used to power SPI NOR Flash (READ HELP)

Found in: Component config > Hardware Settings > LDO Regulator Configurations > CONFIG_ESP_LDO_RESERVE_SPI_NOR_FLASH

Select which LDO channel to connect to the SPI Flash chip.

Range:
- from 1 to 1

Default value:
- 1

CONFIG_ESP_LDO_VOLTAGE_SPI_NOR_FLASH_DOMAIN

SPI NOR Flash power domain voltage

Found in: Component config > Hardware Settings > LDO Regulator Configurations > CONFIG_ESP_LDO_RESERVE_SPI_NOR_FLASH

Select the voltage used by the Flash power domain.

Available options:

- 3.3V (CONFIG_ESP_LDO_VOLTAGE_SPI_NOR_FLASH_3300_MV)

CONFIG_ESP_LDO_RESERVE_PSRAM

Reserve one LDO regulator channel for PSRAM (READ HELP)

Found in: Component config > Hardware Settings > LDO Regulator Configurations

The LDO channel 2 can be used to power the PSRAM chip. If the PSRAM chip is not powered by ESP internal LDO, you can disable this option. Then you will free up one LDO channel for other general purpose.

Default value:

- Yes (enabled)

CONFIG_ESP_LDO_CHAN_PSRAM_DOMAIN

LDO regulator channel that used to power PSRAM and MPLL (READ HELP)

Found in: Component config > Hardware Settings > LDO Regulator Configurations > CONFIG_ESP_LDO_RESERVE_PSRAM

Select which LDO channel to connect to the PSRAM chip.

Range:

- from 2 to 2

Default value:

- 2

CONFIG_ESP_LDO_VOLTAGE_PSRAM_DOMAIN

PSRAM power domain voltage

Found in: Component config > Hardware Settings > LDO Regulator Configurations > CONFIG_ESP_LDO_RESERVE_PSRAM

Select the voltage used by the PSRAM power domain.

Available options:

- 1.9V (CONFIG_ESP_LDO_VOLTAGE_PSRAM_1900_MV)

LCD and Touch Panel Contains:

- *LCD Peripheral Configuration*

LCD Peripheral Configuration Contains:

- *CONFIG_LCD_DSI_ISR_IRAM_SAFE*
- *CONFIG_LCD_ENABLE_DEBUG_LOG*
- *CONFIG_LCD_RGB_RESTART_IN_VSYNC*
- *CONFIG_LCD_RGB_ISR_IRAM_SAFE*

CONFIG_LCD_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration

whether to enable the debug log message for LCD driver. Note that, this option only controls the LCD driver log, won't affect other drivers.

Default value:

- No (disabled)

CONFIG_LCD_RGB_ISR_IRAM_SAFE

RGB LCD ISR IRAM-Safe

Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration

Ensure the LCD interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write). If you want the LCD driver to keep flushing the screen even when cache ops disabled, you can enable this option. Note, this will also increase the IRAM usage.

Default value:

- No (disabled) if SOC_LCD_RGB_SUPPORTED

CONFIG_LCD_RGB_RESTART_IN_VSYNC

Restart transmission in VSYNC

Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration

Reset the GDMA channel every VBlank to stop permanent desyncs from happening. Only need to enable it when in your application, the DMA can't deliver data as fast as the LCD consumes it.

Default value:

- No (disabled) if SOC_LCD_RGB_SUPPORTED

CONFIG_LCD_DSI_ISR_IRAM_SAFE

DSI LCD ISR IRAM-Safe

Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration

Ensure the LCD interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write). If you want the LCD driver to keep flushing the screen even when cache ops disabled, you can enable this option. Note, this will also increase the IRAM usage.

Default value:

- No (disabled)

ESP NETIF Adapter Contains:

- *CONFIG_ESP_NETIF_SET_DNS_PER_DEFAULT_NETIF*
- *CONFIG_ESP_NETIF_BRIDGE_EN*
- *CONFIG_ESP_NETIF_L2_TAP*
- *CONFIG_ESP_NETIF_IP_LOST_TIMER_INTERVAL*
- *CONFIG_ESP_NETIF_USE_TCPIP_STACK_LIB*

- *CONFIG_ESP_NETIF_RECEIVE_REPORT_ERRORS*

CONFIG_ESP_NETIF_IP_LOST_TIMER_INTERVAL

IP Address lost timer interval (seconds)

Found in: Component config > ESP NETIF Adapter

The value of 0 indicates the IP lost timer is disabled, otherwise the timer is enabled.

The IP address may be lost because of some reasons, e.g. when the station disconnects from soft-AP, or when DHCP IP renew fails etc. If the IP lost timer is enabled, it will be started every time the IP is lost. Event SYSTEM_EVENT_STA_LOST_IP will be raised if the timer expires. The IP lost timer is stopped if the station get the IP again before the timer expires.

Range:
- from 0 to 65535

Default value:
- 120

CONFIG_ESP_NETIF_USE_TCPIP_STACK_LIB

TCP/IP Stack Library

Found in: Component config > ESP NETIF Adapter

Choose the TCP/IP Stack to work, for example, LwIP, uIP, etc.

Available options:

- LwIP (CONFIG_ESP_NETIF_TCPIP_LWIP)
  lwIP is a small independent implementation of the TCP/IP protocol suite.
- Loopback (CONFIG_ESP_NETIF_LOOPBACK)
  Dummy implementation of esp-netif functionality which connects driver transmit to receive function. This option is for testing purpose only

CONFIG_ESP_NETIF_RECEIVE_REPORT_ERRORS

Use esp_err_t to report errors from esp_netif_receive

Found in: Component config > ESP NETIF Adapter

Enable if esp_netif_receive() should return error code. This is useful to inform upper layers that packet input to TCP/IP stack failed, so the upper layers could implement flow control. This option is disabled by default due to backward compatibility and will be enabled in v6.0 (IDF-7194)

Default value:
- No (disabled)

CONFIG_ESP_NETIF_L2_TAP

Enable netif L2 TAP support

Found in: Component config > ESP NETIF Adapter

A user program can read/write link layer (L2) frames from/to ESP TAP device. The ESP TAP device can be currently associated only with Ethernet physical interfaces.

CONFIG_ESP_NETIF_L2_TAP_MAX_FDS

Maximum number of opened L2 TAP File descriptors

Found in: Component config > ESP NETIF Adapter > CONFIG_ESP_NETIF_L2_TAP

Maximum number of opened File descriptors (FD's) associated with ESP TAP device. ESP TAP FD's take up a certain amount of memory, and allowing fewer FD's to be opened at the same time conserves memory.

Range:
- from 1 to 10 if *CONFIG_ESP_NETIF_L2_TAP*

Default value:
- 5 if *CONFIG_ESP_NETIF_L2_TAP*

CONFIG_ESP_NETIF_L2_TAP_RX_QUEUE_SIZE

Size of L2 TAP Rx queue

Found in: Component config > ESP NETIF Adapter > CONFIG_ESP_NETIF_L2_TAP

Maximum number of frames queued in opened File descriptor. Once the queue is full, the newly arriving frames are dropped until the queue has enough room to accept incoming traffic (Tail Drop queue management).

Range:
- from 1 to 100 if *CONFIG_ESP_NETIF_L2_TAP*

Default value:
- 20 if *CONFIG_ESP_NETIF_L2_TAP*

CONFIG_ESP_NETIF_BRIDGE_EN

Enable LwIP IEEE 802.1D bridge

Found in: Component config > ESP NETIF Adapter

Enable LwIP IEEE 802.1D bridge support in ESP-NETIF. Note that "Number of clients store data in netif" (LWIP_NUM_NETIF_CLIENT_DATA) option needs to be properly configured to be LwIP bridge available!

Default value:
- No (disabled)

CONFIG_ESP_NETIF_SET_DNS_PER_DEFAULT_NETIF

Enable DNS server per interface

Found in: Component config > ESP NETIF Adapter

Enable this option to use the DNS server which belongs to the selected default network interface. This feature collects DNS server and netif information from LWIP core modules. Whenever a new default netif is selected, global DNS servers in LWIP are updated with the netif related servers.

Default value:
- No (disabled)

Partition API Configuration

Power Management Contains:

- *CONFIG_PM_LIGHTSLEEP_RTC_OSC_CAL_INTERVAL*
- *CONFIG_PM_SLP_DISABLE_GPIO*
- *CONFIG_PM_LIGHT_SLEEP_CALLBACKS*

- *CONFIG_PM_POWER_DOWN_CPU_IN_LIGHT_SLEEP*
- *CONFIG_PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP*
- *CONFIG_PM_SLP_IRAM_OPT*
- *CONFIG_PM_RTOS_IDLE_OPT*
- *CONFIG_PM_ENABLE*

CONFIG_PM_ENABLE

Support for power management

Found in: Component config > Power Management

If enabled, application is compiled with support for power management. This option has run-time overhead (increased interrupt latency, longer time to enter idle state), and it also reduces accuracy of RTOS ticks and timers used for timekeeping. Enable this option if application uses power management APIs.

Default value:

- No (disabled)

CONFIG_PM_DFS_INIT_AUTO

Enable dynamic frequency scaling (DFS) at startup

Found in: Component config > Power Management > CONFIG_PM_ENABLE

If enabled, startup code configures dynamic frequency scaling. Max CPU frequency is set to DEFAULT_CPU_FREQ_MHZ setting, min frequency is set to XTAL frequency. If disabled, DFS will not be active until the application configures it using esp_pm_configure function.

Default value:

- No (disabled) if *CONFIG_PM_ENABLE*

CONFIG_PM_PROFILING

Enable profiling counters for PM locks

Found in: Component config > Power Management > CONFIG_PM_ENABLE

If enabled, esp_pm_* functions will keep track of the amount of time each of the power management locks has been held, and esp_pm_dump_locks function will print this information. This feature can be used to analyze which locks are preventing the chip from going into a lower power state, and see what time the chip spends in each power saving mode. This feature does incur some run-time overhead, so should typically be disabled in production builds.

Default value:

- No (disabled) if *CONFIG_PM_ENABLE*

CONFIG_PM_TRACE

Enable debug tracing of PM using GPIOs

Found in: Component config > Power Management > CONFIG_PM_ENABLE

If enabled, some GPIOs will be used to signal events such as RTOS ticks, frequency switching, entry/exit from idle state. Refer to pm_trace.c file for the list of GPIOs. This feature is intended to be used when analyzing/debugging behavior of power management implementation, and should be kept disabled in applications.

Default value:

- No (disabled) if *CONFIG_PM_ENABLE*

CONFIG_PM_SLP_IRAM_OPT

Put lightsleep related codes in internal RAM

Found in: Component config > Power Management

If enabled, about 2.1KB of lightsleep related source code would be in IRAM and chip would sleep longer for 310us at 160MHz CPU frequency most each time. This feature is intended to be used when lower power consumption is needed while there is enough place in IRAM to place source code.

CONFIG_PM_RTOS_IDLE_OPT

Put RTOS IDLE related codes in internal RAM

Found in: Component config > Power Management

If enabled, about 180Bytes of RTOS_IDLE related source code would be in IRAM and chip would sleep longer for 20us at 160MHz CPU frequency most each time. This feature is intended to be used when lower power consumption is needed while there is enough place in IRAM to place source code.

CONFIG_PM_SLP_DISABLE_GPIO

Disable all GPIO when chip at sleep

Found in: Component config > Power Management

This feature is intended to disable all GPIO pins at automantic sleep to get a lower power mode. If enabled, chips will disable all GPIO pins at automantic sleep to reduce about 200~300 uA current. If you want to specifically use some pins normally as chip wakes when chip sleeps, you can call 'gpio_sleep_sel_dis' to disable this feature on those pins. You can also keep this feature on and call 'gpio_sleep_set_direction' and 'gpio_sleep_set_pull_mode' to have a different GPIO configuration at sleep. Warning: If you want to enable this option on ESP32, you should enable *GPIO_ESP32_SUPPORT_SWITCH_SLP_PULL* at first, otherwise you will not be able to switch pullup/pulldown mode.

CONFIG_PM_LIGHTSLEEP_RTC_OSC_CAL_INTERVAL

Calibrate the RTC_FAST/SLOW clock every N times of light sleep

Found in: Component config > Power Management

The value of this option determines the calibration interval of the RTC_FAST/SLOW clock during sleep when power management is enabled. When it is configured as N, the RTC_FAST/SLOW clock will be calibrated every N times of lightsleep. Decreasing this value will increase the time the chip is in the active state, thereby increasing the average power consumption of the chip. Increasing this value can reduce the average power consumption, but when the external environment changes drastically and the chip RTC_FAST/SLOW oscillator frequency drifts, it may cause system instability.

Range:

- from 1 to 128 if *CONFIG_PM_ENABLE*

Default value:

- 1 if *CONFIG_PM_ENABLE*

CONFIG_PM_POWER_DOWN_CPU_IN_LIGHT_SLEEP

Power down CPU in light sleep

Found in: Component config > Power Management

If enabled, the CPU will be powered down in light sleep, ESP chips supports saving and restoring CPU's running context before and after light sleep, the feature provides applications with seamless CPU powerdowned lightsleep without user awareness. But this will takes up some internal memory. On esp32c3

soc, enabling this option will consume 1.68 KB of internal RAM and will reduce sleep current consumption by about 100 uA. On esp32s3 soc, enabling this option will consume 8.58 KB of internal RAM and will reduce sleep current consumption by about 650 uA.

Default value:

- Yes (enabled) if SOC_PM_SUPPORT_CPU_PD

CONFIG_PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP

Power down Digital Peripheral in light sleep (EXPERIMENTAL)

Found in: Component config > Power Management

If enabled, digital peripherals will be powered down in light sleep, it will reduce sleep current consumption by about 100 uA. Chip will save/restore register context at sleep/wake time to keep the system running. Enabling this option will increase static RAM and heap usage, the actual cost depends on the peripherals you have initialized. In order to save/restore the context of the necessary hardware for FreeRTOS to run, it will need at least 4.55 KB free heap at sleep time. Otherwise sleep will not power down the peripherals.

Note1: Please use this option with caution, the current IDF does not support the retention of all peripherals. When the digital peripherals are powered off and a sleep and wake-up is completed, the peripherals that have not saved the running context are equivalent to performing a reset. !!! Please confirm the peripherals used in your application and their sleep retention support status before enabling this option, peripherals sleep retention driver support status is tracked in power_management.rst

Note2: When this option is enabled simultaneously with FREERTOS_USE_TICKLESS_IDLE, since the UART will be powered down, the uart FIFO will be flushed before sleep to avoid data loss, however, this has the potential to block the sleep process and cause the wakeup time to be skipped, which will cause the tick of freertos to not be compensated correctly when returning from sleep and cause the system to crash. To avoid this, you can increase FREERTOS_IDLE_TIME_BEFORE_SLEEP threshold in menuconfig.

Default value:

- No (disabled)

CONFIG_PM_LIGHT_SLEEP_CALLBACKS

Enable registration of pm light sleep callbacks

Found in: Component config > Power Management

If enabled, it allows user to register entry and exit callbacks which are called before and after entering auto light sleep.

NOTE: These callbacks are executed from the IDLE task context hence you cannot have any blocking calls in your callbacks.

NOTE: Enabling these callbacks may change sleep duration calculations based on time spent in callback and hence it is highly recommended to keep them as short as possible

Default value:

- No (disabled) if *CONFIG_FREERTOS_USE_TICKLESS_IDLE*

ESP PSRAM

Contains:

- *CONFIG_SPIRAM*

CONFIG_SPIRAM

Support for external PSRAM

Found in: Component config > ESP PSRAM

This enables support for an external PSRAM chip, connected in parallel with the main SPI flash chip.

PSRAM config Contains:

- *CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY*
- *CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY*
- *CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY*
- *CONFIG_SPIRAM_XIP_FROM_PSRAM*
- *CONFIG_SPIRAM_ECC_ENABLE*
- *CONFIG_SPIRAM_BOOT_INIT*
- *CONFIG_SPIRAM_MODE*
- *CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL*
- *CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL*
- *CONFIG_SPIRAM_MEMTEST*
- *CONFIG_SPIRAM_SPEED*
- *CONFIG_SPIRAM_USE*
- *CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP*

CONFIG_SPIRAM_MODE

Line Mode of PSRAM chip in use

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

Available options:

- 16-Line-Mode PSRAM (CONFIG_SPIRAM_MODE_HEX)

CONFIG_SPIRAM_SPEED

Set PSRAM clock speed

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

Select the speed for the PSRAM chip.

Available options:

- 200MHz clock speed (CONFIG_SPIRAM_SPEED_200M)
- 20MHz clock speed (CONFIG_SPIRAM_SPEED_20M)

CONFIG_SPIRAM_XIP_FROM_PSRAM

Enable Executable in place from (XiP) from PSRAM feature (READ HELP)

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

If enabled, firmware in flash including instructions and data will be moved into PSRAM on startup, firmware code will execute directly from PSRAM.

With this option enabled, code that requires execution during an MSPI1 Flash operation does not have to be placed in IRAM. Therefore codes that need to be executing during Flash operations can continue working normally.

Because P4 flash and PSRAM are using two separate SPI buses, moving flash content to PSRAM will actually increase the load of the PSRAM MSPI bus, so the exact impact on performance will be dependent on your app usage of PSRAM. For example, as the PSRAM bus speed could be much faster than flash bus speed, if the instructions and data that are used to be in flash are not accessed very frequently, you might get better performance with this option enabled. We suggest doing performance profiling to determine if enabling this option.

CONFIG_SPIRAM_ECC_ENABLE

Enable PSRAM ECC

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

Enable Error-Correcting Code function when accessing PSRAM.

If enabled, 1/8 of the PSRAM total size will be reserved for error-correcting code.

CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY

Allow external memory as an argument to xTaskCreateStatic

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

Accessing memory in PSRAM has certain restrictions, so task stacks allocated by xTaskCreate are by default allocated from internal RAM.

This option allows for passing memory allocated from PSRAM to be passed to xTaskCreateStatic. This should only be used for tasks where the stack is never accessed while the L2Cache is disabled, e.g. during SPI Flash operations

CONFIG_SPIRAM_BOOT_INIT

Initialize SPI RAM during startup

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

If this is enabled, the SPI RAM will be enabled during initial boot. Unless you have specific requirements, you'll want to leave this enabled so memory allocated during boot-up can also be placed in SPI RAM.

CONFIG_SPIRAM_IGNORE_NOTFOUND

Ignore PSRAM when not found

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config > CONFIG_SPIRAM_BOOT_INIT

Normally, if psram initialization is enabled during compile time but not found at runtime, it is seen as an error making the CPU panic. If this is enabled, booting will complete but no PSRAM will be available. If PSRAM failed to initialize, the following configs may be affected and may need to be corrected manually. SPIRAM_TRY_ALLOCATE_WIFI_LWIP will affect some LWIP and WiFi buffer default values and range values. Enable SPIRAM_TRY_ALLOCATE_WIFI_LWIP, ESP_WIFI_AMSDU_TX_ENABLED, ESP_WIFI_CACHE_TX_BUFFER_NUM and use static WiFi Tx buffer may cause potential memory exhaustion issues. Suggest disable SPIRAM_TRY_ALLOCATE_WIFI_LWIP. Suggest disable ESP_WIFI_AMSDU_TX_ENABLED. Suggest disable ESP_WIFI_CACHE_TX_BUFFER_NUM, need clear CONFIG_FEATURE_CACHE_TX_BUF_BIT of config->feature_caps. Suggest change ESP_WIFI_TX_BUFFER from static to dynamic. Also suggest to adjust some buffer numbers to the values used without PSRAM case. Such as, ESP_WIFI_STATIC_TX_BUFFER_NUM, ESP_WIFI_DYNAMIC_TX_BUFFER_NUM.

CONFIG_SPIRAM_USE

SPI RAM access method

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

The SPI RAM can be accessed in multiple methods: by just having it available as an unmanaged memory region in the CPU's memory map, by integrating it in the heap as 'special' memory needing heap_caps_malloc to allocate, or by fully integrating it making malloc() also able to return SPI RAM pointers.

Available options:

- Integrate RAM into memory map (CONFIG_SPIRAM_USE_MEMMAP)
- Make RAM allocatable using heap_caps_malloc(..., MALLOC_CAP_SPIRAM) (CONFIG_SPIRAM_USE_CAPS_ALLOC)
- Make RAM allocatable using malloc() as well (CONFIG_SPIRAM_USE_MALLOC)

CONFIG_SPIRAM_MEMTEST

Run memory test on SPI RAM initialization

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

Runs a rudimentary memory test on initialization. Aborts when memory test fails. Disable this for slightly faster startup.

CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL

Maximum malloc() size, in bytes, to always put in internal memory

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

If malloc() is capable of also allocating SPI-connected ram, its allocation strategy will prefer to allocate chunks less than this size in internal memory, while allocations larger than this will be done from external RAM. If allocation from the preferred region fails, an attempt is made to allocate from the non-preferred region instead, so malloc() will not suddenly fail when either internal or external memory is full.

CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP

Try to allocate memories of WiFi and LWIP in SPIRAM firstly. If failed, allocate internal memory

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

Try to allocate memories of WiFi and LWIP in SPIRAM firstly. If failed, try to allocate internal memory then.

CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL

Reserve this amount of bytes for data that specifically needs to be in DMA or internal memory

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

Because the external/internal RAM allocation strategy is not always perfect, it sometimes may happen that the internal memory is entirely filled up. This causes allocations that are specifically done in internal memory, for example the stack for new tasks or memory to service DMA or have memory that's also available when SPI cache is down, to fail. This option reserves a pool specifically for requests like that; the memory in this pool is not given out when a normal malloc() is called.

Set this to 0 to disable this feature.

Note that because FreeRTOS stacks are forced to internal memory, they will also use this memory pool; be sure to keep this in mind when adjusting this value.

Note also that the DMA reserved pool may not be one single contiguous memory region, depending on the configured size and the static memory usage of the app.

CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY

Allow .bss segment placed in external memory

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

If enabled, variables with EXT_RAM_BSS_ATTR attribute will be placed in SPIRAM instead of internal DRAM. BSS section of *lwip*, *net80211*, *pp*, *bt* libraries will be automatically placed in SPIRAM. BSS sections from other object files and libraries can also be placed in SPIRAM through linker fragment scheme *extram_bss*.

Note that the variables placed in SPIRAM using EXT_RAM_BSS_ATTR will be zero initialized.

CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY

Allow .noinit segment placed in external memory

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > PSRAM config

If enabled, noinit variables can be placed in PSRAM using EXT_RAM_NOINIT_ATTR.

Note the values placed into this section will not be initialized at startup and should keep its value after software restart.

ESP Ringbuf Contains:

- *CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH*

CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH

Place non-ISR ringbuf functions into flash

Found in: Component config > ESP Ringbuf

Place non-ISR ringbuf functions (like xRingbufferCreate/xRingbufferSend) into flash. This frees up IRAM, but the functions can no longer be called when the cache is disabled.

Default value:

- No (disabled)

CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH

Place ISR ringbuf functions into flash

Found in: Component config > ESP Ringbuf > CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH

Place ISR ringbuf functions (like xRingbufferSendFromISR/xRingbufferReceiveFromISR) into flash. This frees up IRAM, but the functions can no longer be called when the cache is disabled or from an IRAM interrupt context.

This option is not compatible with ESP-IDF drivers which are configured to run the ISR from an IRAM context, e.g. CONFIG_UART_ISR_IN_IRAM.

Default value:

- No (disabled) if *CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH*

ESP System Settings Contains:

- *CONFIG_ESP_SYSTEM_RTC_EXT_XTAL_BOOTSTRAP_CYCLES*
- *Brownout Detector*
- *Cache config*
- *CONFIG_ESP_CONSOLE_UART*
- *CONFIG_ESP_CONSOLE_SECONDARY*
- *CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ*

- *CONFIG_ESP_SYSTEM_ALLOW_RTC_FAST_MEM_AS_HEAP*
- *CONFIG_ESP_TASK_WDT_EN*
- *CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE*
- *CONFIG_ESP_SYSTEM_USE_EH_FRAME*
- *CONFIG_ESP_SYSTEM_HW_PC_RECORD*
- *CONFIG_ESP_SYSTEM_HW_STACK_GUARD*
- *CONFIG_ESP_XT_WDT*
- *CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL*
- *CONFIG_ESP_INT_WDT*
- *CONFIG_ESP_MAIN_TASK_AFFINITY*
- *CONFIG_ESP_MAIN_TASK_STACK_SIZE*
- *CONFIG_ESP_DEBUG_OCDAWARE*
- *Memory protection*
- *CONFIG_ESP_MINIMAL_SHARED_STACK_SIZE*
- *CONFIG_ESP_DEBUG_STUBS_ENABLE*
- *CONFIG_ESP_SYSTEM_PANIC*
- *CONFIG_ESP_SYSTEM_PANIC_REBOOT_DELAY_SECONDS*
- *CONFIG_ESP_PANIC_HANDLER_IRAM*
- *CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE*
- *CONFIG_ESP_CONSOLE_UART_BAUDRATE*
- *CONFIG_ESP_CONSOLE_UART_NUM*
- *CONFIG_ESP_CONSOLE_UART_RX_GPIO*
- *CONFIG_ESP_CONSOLE_UART_TX_GPIO*

CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ

CPU frequency

Found in: Component config > ESP System Settings

CPU frequency to be set on application startup.

Available options:

- 40 MHz (CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ_40)
- 360 MHz (CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ_360)

Cache config Contains:

- *CONFIG_CACHE_L2_CACHE_LINE_SIZE*
- *CONFIG_CACHE_L2_CACHE_SIZE*

CONFIG_CACHE_L2_CACHE_SIZE

L2 cache size

Found in: Component config > ESP System Settings > Cache config

L2 cache size to be set on application startup.

Available options:

- 128KB (CONFIG_CACHE_L2_CACHE_128KB)
- 256KB (CONFIG_CACHE_L2_CACHE_256KB)
- 512KB (CONFIG_CACHE_L2_CACHE_512KB)

CONFIG_CACHE_L2_CACHE_LINE_SIZE

L2 cache line size

Found in: Component config > ESP System Settings > Cache config

L2 cache line size to be set on application startup.

Available options:

- 64 Bytes (CONFIG_CACHE_L2_CACHE_LINE_64B)
- 128 Bytes (CONFIG_CACHE_L2_CACHE_LINE_128B)

CONFIG_ESP_SYSTEM_PANIC

Panic handler behaviour

Found in: Component config > ESP System Settings

If FreeRTOS detects unexpected behaviour or an unhandled exception, the panic handler is invoked. Configure the panic handler's action here.

Available options:

- Print registers and halt (CONFIG_ESP_SYSTEM_PANIC_PRINT_HALT)
  Outputs the relevant registers over the serial port and halt the processor. Needs a manual reset to restart.
- Print registers and reboot (CONFIG_ESP_SYSTEM_PANIC_PRINT_REBOOT)
  Outputs the relevant registers over the serial port and immediately reset the processor.
- Silent reboot (CONFIG_ESP_SYSTEM_PANIC_SILENT_REBOOT)
  Just resets the processor without outputting anything
- GDBStub on panic (CONFIG_ESP_SYSTEM_PANIC_GDBSTUB)
  Invoke gdbstub on the serial port, allowing for gdb to attach to it to do a postmortem of the crash.

CONFIG_ESP_SYSTEM_PANIC_REBOOT_DELAY_SECONDS

Panic reboot delay (Seconds)

Found in: Component config > ESP System Settings

After the panic handler executes, you can specify a number of seconds to wait before the device reboots.

Range:

- from 0 to 99

Default value:

- 0

CONFIG_ESP_SYSTEM_RTC_EXT_XTAL_BOOTSTRAP_CYCLES

Bootstrap cycles for external 32kHz crystal

Found in: Component config > ESP System Settings

To reduce the startup time of an external RTC crystal, we bootstrap it with a 32kHz square wave for a fixed number of cycles. Setting 0 will disable bootstrapping (if disabled, the crystal may take longer to start up or fail to oscillate under some conditions).

If this value is too high, a faulty crystal may initially start and then fail. If this value is too low, an otherwise good crystal may not start.

To accurately determine if the crystal has started, set a larger "Number of cycles for RTC_SLOW_CLK calibration" (about 3000).

CONFIG_ESP_SYSTEM_ALLOW_RTC_FAST_MEM_AS_HEAP

Enable RTC fast memory for dynamic allocations

Found in: Component config > ESP System Settings

This config option allows to add RTC fast memory region to system heap with capability similar to that of DRAM region but without DMA. This memory will be consumed first per heap initialization order by early startup services and scheduler related code. Speed wise RTC fast memory operates on APB clock and hence does not have much performance impact.

Default value:

- Yes (enabled)

CONFIG_ESP_SYSTEM_USE_EH_FRAME

Generate and use eh_frame for backtracing

Found in: Component config > ESP System Settings

Generate DWARF information for each function of the project. These information will parsed and used to perform backtracing when panics occur. Activating this option will activate asynchronous frame unwinding and generation of both .eh_frame and .eh_frame_hdr sections, resulting in a bigger binary size (20% to 100% larger). The main purpose of this option is to be able to have a backtrace parsed and printed by the program itself, regardless of the serial monitor used. This option shall NOT be used for production.

Default value:

- No (disabled)

Memory protection Contains:

- *CONFIG_ESP_SYSTEM_PMP_IDRAM_SPLIT*
- *CONFIG_ESP_SYSTEM_MEMPROT_FEATURE*

CONFIG_ESP_SYSTEM_PMP_IDRAM_SPLIT

Enable IRAM/DRAM split protection

Found in: Component config > ESP System Settings > Memory protection

If enabled, the CPU watches all the memory access and raises an exception in case of any memory violation. This feature automatically splits the SRAM memory, using PMP, into data and instruction segments and sets Read/Execute permissions for the instruction part (below given splitting address) and Read/Write permissions for the data part (above the splitting address). The memory protection is effective on all access through the IRAM0 and DRAM0 buses.

Default value:

- Yes (enabled)

CONFIG_ESP_SYSTEM_MEMPROT_FEATURE

Enable memory protection

Found in: Component config > ESP System Settings > Memory protection

If enabled, the permission control module watches all the memory access and fires the panic handler if a permission violation is detected. This feature automatically splits the SRAM memory into data and instruction segments and sets Read/Execute permissions for the instruction part (below given splitting address) and Read/Write permissions for the data part (above the splitting address). The memory protection is effective on all access through the IRAM0 and DRAM0 buses.

Default value:

- Yes (enabled) if SOC_MEMPROT_SUPPORTED

CONFIG_ESP_SYSTEM_MEMPROT_FEATURE_LOCK

Lock memory protection settings

Found in: Component config > ESP System Settings > Memory protection > CONFIG_ESP_SYSTEM_MEMPROT_FEATURE

Once locked, memory protection settings cannot be changed anymore. The lock is reset only on the chip startup.

Default value:

- Yes (enabled) if *CONFIG_ESP_SYSTEM_MEMPROT_FEATURE*

CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE

System event queue size

Found in: Component config > ESP System Settings

Config system event queue size in different application.

Default value:

- 32

CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE

Event loop task stack size

Found in: Component config > ESP System Settings

Config system event task stack size in different application.

Default value:

- 2304

CONFIG_ESP_MAIN_TASK_STACK_SIZE

Main task stack size

Found in: Component config > ESP System Settings

Configure the ”main task” stack size. This is the stack of the task which calls app_main(). If app_main() returns then this task is deleted and its stack memory is freed.

Default value:

- 3584

CONFIG_ESP_MAIN_TASK_AFFINITY

Main task core affinity

Found in: Component config > ESP System Settings

Configure the ”main task” core affinity. This is the used core of the task which calls app_main(). If app_main() returns then this task is deleted.

Available options:

- CPU0 (CONFIG_ESP_MAIN_TASK_AFFINITY_CPU0)
- CPU1 (CONFIG_ESP_MAIN_TASK_AFFINITY_CPU1)
- No affinity (CONFIG_ESP_MAIN_TASK_AFFINITY_NO_AFFINITY)

CONFIG_ESP_MINIMAL_SHARED_STACK_SIZE

Minimal allowed size for shared stack

Found in: Component config > ESP System Settings

Minimal value of size, in bytes, accepted to execute a expression with shared stack.

Default value:

- 2048

CONFIG_ESP_CONSOLE_UART

Channel for console output

Found in: Component config > ESP System Settings

Select where to send console output (through stdout and stderr).

- Default is to use UART0 on pre-defined GPIOs.
- If "Custom" is selected, UART0 or UART1 can be chosen, and any pins can be selected.
- If "None" is selected, there will be no console output on any UART, except for initial output from ROM bootloader. This ROM output can be suppressed by GPIO strapping or EFUSE, refer to chip datasheet for details.
- On chips with USB OTG peripheral, "USB CDC" option redirects output to the CDC port. This option uses the CDC driver in the chip ROM. This option is incompatible with TinyUSB stack.
- On chips with an USB serial/JTAG debug controller, selecting the option for that redirects output to the CDC/ACM (serial port emulation) component of that device.

Available options:

- Default: UART0 (CONFIG_ESP_CONSOLE_UART_DEFAULT)
- USB CDC (CONFIG_ESP_CONSOLE_USB_CDC)
- USB Serial/JTAG Controller (CONFIG_ESP_CONSOLE_USB_SERIAL_JTAG)
- Custom UART (CONFIG_ESP_CONSOLE_UART_CUSTOM)
- None (CONFIG_ESP_CONSOLE_NONE)

CONFIG_ESP_CONSOLE_SECONDARY

Channel for console secondary output

Found in: Component config > ESP System Settings

This secondary option supports output through other specific port like USB_SERIAL_JTAG when UART0 port as a primary is selected but not connected. This secondary output currently only supports non-blocking mode without using REPL. If you want to output in blocking mode with REPL or input through this secondary port, please change the primary config to this port in *Channel for console output* menu.

Available options:

- No secondary console (CONFIG_ESP_CONSOLE_SECONDARY_NONE)
- USB_SERIAL_JTAG PORT (CONFIG_ESP_CONSOLE_SECONDARY_USB_SERIAL_JTAG)
  This option supports output through USB_SERIAL_JTAG port when the UART0 port is not connected. The output currently only supports non-blocking mode without using the console. If you want to output in blocking mode with REPL or input through USB_SERIAL_JTAG port, please change the primary config to ESP_CONSOLE_USB_SERIAL_JTAG above.

CONFIG_ESP_CONSOLE_UART_NUM

UART peripheral to use for console output (0-1)

Found in: Component config > ESP System Settings

This UART peripheral is used for console output from the ESP-IDF Bootloader and the app.

If the configuration is different in the Bootloader binary compared to the app binary, UART is reconfigured after the bootloader exits and the app starts.

Due to an ESP32 ROM bug, UART2 is not supported for console output via esp_rom_printf.

Available options:

- UART0 (CONFIG_ESP_CONSOLE_UART_CUSTOM_NUM_0)
- UART1 (CONFIG_ESP_CONSOLE_UART_CUSTOM_NUM_1)

CONFIG_ESP_CONSOLE_UART_TX_GPIO

UART TX on GPIO#

Found in: Component config > ESP System Settings

This GPIO is used for console UART TX output in the ESP-IDF Bootloader and the app (including boot log output and default standard output and standard error of the app).

If the configuration is different in the Bootloader binary compared to the app binary, UART is reconfigured after the bootloader exits and the app starts.

Range:

- from 0 to 54 if *CONFIG_ESP_CONSOLE_UART_CUSTOM*

Default value:

- 37 if *CONFIG_ESP_CONSOLE_UART_CUSTOM*
- 43 if *CONFIG_ESP_CONSOLE_UART_CUSTOM*

CONFIG_ESP_CONSOLE_UART_RX_GPIO

UART RX on GPIO#

Found in: Component config > ESP System Settings

This GPIO is used for UART RX input in the ESP-IDF Bootloader and the app (including default default standard input of the app).

Note: The default ESP-IDF Bootloader configures this pin but doesn't read anything from the UART.

If the configuration is different in the Bootloader binary compared to the app binary, UART is reconfigured after the bootloader exits and the app starts.

Range:

- from 0 to 54 if *CONFIG_ESP_CONSOLE_UART_CUSTOM*

Default value:

- 38 if *CONFIG_ESP_CONSOLE_UART_CUSTOM*
- 44 if *CONFIG_ESP_CONSOLE_UART_CUSTOM*

CONFIG_ESP_CONSOLE_UART_BAUDRATE

UART console baud rate

Found in: Component config > ESP System Settings

This baud rate is used by both the ESP-IDF Bootloader and the app (including boot log output and default standard input/output/error of the app).

The app's maximum baud rate depends on the UART clock source. If Power Management is disabled, the UART clock source is the APB clock and all baud rates in the available range will be sufficiently accurate. If Power Management is enabled, REF_TICK clock source is used so the baud rate is divided from 1MHz. Baud rates above 1Mbps are not possible and values between 500Kbps and 1Mbps may not be accurate.

If the configuration is different in the Bootloader binary compared to the app binary, UART is reconfigured after the bootloader exits and the app starts.

Range:
- from 1200 to 1000000 if *CONFIG_PM_ENABLE*

Default value:
- 115200

CONFIG_ESP_INT_WDT

Interrupt watchdog

Found in: Component config > ESP System Settings

This watchdog timer can detect if the FreeRTOS tick interrupt has not been called for a certain time, either because a task turned off interrupts and did not turn them on for a long time, or because an interrupt handler did not return. It will try to invoke the panic handler first and failing that reset the SoC.

Default value:
- Yes (enabled)

CONFIG_ESP_INT_WDT_TIMEOUT_MS

Interrupt watchdog timeout (ms)

Found in: Component config > ESP System Settings > CONFIG_ESP_INT_WDT

The timeout of the watchdog, in milliseconds. Make this higher than the FreeRTOS tick rate.

Range:
- from 10 to 10000

Default value:
- 300

CONFIG_ESP_INT_WDT_CHECK_CPU1

Also watch CPU1 tick interrupt

Found in: Component config > ESP System Settings > CONFIG_ESP_INT_WDT

Also detect if interrupts on CPU 1 are disabled for too long.

CONFIG_ESP_TASK_WDT_EN

Enable Task Watchdog Timer

Found in: Component config > ESP System Settings

The Task Watchdog Timer can be used to make sure individual tasks are still running. Enabling this option will enable the Task Watchdog Timer. It can be either initialized automatically at startup or initialized after startup (see Task Watchdog Timer API Reference)

Default value:
- Yes (enabled)

CONFIG_ESP_TASK_WDT_INIT

Initialize Task Watchdog Timer on startup

Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN

Enabling this option will cause the Task Watchdog Timer to be initialized automatically at startup.

Default value:

- Yes (enabled)

CONFIG_ESP_TASK_WDT_PANIC

Invoke panic handler on Task Watchdog timeout

Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN > CONFIG_ESP_TASK_WDT_INIT

If this option is enabled, the Task Watchdog Timer will be configured to trigger the panic handler when it times out. This can also be configured at run time (see Task Watchdog Timer API Reference)

Default value:

- No (disabled)

CONFIG_ESP_TASK_WDT_TIMEOUT_S

Task Watchdog timeout period (seconds)

Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN > CONFIG_ESP_TASK_WDT_INIT

Timeout period configuration for the Task Watchdog Timer in seconds. This is also configurable at run time (see Task Watchdog Timer API Reference)

Range:

- from 1 to 60

Default value:

- 5

CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0

Watch CPU0 Idle Task

Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN > CONFIG_ESP_TASK_WDT_INIT

If this option is enabled, the Task Watchdog Timer will watch the CPU0 Idle Task. Having the Task Watchdog watch the Idle Task allows for detection of CPU starvation as the Idle Task not being called is usually a symptom of CPU starvation. Starvation of the Idle Task is detrimental as FreeRTOS household tasks depend on the Idle Task getting some runtime every now and then.

Default value:

- Yes (enabled)

CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1

Watch CPU1 Idle Task

Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN > CONFIG_ESP_TASK_WDT_INIT

If this option is enabled, the Task Watchdog Timer will wach the CPU1 Idle Task.

CONFIG_ESP_XT_WDT

Initialize XTAL32K watchdog timer on startup

Found in: Component config > ESP System Settings

This watchdog timer can detect oscillation failure of the XTAL32K_CLK. When such a failure is detected the hardware can be set up to automatically switch to BACKUP32K_CLK and generate an interrupt.

CONFIG_ESP_XT_WDT_TIMEOUT

XTAL32K watchdog timeout period

Found in: Component config > ESP System Settings > CONFIG_ESP_XT_WDT

Timeout period configuration for the XTAL32K watchdog timer based on RTC_CLK.

Range:

- from 1 to 255 if *CONFIG_ESP_XT_WDT*

Default value:

- 200 if *CONFIG_ESP_XT_WDT*

CONFIG_ESP_XT_WDT_BACKUP_CLK_ENABLE

Automatically switch to BACKUP32K_CLK when timer expires

Found in: Component config > ESP System Settings > CONFIG_ESP_XT_WDT

Enable this to automatically switch to BACKUP32K_CLK as the source of RTC_SLOW_CLK when the watchdog timer expires.

Default value:

- Yes (enabled) if *CONFIG_ESP_XT_WDT*

CONFIG_ESP_PANIC_HANDLER_IRAM

Place panic handler code in IRAM

Found in: Component config > ESP System Settings

If this option is disabled (default), the panic handler code is placed in flash not IRAM. This means that if ESP-IDF crashes while flash cache is disabled, the panic handler will automatically re-enable flash cache before running GDB Stub or Core Dump. This adds some minor risk, if the flash cache status is also corrupted during the crash.

If this option is enabled, the panic handler code (including required UART functions) is placed in IRAM. This may be necessary to debug some complex issues with crashes while flash cache is disabled (for example, when writing to SPI flash) or when flash cache is corrupted when an exception is triggered.

Default value:

- No (disabled)

CONFIG_ESP_DEBUG_STUBS_ENABLE

OpenOCD debug stubs

Found in: Component config > ESP System Settings

Debug stubs are used by OpenOCD to execute pre-compiled onboard code which does some useful debugging stuff, e.g. GCOV data dump.

CONFIG_ESP_DEBUG_OCDAWARE

Make exception and panic handlers JTAG/OCD aware

Found in: Component config > ESP System Settings

The FreeRTOS panic and unhandled exception handers can detect a JTAG OCD debugger and instead of panicking, have the debugger stop on the offending instruction.

Default value:

- Yes (enabled)

CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL

Interrupt level to use for Interrupt Watchdog and other system checks

Found in: Component config > ESP System Settings

Interrupt level to use for Interrupt Watchdog, IPC_ISR and other system checks.

Available options:

- Level 5 interrupt (CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL_5)
  Using level 5 interrupt for Interrupt Watchdog, IPC_ISR and other system checks.
- Level 4 interrupt (CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL_4)
  Using level 4 interrupt for Interrupt Watchdog, IPC_ISR and other system checks.

Brownout Detector Contains:

- *CONFIG_ESP_BROWNOUT_DET*

CONFIG_ESP_BROWNOUT_DET

Hardware brownout detect & reset

Found in: Component config > ESP System Settings > Brownout Detector

The ESP32-P4 has a built-in brownout detector which can detect if the voltage is lower than a specific value. If this happens, it will reset the chip in order to prevent unintended behaviour.

Default value:

- Yes (enabled)

CONFIG_ESP_BROWNOUT_DET_LVL_SEL

Brownout voltage level

Found in: Component config > ESP System Settings > Brownout Detector > CONFIG_ESP_BROWNOUT_DET

The brownout detector will reset the chip when the supply voltage is approximately below this level. Note that there may be some variation of brownout voltage level between each chip.

Please note that this config is only valid when P4 SOC version is above v1.0. When you are using a earlier P4 SOC version (v0.x), the brownout value should be fixed around 2.52V and not configurable.

#The voltage levels here are estimates, more work needs to be done to figure out the exact voltages #of the brownout threshold levels.

Available options:

- 2.6V (CONFIG_ESP_BROWNOUT_DET_LVL_SEL_7)
- 2.52V (CONFIG_ESP_BROWNOUT_DET_LVL_SEL_6)

- 2.42V (CONFIG_ESP_BROWNOUT_DET_LVL_SEL_5)

CONFIG_ESP_SYSTEM_HW_STACK_GUARD

Hardware stack guard

Found in: Component config > ESP System Settings

This config allows to trigger a panic interrupt when Stack Pointer register goes out of allocated stack memory bounds.

Default value:

- Yes (enabled)

CONFIG_ESP_SYSTEM_HW_PC_RECORD

Hardware PC recording

Found in: Component config > ESP System Settings

This option will enable the PC recording function of assist_debug module. The PC value of the CPU will be recorded to PC record register in assist_debug module in real time. When an exception occurs and the CPU is reset, this register will be kept, then we can use the recorded PC to debug the causes of the reset.

Default value:

- Yes (enabled)

IPC (Inter-Processor Call)

Contains:

- *CONFIG_ESP_IPC_TASK_STACK_SIZE*
- *CONFIG_ESP_IPC_USES_CALLERS_PRIORITY*

CONFIG_ESP_IPC_TASK_STACK_SIZE

Inter-Processor Call (IPC) task stack size

Found in: Component config > IPC (Inter-Processor Call)

Configure the IPC tasks stack size. An IPC task runs on each core (in dual core mode), and allows for cross-core function calls. See IPC documentation for more details. The default IPC stack size should be enough for most common simple use cases. However, users can increase/decrease the stack size to their needs.

Range:

- from 512 to 65536

Default value:

- 1024

CONFIG_ESP_IPC_USES_CALLERS_PRIORITY

IPC runs at caller's priority

Found in: Component config > IPC (Inter-Processor Call)

If this option is not enabled then the IPC task will keep behavior same as prior to that of ESP-IDF v4.0, hence IPC task will run at (configMAX_PRIORITIES - 1) priority.

ESP Timer (High Resolution Timer) Contains:

- *CONFIG_ESP_TIMER_PROFILING*
- *CONFIG_ESP_TIMER_TASK_AFFINITY*
- *CONFIG_ESP_TIMER_TASK_STACK_SIZE*
- *CONFIG_ESP_TIMER_INTERRUPT_LEVEL*
- *CONFIG_ESP_TIMER_SHOW_EXPERIMENTAL*
- *CONFIG_ESP_TIMER_SUPPORTS_ISR_DISPATCH_METHOD*
- *CONFIG_ESP_TIMER_ISR_AFFINITY*

CONFIG_ESP_TIMER_PROFILING

Enable esp_timer profiling features

Found in: Component config > ESP Timer (High Resolution Timer)

If enabled, esp_timer_dump will dump information such as number of times the timer was started, number of times the timer has triggered, and the total time it took for the callback to run. This option has some effect on timer performance and the amount of memory used for timer storage, and should only be used for debugging/testing purposes.

Default value:

- No (disabled)

CONFIG_ESP_TIMER_TASK_STACK_SIZE

High-resolution timer task stack size

Found in: Component config > ESP Timer (High Resolution Timer)

Configure the stack size of ”timer_task” task. This task is used to dispatch callbacks of timers created using ets_timer and esp_timer APIs. If you are seing stack overflow errors in timer task, increase this value.

Note that this is not the same as FreeRTOS timer task. To configure FreeRTOS timer task size, see ”FreeRTOS timer task stack size” option in ”FreeRTOS”.

Range:

- from 2048 to 65536

Default value:

- 3584

CONFIG_ESP_TIMER_INTERRUPT_LEVEL

Interrupt level

Found in: Component config > ESP Timer (High Resolution Timer)

This sets the interrupt priority level for esp_timer ISR. A higher value reduces interrupt latency by minimizing the timer processing delay.

Range:

- from 1 to 1

Default value:

- 1

CONFIG_ESP_TIMER_SHOW_EXPERIMENTAL

show esp_timer’s experimental features

Found in: Component config > ESP Timer (High Resolution Timer)

This shows some hidden features of esp_timer. Note that they may break other features, use them with care.

CONFIG_ESP_TIMER_TASK_AFFINITY

esp_timer task core affinity

Found in: Component config > ESP Timer (High Resolution Timer)

The default settings: timer TASK on CPU0 and timer ISR on CPU0. Other settings may help in certain cases, but note that they may break other features, use them with care. - ”CPU0”: (default) esp_timer task is processed by CPU0. - ”CPU1”: esp_timer task is processed by CPU1. - ”No affinity”: esp_timer task can be processed by any CPU.

Available options:

- CPU0 (CONFIG_ESP_TIMER_TASK_AFFINITY_CPU0)
- CPU1 (CONFIG_ESP_TIMER_TASK_AFFINITY_CPU1)
- No affinity (CONFIG_ESP_TIMER_TASK_AFFINITY_NO_AFFINITY)

CONFIG_ESP_TIMER_ISR_AFFINITY

timer interrupt core affinity

Found in: Component config > ESP Timer (High Resolution Timer)

The default settings: timer TASK on CPU0 and timer ISR on CPU0. Other settings may help in certain cases, but note that they may break other features, use them with care. - ”CPU0”: (default) timer interrupt is processed by CPU0. - ”CPU1”: timer interrupt is processed by CPU1. - ”No affinity”: timer interrupt can be processed by any CPU. It helps to reduce latency but there is a disadvantage it leads to the timer ISR running on every core. It increases the CPU time usage for timer ISRs by N on an N-core system.

Available options:

- CPU0 (CONFIG_ESP_TIMER_ISR_AFFINITY_CPU0)
- CPU1 (CONFIG_ESP_TIMER_ISR_AFFINITY_CPU1)
- No affinity (CONFIG_ESP_TIMER_ISR_AFFINITY_NO_AFFINITY)

CONFIG_ESP_TIMER_SUPPORTS_ISR_DISPATCH_METHOD

Support ISR dispatch method

Found in: Component config > ESP Timer (High Resolution Timer)

Allows using ESP_TIMER_ISR dispatch method (ESP_TIMER_TASK dispatch method is also available). - ESP_TIMER_TASK - Timer callbacks are dispatched from a high-priority esp_timer task. - ESP_TIMER_ISR - Timer callbacks are dispatched directly from the timer interrupt handler. The ISR dispatch can be used, in some cases, when a callback is very simple or need a lower-latency.

Default value:

- No (disabled)

Wi-Fi

Contains:

- *CONFIG_ESP_WIFI_TESTING_OPTIONS*
- *CONFIG_ESP_WIFI_WPS_SOFTAP_REGISTRAR*
- *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP*
- *CONFIG_ESP_WIFI_11KV_SUPPORT*
- *CONFIG_ESP_WIFI_11R_SUPPORT*
- *CONFIG_ESP_WIFI_DPP_SUPPORT*
- *CONFIG_ESP_WIFI_ENTERPRISE_SUPPORT*
- *CONFIG_ESP_WIFI_MBO_SUPPORT*

- *CONFIG_ESP_WIFI_SUITE_B_192*
- *CONFIG_ESP_WIFI_ENABLE_WPA3_OWE_STA*
- *CONFIG_ESP_WIFI_WAPI_PSK*
- *CONFIG_ESP_WIFI_ENABLE_DUMP_CTRL_BFRP*
- *CONFIG_ESP_WIFI_ENABLE_DUMP_HESIGB*
- *CONFIG_ESP_WIFI_ENABLE_DUMP_MU_CFO*
- *CONFIG_ESP_WIFI_ENABLE_DUMP_CTRL_NDPA*
- *CONFIG_ESP_WIFI_ENABLE_WIFI_RX_STATS*
- *CONFIG_ESP_WIFI_ENABLE_WIFI_TX_STATS*
- *CONFIG_ESP_WIFI_ENABLE_WPA3_SAE*
- *CONFIG_ESP_HOST_WIFI_ENABLED*
- *CONFIG_ESP_WIFI_SOFTAP_BEACON_MAX_LEN*
- *CONFIG_ESP_WIFI_CACHE_TX_BUFFER_NUM*
- *CONFIG_ESP_WIFI_DYNAMIC_RX_BUFFER_NUM*
- *CONFIG_ESP_WIFI_DYNAMIC_TX_BUFFER_NUM*
- *CONFIG_ESP_WIFI_RX_MGMT_BUF_NUM_DEF*
- *CONFIG_ESP_WIFI_STATIC_RX_BUFFER_NUM*
- *CONFIG_ESP_WIFI_STATIC_TX_BUFFER_NUM*
- *CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM*
- *CONFIG_ESP_WIFI_SLP_DEFAULT_MAX_ACTIVE_TIME*
- *CONFIG_ESP_WIFI_SLP_DEFAULT_MIN_ACTIVE_TIME*
- *CONFIG_ESP_WIFI_SLP_DEFAULT_WAIT_BROADCAST_DATA_TIME*
- *CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE*
- *CONFIG_ESP_WIFI_DEBUG_PRINT*
- *CONFIG_ESP_WIFI_MGMT_RX_BUFFER*
- *CONFIG_ESP_WIFI_TX_BUFFER*
- *CONFIG_ESP_WIFI_MBEDTLS_CRYPTO*
- *CONFIG_ESP_WIFI_AMPDU_RX_ENABLED*
- *CONFIG_ESP_WIFI_AMPDU_TX_ENABLED*
- *CONFIG_ESP_WIFI_AMSDU_TX_ENABLED*
- *CONFIG_ESP_WIFI_NAN_ENABLE*
- *CONFIG_ESP_WIFI_CSI_ENABLED*
- *CONFIG_ESP_WIFI_EXTRA_IRAM_OPT*
- *CONFIG_ESP_WIFI_FTM_ENABLE*
- *CONFIG_ESP_WIFI_GCMP_SUPPORT*
- *CONFIG_ESP_WIFI_GMAC_SUPPORT*
- *CONFIG_ESP_WIFI_IRAM_OPT*
- *CONFIG_ESP_WIFI_MGMT_SBUF_NUM*
- *CONFIG_ESP_WIFI_ENHANCED_LIGHT_SLEEP*
- *CONFIG_ESP_WIFI_NVS_ENABLED*
- *CONFIG_ESP_WIFI_RX_IRAM_OPT*
- *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*
- *CONFIG_ESP_WIFI_SLP_IRAM_OPT*
- *CONFIG_ESP_WIFI_SOFTAP_SUPPORT*
- *CONFIG_ESP_WIFI_TASK_CORE_ID*
- *CONFIG_ESP_WIFI_TX_HETB_QUEUE_NUM*
- *WPS Configuration Options*

CONFIG_ESP_HOST_WIFI_ENABLED

Host WiFi Enable

Found in: Component config > Wi-Fi

Default value:

- No (disabled)

CONFIG_ESP_WIFI_STATIC_RX_BUFFER_NUM

Max number of WiFi static RX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi static RX buffers. Each buffer takes approximately 1.6KB of RAM. The static rx buffers are allocated when esp_wifi_init is called, they are not freed until esp_wifi_deinit is called.

WiFi hardware use these buffers to receive all 802.11 frames. A higher number may allow higher throughput but increases memory use. If ESP_WIFI_AMPDU_RX_ENABLED is enabled, this value is recommended to set equal or bigger than ESP_WIFI_RX_BA_WIN in order to achieve better throughput and compatibility with both stations and APs.

Range:

- from 2 to 128 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 16 if *CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_DYNAMIC_RX_BUFFER_NUM

Max number of WiFi dynamic RX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi dynamic RX buffers, 0 means unlimited RX buffers will be allocated (provided sufficient free RAM). The size of each dynamic RX buffer depends on the size of the received data frame.

For each received data frame, the WiFi driver makes a copy to an RX buffer and then delivers it to the high layer TCP/IP stack. The dynamic RX buffer is freed after the higher layer has successfully received the data frame.

For some applications, WiFi data frames may be received faster than the application can process them. In these cases we may run out of memory if RX buffer number is unlimited (0).

If a dynamic RX buffer limit is set, it should be at least the number of static RX buffers.

Range:

- from 0 to 1024 if *CONFIG_LWIP_WND_SCALE* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 32 if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_TX_BUFFER

Type of WiFi TX buffers

Found in: Component config > Wi-Fi

Select type of WiFi TX buffers:

If ”Static” is selected, WiFi TX buffers are allocated when WiFi is initialized and released when WiFi is de-initialized. The size of each static TX buffer is fixed to about 1.6KB.

If ”Dynamic” is selected, each WiFi TX buffer is allocated as needed when a data frame is delivered to the Wifi driver from the TCP/IP stack. The buffer is freed after the data frame has been sent by the WiFi driver. The size of each dynamic TX buffer depends on the length of each data frame sent by the TCP/IP layer.

If PSRAM is enabled, ”Static” should be selected to guarantee enough WiFi TX buffers. If PSRAM is disabled, ”Dynamic” should be selected to improve the utilization of RAM.

Available options:

- Static (CONFIG_ESP_WIFI_STATIC_TX_BUFFER)
- Dynamic (CONFIG_ESP_WIFI_DYNAMIC_TX_BUFFER)

CONFIG_ESP_WIFI_STATIC_TX_BUFFER_NUM

Max number of WiFi static TX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi static TX buffers. Each buffer takes approximately 1.6KB of RAM. The static RX buffers are allocated when esp_wifi_init() is called, they are not released until esp_wifi_deinit() is called.

For each transmitted data frame from the higher layer TCP/IP stack, the WiFi driver makes a copy of it in a TX buffer. For some applications especially UDP applications, the upper layer can deliver frames faster than WiFi layer can transmit. In these cases, we may run out of TX buffers.

Range:

- from 1 to 64 if *CONFIG_ESP_WIFI_STATIC_TX_BUFFER* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 16 if *CONFIG_ESP_WIFI_STATIC_TX_BUFFER* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_CACHE_TX_BUFFER_NUM

Max number of WiFi cache TX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi cache TX buffer number.

For each TX packet from uplayer, such as LWIP etc, WiFi driver needs to allocate a static TX buffer and makes a copy of uplayer packet. If WiFi driver fails to allocate the static TX buffer, it caches the uplayer packets to a dedicated buffer queue, this option is used to configure the size of the cached TX queue.

Range:

- from 16 to 128 if *CONFIG_SPIRAM* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 32 if *CONFIG_SPIRAM* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_DYNAMIC_TX_BUFFER_NUM

Max number of WiFi dynamic TX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi dynamic TX buffers. The size of each dynamic TX buffer is not fixed, it depends on the size of each transmitted data frame.

For each transmitted frame from the higher layer TCP/IP stack, the WiFi driver makes a copy of it in a TX buffer. For some applications, especially UDP applications, the upper layer can deliver frames faster than WiFi layer can transmit. In these cases, we may run out of TX buffers.

Range:

- from 1 to 128 if *CONFIG_ESP_WIFI_DYNAMIC_TX_BUFFER* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 32 if *CONFIG_ESP_WIFI_DYNAMIC_TX_BUFFER* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_MGMT_RX_BUFFER

Type of WiFi RX MGMT buffers

Found in: Component config > Wi-Fi

Select type of WiFi RX MGMT buffers:

If "Static" is selected, WiFi RX MGMT buffers are allocated when WiFi is initialized and released when WiFi is de-initialized. The size of each static RX MGMT buffer is fixed to about 500 Bytes.

If "Dynamic" is selected, each WiFi RX MGMT buffer is allocated as needed when a MGMT data frame is received. The MGMT buffer is freed after the MGMT data frame has been processed by the WiFi driver.

Available options:

- Static (CONFIG_ESP_WIFI_STATIC_RX_MGMT_BUFFER)
- Dynamic (CONFIG_ESP_WIFI_DYNAMIC_RX_MGMT_BUFFER)

CONFIG_ESP_WIFI_RX_MGMT_BUF_NUM_DEF

Max number of WiFi RX MGMT buffers

Found in: Component config > Wi-Fi

Set the number of WiFi RX_MGMT buffers.

For Management buffers, the number of dynamic and static management buffers is the same. In order to prevent memory fragmentation, the management buffer type should be set to static first.

Range:
- from 1 to 10 if *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:
- 5 if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_CSI_ENABLED

WiFi CSI(Channel State Information)

Found in: Component config > Wi-Fi

Select this option to enable CSI(Channel State Information) feature. CSI takes about CONFIG_ESP_WIFI_STATIC_RX_BUFFER_NUM KB of RAM. If CSI is not used, it is better to disable this feature in order to save memory.

Default value:
- No (disabled) if SOC_WIFI_CSI_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_AMPDU_TX_ENABLED

WiFi AMPDU TX

Found in: Component config > Wi-Fi

Select this option to enable AMPDU TX feature

Default value:
- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_TX_BA_WIN

WiFi AMPDU TX BA window size

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_AMPDU_TX_ENABLED

Set the size of WiFi Block Ack TX window. Generally a bigger value means higher throughput but more memory. Most of time we should NOT change the default value unless special reason, e.g. test the maximum UDP TX throughput with iperf etc. For iperf test in shieldbox, the recommended value is 9~12.

Range:

- from 2 to 64 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_WIFI_AMPDU_TX_ENABLED* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 6 if *CONFIG_ESP_WIFI_AMPDU_TX_ENABLED* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_AMPDU_RX_ENABLED

WiFi AMPDU RX

Found in: Component config > Wi-Fi

Select this option to enable AMPDU RX feature

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_RX_BA_WIN

WiFi AMPDU RX BA window size

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_AMPDU_RX_ENABLED

Set the size of WiFi Block Ack RX window. Generally a bigger value means higher throughput and better compatibility but more memory. Most of time we should NOT change the default value unless special reason, e.g. test the maximum UDP RX throughput with iperf etc. For iperf test in shieldbox, the recommended value is 9~12. If PSRAM is used and WiFi memory is preferred to allocate in PSRAM first, the default and minimum value should be 16 to achieve better throughput and compatibility with both stations and APs.

Range:

- from 2 to 64 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_WIFI_AMPDU_RX_ENABLED* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 16 if *CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP* && *CONFIG_ESP_WIFI_AMPDU_RX_ENABLED* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_AMSDU_TX_ENABLED

WiFi AMSDU TX

Found in: Component config > Wi-Fi

Select this option to enable AMSDU TX feature

Default value:

- No (disabled) if *CONFIG_SPIRAM* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_NVS_ENABLED

WiFi NVS flash

Found in: Component config > Wi-Fi

Select this option to enable WiFi NVS flash

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_TASK_CORE_ID

WiFi Task Core ID

Found in: Component config > Wi-Fi

Pinned WiFi task to core 0 or core 1.

Available options:

- Core 0 (CONFIG_ESP_WIFI_TASK_PINNED_TO_CORE_0)
- Core 1 (CONFIG_ESP_WIFI_TASK_PINNED_TO_CORE_1)

CONFIG_ESP_WIFI_SOFTAP_BEACON_MAX_LEN

Max length of WiFi SoftAP Beacon

Found in: Component config > Wi-Fi

ESP-MESH utilizes beacon frames to detect and resolve root node conflicts (see documentation). However the default length of a beacon frame can simultaneously hold only five root node identifier structures, meaning that a root node conflict of up to five nodes can be detected at one time. In the occurrence of more root nodes conflict involving more than five root nodes, the conflict resolution process will detect five of the root nodes, resolve the conflict, and re-detect more root nodes. This process will repeat until all root node conflicts are resolved. However this process can generally take a very long time.

To counter this situation, the beacon frame length can be increased such that more root nodes can be detected simultaneously. Each additional root node will require 36 bytes and should be added on top of the default beacon frame length of 752 bytes. For example, if you want to detect 10 root nodes simultaneously, you need to set the beacon frame length as 932 (752+36*5).

Setting a longer beacon length also assists with debugging as the conflicting root nodes can be identified more quickly.

Range:
- from 752 to 1256 if *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:
- 752 if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_MGMT_SBUF_NUM

WiFi mgmt short buffer number

Found in: Component config > Wi-Fi

Set the maximum number of Wi-Fi management short buffers. These buffers are dynamically allocated, with their size determined by the length of the management packet to be sent. When a management packet is less than 64 bytes, the Wi-Fi driver classifies it as a short management packet and assigns it to one of these buffers.

Range:
- from 6 to 32 if *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:
- 32 if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_IRAM_OPT

WiFi IRAM speed optimization

Found in: Component config > Wi-Fi

Select this option to place frequently called Wi-Fi library functions in IRAM. When this option is disabled, more than 10Kbytes of IRAM memory will be saved but Wi-Fi throughput will be reduced.

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_EXTRA_IRAM_OPT

WiFi EXTRA IRAM speed optimization

Found in: Component config > Wi-Fi

Select this option to place additional frequently called Wi-Fi library functions in IRAM. When this option is disabled, more than 5Kbytes of IRAM memory will be saved but Wi-Fi throughput will be reduced.

Default value:

- Yes (enabled) if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*
- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_RX_IRAM_OPT

WiFi RX IRAM speed optimization

Found in: Component config > Wi-Fi

Select this option to place frequently called Wi-Fi library RX functions in IRAM. When this option is disabled, more than 17Kbytes of IRAM memory will be saved but Wi-Fi performance will be reduced.

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_WPA3_SAE

Enable WPA3-Personal

Found in: Component config > Wi-Fi

Select this option to allow the device to establish a WPA3-Personal connection with eligible AP's. PMF (Protected Management Frames) is a prerequisite feature for a WPA3 connection, it needs to be explicitly configured before attempting connection. Please refer to the Wi-Fi Driver API Guide for details.

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_SAE_PK

Enable SAE-PK

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_WPA3_SAE

Select this option to enable SAE-PK

Default value:

- Yes (enabled) if *CONFIG_ESP_WIFI_ENABLE_WPA3_SAE* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SOFTAP_SAE_SUPPORT

Enable WPA3 Personal(SAE) SoftAP

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_WPA3_SAE

Select this option to enable SAE support in softAP mode.

Default value:

- Yes (enabled) if *CONFIG_ESP_WIFI_ENABLE_WPA3_SAE* && *CONFIG_ESP_WIFI_SOFTAP_SUPPORT* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_WPA3_OWE_STA

Enable OWE STA

Found in: Component config > Wi-Fi

Select this option to allow the device to establish OWE connection with eligible AP's. PMF (Protected Management Frames) is a prerequisite feature for a WPA3 connection, it needs to be explicitly configured before attempting connection. Please refer to the Wi-Fi Driver API Guide for details.

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SLP_IRAM_OPT

WiFi SLP IRAM speed optimization

Found in: Component config > Wi-Fi

Select this option to place called Wi-Fi library TBTT process and receive beacon functions in IRAM. Some functions can be put in IRAM either by ESP_WIFI_IRAM_OPT and ESP_WIFI_RX_IRAM_OPT, or this one. If already enabled ESP_WIFI_IRAM_OPT, the other 7.3KB IRAM memory would be taken by this option. If already enabled ESP_WIFI_RX_IRAM_OPT, the other 1.3KB IRAM memory would be taken by this option. If neither of them are enabled, the other 7.4KB IRAM memory would be taken by this option. Wi-Fi power-save mode average current would be reduced if this option is enabled.

Default value:

- Yes (enabled) if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SLP_DEFAULT_MIN_ACTIVE_TIME

Minimum active time

Found in: Component config > Wi-Fi

Only for station in WIFI_PS_MIN_MODEM or WIFI_PS_MAX_MODEM. When the station enters the active state, it will work for at least ESP_WIFI_SLP_DEFAULT_MIN_ACTIVE_TIME. If a data packet is received or sent during this period, the time will be refreshed. If the time is up, but the station still has packets to receive or send, the time will also be refreshed. unit: milliseconds.

Range:

- from 8 to 60 if *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 50 if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SLP_DEFAULT_MAX_ACTIVE_TIME

Maximum keep alive time

Found in: Component config > Wi-Fi

Only for station in WIFI_PS_MIN_MODEM or WIFI_PS_MAX_MODEM. If no packet has been sent within ESP_WIFI_SLP_DEFAULT_MAX_ACTIVE_TIME, a null data packet will be sent to maintain the connection with the AP. unit: seconds.

Range:

- from 10 to 60 if *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 10 if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SLP_DEFAULT_WAIT_BROADCAST_DATA_TIME

Minimum wait broadcast data time

Found in: Component config > Wi-Fi

Only for station in WIFI_PS_MIN_MODEM or WIFI_PS_MAX_MODEM. When the station knows through the beacon that AP will send broadcast packet, it will wait for ESP_WIFI_SLP_DEFAULT_WAIT_BROADCAST_DATA_TIME before entering the sleep process. If a broadcast packet is received with more data bits, the time will refreshed. unit: milliseconds.

Range:
- from 10 to 30 if *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:
- 15 if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_FTM_ENABLE

WiFi FTM

Found in: Component config > Wi-Fi

Enable feature Fine Timing Measurement for calculating WiFi Round-Trip-Time (RTT).

Default value:
- No (disabled) if SOC_WIFI_FTM_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_FTM_INITIATOR_SUPPORT

FTM Initiator support

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_FTM_ENABLE

Default value:
- Yes (enabled) if *CONFIG_ESP_WIFI_FTM_ENABLE* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_FTM_RESPONDER_SUPPORT

FTM Responder support

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_FTM_ENABLE

Default value:
- Yes (enabled) if *CONFIG_ESP_WIFI_FTM_ENABLE* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE

Power Management for station at disconnected

Found in: Component config > Wi-Fi

Select this option to enable power_management for station when disconnected. Chip will do modem-sleep when rf module is not in use any more.

Default value:
- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_GCMP_SUPPORT

WiFi GCMP Support(GCMP128 and GCMP256)

Found in: Component config > Wi-Fi

Select this option to enable GCMP support. GCMP support is compulsory for WiFi Suite-B support.

Default value:

- No (disabled) if SOC_WIFI_GCMP_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_GMAC_SUPPORT

WiFi GMAC Support(GMAC128 and GMAC256)

Found in: Component config > Wi-Fi

Select this option to enable GMAC support. GMAC support is compulsory for WiFi 192 bit certification.

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SOFTAP_SUPPORT

WiFi SoftAP Support

Found in: Component config > Wi-Fi

WiFi module can be compiled without SoftAP to save code size.

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENHANCED_LIGHT_SLEEP

WiFi modem automatically receives the beacon

Found in: Component config > Wi-Fi

The wifi modem automatically receives the beacon frame during light sleep.

Default value:

- No (disabled) if ESP_PHY_MAC_BB_PD && SOC_PM_SUPPORT_BEACON_WAKEUP && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT

Wifi sleep optimize when beacon lost

Found in: Component config > Wi-Fi

Enable wifi sleep optimization when beacon loss occurs and immediately enter sleep mode when the WiFi module detects beacon loss.

CONFIG_ESP_WIFI_SLP_BEACON_LOST_TIMEOUT

Beacon loss timeout

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT

Timeout time for close rf phy when beacon loss occurs, Unit: 1024 microsecond.

Range:

- from 5 to 100 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 10 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SLP_BEACON_LOST_THRESHOLD

Maximum number of consecutive lost beacons allowed

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT

Maximum number of consecutive lost beacons allowed, WiFi keeps Rx state when the number of consecutive beacons lost is greater than the given threshold.

Range:

- from 0 to 8 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 3 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SLP_PHY_ON_DELTA_EARLY_TIME

Delta early time for RF PHY on

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT

Delta early time for rf phy on, When the beacon is lost, the next rf phy on will be earlier the time specified by the configuration item, Unit: 32 microsecond.

Range:

- from 0 to 100 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT* && SOC_WIFI_SUPPORT_VARIABLE_BEACON_WINDOW && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 2 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT* && SOC_WIFI_SUPPORT_VARIABLE_BEACON_WINDOW && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SLP_PHY_OFF_DELTA_TIMEOUT_TIME

Delta timeout time for RF PHY off

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT

Delta timeout time for rf phy off, When the beacon is lost, the next rf phy off will be delayed for the time specified by the configuration item. Unit: 1024 microsecond.

Range:

- from 0 to 8 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT* && SOC_WIFI_SUPPORT_VARIABLE_BEACON_WINDOW && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 2 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT* && SOC_WIFI_SUPPORT_VARIABLE_BEACON_WINDOW && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM

Maximum espnow encrypt peers number

Found in: Component config > Wi-Fi

Maximum number of encrypted peers supported by espnow. The number of hardware keys for encryption is fixed. And the espnow and SoftAP share the same hardware keys. So this configuration will affect the maximum connection number of SoftAP. Maximum espnow encrypted peers number + maximum number of connections of SoftAP = Max hardware keys number. When using ESP mesh, this value should be set to a maximum of 6.

Range:

- from 0 to 17 if *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 7 if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_NAN_ENABLE

WiFi Aware

Found in: Component config > Wi-Fi

Enable WiFi Aware (NAN) feature.

Default value:

- No (disabled) if SOC_WIFI_NAN_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_MBEDTLS_CRYPTO

Use MbedTLS crypto APIs

Found in: Component config > Wi-Fi

Select this option to enable the use of MbedTLS crypto APIs. The internal crypto support within the supplicant is limited and may not suffice for all new security features, including WPA3.

It is recommended to always keep this option enabled. Additionally, note that MbedTLS can leverage hardware acceleration if available, resulting in significantly faster cryptographic operations.

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_MBEDTLS_TLS_CLIENT

Use MbedTLS TLS client for WiFi Enterprise connection

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_MBEDTLS_CRYPTO

Select this option to use MbedTLS TLS client for WPA2 enterprise connection. Please note that from MbedTLS-3.0 onwards, MbedTLS does not support SSL-3.0 TLS-v1.0, TLS-v1.1 versions. In case your server is using one of these version, it is advisable to update your server. Please disable this option for compatibility with older TLS versions.

Default value:

- Yes (enabled) if *CONFIG_ESP_WIFI_ENTERPRISE_SUPPORT* && *CONFIG_ESP_WIFI_MBEDTLS_CRYPTO* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_EAP_TLS1_3

Enable EAP-TLS v1.3 Support for WiFi Enterprise connection

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_MBEDTLS_CRYPTO > CONFIG_ESP_WIFI_MBEDTLS_TLS_CLIENT

Select this option to support EAP with TLS v1.3. This configuration still supports compatibility with EAP-TLS v1.2. Please note that enabling this configuration will cause every application which uses TLS go for TLS1.3 if server supports that. TLS1.3 is still in development in mbedtls and there may be interoperability issues with this. Please modify your application to set max version as TLS1.2 if you want to enable TLS1.3 only for WiFi connection.

Default value:

- No (disabled) if *CONFIG_ESP_WIFI_MBEDTLS_TLS_CLIENT* && *CONFIG_IDF_EXPERIMENTAL_FEATURES* && *CONFIG_ESP_WIFI_MBEDTLS_CRYPTO* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_WAPI_PSK

Enable WAPI PSK support

Found in: Component config > Wi-Fi

Select this option to enable WAPI-PSK which is a Chinese National Standard Encryption for Wireless LANs (GB 15629.11-2003).

Default value:

- No (disabled) if SOC_WIFI_WAPI_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SUITE_B_192

Enable NSA suite B support with 192 bit key

Found in: Component config > Wi-Fi

Select this option to enable 192 bit NSA suite-B. This is necessary to support WPA3 192 bit security.

Default value:

- No (disabled) if SOC_WIFI_GCMP_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_11KV_SUPPORT

Enable 802.11k, 802.11v APIs Support

Found in: Component config > Wi-Fi

Select this option to enable 802.11k 802.11v APIs(RRM and BTM support). Only APIs which are helpful for network assisted roaming are supported for now. Enable this option with BTM and RRM enabled in sta config to make device ready for network assisted roaming. BTM: BSS transition management enables an AP to request a station to transition to a specific AP, or to indicate to a station a set of preferred APs. RRM: Radio measurements enable STAs to understand the radio environment, it enables STAs to observe and gather data on radio link performance and on the radio environment. Current implementation adds beacon report, link measurement, neighbor report.

Default value:

- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_SCAN_CACHE

Keep scan results in cache

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_11KV_SUPPORT

Keep scan results in cache, if not enabled, those will be flushed immediately.

Default value:

- No (disabled) if *CONFIG_ESP_WIFI_11KV_SUPPORT* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_MBO_SUPPORT

Enable Multi Band Operation Certification Support

Found in: Component config > Wi-Fi

Select this option to enable WiFi Multiband operation certification support.

Default value:

- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_ROAMING_APP

Advanced support for Wi-Fi Roaming (Experimental)

Found in: Component config > Wi-Fi

Enable Espressif's roaming app to allow for efficient Wi-Fi roaming. This includes configurable periodic environment scans, maintaining a cache of the best APs, handling low rssi events etc.

Risk Warning Please note that this feature is still experimental and enabling this potentially can lead to unpredictable scanning, connection and roaming attempts. We are still working on tuning and optimising this feature to ensure reliable and stable use.

Default value:

- No (disabled) if *CONFIG_IDF_EXPERIMENTAL_FEATURES* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Configure roaming App Contains:

- *CONFIG_ESP_WIFI_ROAMING_BACKOFF_TIME*
- *Roaming Methods*
- *Roaming triggers*
- *Scan Configuration*
- *CONFIG_ESP_WIFI_ROAMING_PERIODIC_RRM_MONITORING*

Roaming triggers Contains:

- *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR*
- *CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_ROAMING*

CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_ROAMING

Use Low RSSI to trigger roaming.

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App > Roaming triggers

Enable to use a RSSI threshold to trigger roaming.

Default value:

- Yes (enabled) if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_THRESHOLD

WiFi RSSI threshold to trigger roaming

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App > Roaming triggers > CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_ROAMING

WiFi RSSI threshold to trigger roaming value in dBm (-99 to -1). Values under -30 dbm might lead to a flood of low rssi events. This interferes with normal functioning and TX/Rx performance.

Range:

- from -99 to -30 if *CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_ROAMING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- "-60" if *CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_ROAMING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_OFFSET

Offset by which to reset the RSSI Threshold after attempt to roam.

Found in: Component config > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Roaming triggers* > *CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_ROAMING*

Decide the offset by which to decrease the Low RSSI threshold set by ESP_WIFI_ROAMING_LOW_RSSI_THRESHOLD after each failed attempt to roam. This allows for the station to keep scanning for better AP's after the Low RSSI threshold is reached in a stepped manner, rather than only attempting to roam the first time the current AP's RSSI breaches the set RSSI threshold. Setting 0 here may cause station to be flooded with low rssi events, therefore that's not recommended to be kept.

Range:

- from 0 to 99 if *CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_ROAMING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 5 if *CONFIG_ESP_WIFI_ROAMING_LOW_RSSI_ROAMING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR

Conduct periodic scans to check if a better AP is available

Found in: Component config > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Roaming triggers*

Conduct periodic scans periodically to check if a better AP is available.

Default value:

- Yes (enabled) if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_THRESHOLD

Threshold at which to begin periodic scanning for a better AP.

Found in: Component config > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Roaming triggers* > *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR*

Threshold at which the station will begin scanning to find an AP with better RSSI.

Range:

- from -99 to -1 if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- "-50" if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_SCAN_MONITOR_INTERVAL

Time intervals (in seconds) at which station will initiate a scan

Found in: *Component config* > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Roaming triggers* > *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR*

Intervals at which station will periodically scan to check if better AP is available

Range:

- from 1 to 1500 if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 30 if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_SCAN_ROAM_RSSI_DIFF

RSSI difference b/w current AP and candidate AP to initiate connection

Found in: *Component config* > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Roaming triggers* > *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR*

Minimum RSSI difference b/w current AP and a potential roaming candidate AP to trigger a roaming attempt.

Range:

- from 0 to 99 if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 15 if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_SCAN_MONITOR* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Roaming Methods Contains:

- *CONFIG_ESP_WIFI_ROAMING_LEGACY_ROAMING*
- *CONFIG_ESP_WIFI_ROAMING_NETWORK_ASSISTED_ROAM*

CONFIG_ESP_WIFI_ROAMING_LEGACY_ROAMING

Support Legacy roaming approach

Found in: *Component config* > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Roaming Methods*

Roaming between APs that do not support 802.11kv. This will allow station to roam even when connection is not BTM supported, by forcefully disconnecting from current AP and connecting to better AP.

Default value:

- Yes (enabled) if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_NETWORK_ASSISTED_ROAM

Support Network Assisted roaming using 802.11kv

Found in: *Component config* > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Roaming Methods*

Roaming between APs using network assisted Roaming. This involves BSS Transition Management mechanisms outlined in 802.11v. Note that this moves the responsibility to the AP's network, and hence isn't guaranteed to cause the station to attempt to roam each time.

Default value:

- Yes (enabled) if *CONFIG_ESP_WIFI_11KV_SUPPORT* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_NETWORK_ASSISTED_ROAMING_RETRY_COUNT

Retry count after which to switch to legacy roaming

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App > Roaming Methods > CONFIG_ESP_WIFI_ROAMING_NETWORK_ASSISTED_ROAM

Retry threshold after which the station should stop using Network Assisted roaming methods and start using legacy roaming instead.

Range:

- from 1 to 5 if *CONFIG_ESP_WIFI_ROAMING_NETWORK_ASSISTED_ROAM* && *CONFIG_ESP_WIFI_ROAMING_LEGACY_ROAMING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 2 if *CONFIG_ESP_WIFI_ROAMING_NETWORK_ASSISTED_ROAM* && *CONFIG_ESP_WIFI_ROAMING_LEGACY_ROAMING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Scan Configuration Contains:

- *CONFIG_ESP_WIFI_ROAMING_HOME_CHANNEL_DWELL_TIME*
- *CONFIG_ESP_WIFI_ROAMING_MAX_CANDIDATES*
- *CONFIG_ESP_WIFI_ROAMING_SCAN_MAX_SCAN_TIME*
- *CONFIG_ESP_WIFI_ROAMING_SCAN_MIN_SCAN_TIME*
- *CONFIG_ESP_WIFI_ROAMING_SCAN_CHAN_LIST*
- *CONFIG_ESP_WIFI_ROAMING_SCAN_EXPIRY_WINDOW*

CONFIG_ESP_WIFI_ROAMING_SCAN_MIN_SCAN_TIME

Minimum duration (in milliseconds) of station's per channel active scan

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App > Scan Configuration

Minimum duration of active scanning per channel in milliseconds.

Range:

- from 0 to 120 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 10 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_SCAN_MAX_SCAN_TIME

Maximum duration (in milliseconds) of station's per channel active scan time

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App > Scan Configuration

Maximum duration of active scanning per channel in milliseconds.

Range:

- from 30 to 120 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 70 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_HOME_CHANNEL_DWELL_TIME

Home channel dwell time scanning between consecutive channels

Found in: *Component config* > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Scan Configuration*

If connected, duration for which the station will return to it's home channel for Tx/Rx of frames stored in buffers between scanning on consecutive channels.

Range:

- from 30 to 150 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 30 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_SCAN_CHAN_LIST

Preferred channel list for scanning

Found in: *Component config* > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Scan Configuration*

Channels your wireless network operates on to allow for faster scanning. Specify the channels(between 1-14) in a comma separated manner.

Default value:

- "None" if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_SCAN_EXPIRY_WINDOW

Scan results expiry window (in seconds)

Found in: *Component config* > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Scan Configuration*

Duration for which the results from the most recent scans can be used by the roaming app for determining the roaming candidates.

Range:

- from 5 to 20 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 10 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_MAX_CANDIDATES

Max Candidates in the network

Found in: *Component config* > *Wi-Fi* > *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* > *Configure roaming App* > *Scan Configuration*

Max candidates that can be considered while scanning as a part of the network at one time.

Range:

- from 3 to 20 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 3 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_BACKOFF_TIME

Default time to wait between subsequent roaming attempts.

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App

Time to wait (in seconds) by station before registering for the RSSI event again or start continuous montoring to find better AP.

Range:

- from 0 to 120 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 15 if *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_PERIODIC_RRM_MONITORING

Send periodic neighbor report request to AP for internal list updation

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App

This option will enable station to keep sending RRM neighbor list request to AP and update its internal list.

Default value:

- Yes (enabled) if *CONFIG_ESP_WIFI_11KV_SUPPORT* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_RRM_MONITOR_TIME

Time interval (in seconds) between neighbor report requests to an AP

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App > CONFIG_ESP_WIFI_ROAMING_PERIODIC_RRM_MONITORING

Enable this to send periodic neighbor report requests to the AP. These neighbor report requests provide information about other APs in the same managed network. This information is used for more intelligent roaming.

Range:

- from 0 to 1500 if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_RRM_MONITORING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 60 if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_RRM_MONITORING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ROAMING_RRM_MONITOR_THRESHOLD

Threshold for sending periodic neighbor report requests

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_ROAMING_APP > Configure roaming App > CONFIG_ESP_WIFI_ROAMING_PERIODIC_RRM_MONITORING

The RSSI threshold beyond which we start sending periodic neighbor report requests.

Range:

- from -99 to 0 if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_RRM_MONITORING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- "-20" if *CONFIG_ESP_WIFI_ROAMING_PERIODIC_RRM_MONITORING* && *CONFIG_ESP_WIFI_ENABLE_ROAMING_APP* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_DPP_SUPPORT

Enable DPP support

Found in: Component config > Wi-Fi

Select this option to enable WiFi Easy Connect Support.

Default value:

- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_11R_SUPPORT

Enable 802.11R (Fast Transition) Support

Found in: Component config > Wi-Fi

Select this option to enable WiFi Fast Transition Support.

Default value:

- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_WPS_SOFTAP_REGISTRAR

Add WPS Registrar support in SoftAP mode

Found in: Component config > Wi-Fi

Select this option to enable WPS registrar support in softAP mode.

Default value:

- No (disabled) if *CONFIG_ESP_WIFI_SOFTAP_SUPPORT* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_WIFI_TX_STATS

Enable Wi-Fi transmission statistics

Found in: Component config > Wi-Fi

Enable Wi-Fi transmission statistics. Total support 4 access category. Each access category will use 346 bytes memory.

Default value:

- No (disabled) if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_WIFI_RX_STATS

Enable Wi-Fi reception statistics

Found in: Component config > Wi-Fi

Enable Wi-Fi reception statistics. Total support 2 access category. Each access category will use 190 bytes memory.

Default value:

- No (disabled) if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_WIFI_RX_MU_STATS

Enable Wi-Fi DL MU-MIMO and DL OFDMA reception statistics

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENABLE_WIFI_RX_STATS

Enable Wi-Fi DL MU-MIMO and DL OFDMA reception statistics. Will use 10932 bytes memory.

Default value:

- No (disabled) if *CONFIG_ESP_WIFI_ENABLE_WIFI_RX_STATS* && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_TX_HETB_QUEUE_NUM

WiFi TX HE TB QUEUE number for STA HE TB PPDU transmission

Found in: Component config > Wi-Fi

Set the maximum number of queue that can be aggregated by the STA in the A-MPDU carried in the HE TB PPDU.

Range:

- from 1 to 4 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

Default value:

- 3 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_DUMP_HESIGB

Enable Wi-Fi dump HE-SIGB which is contained in DL HE MU PPDUs

Found in: Component config > Wi-Fi

Enable Wi-Fi dump HE-SIGB which is contained in DL HE MU PPDUs.

Default value:

- No (disabled) if SOC_WIFI_HE_SUPPORT_5G && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_DUMP_MU_CFO

Enable Wi-Fi dump MU CFO

Found in: Component config > Wi-Fi

Enable Wi-Fi dump MU CFO.

Default value:

- No (disabled) if SOC_WIFI_HE_SUPPORT_5G && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_DUMP_CTRL_NDPA

Enable Wi-Fi dump NDPA frames

Found in: Component config > Wi-Fi

Enable Wi-Fi dump NDPA frames.

Default value:

- No (disabled) if SOC_WIFI_HE_SUPPORT_5G && *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENABLE_DUMP_CTRL_BFRP

Enable Wi-Fi dump BFRP frames

Found in: Component config > Wi-Fi

Enable Wi-Fi dump BFRP frames.

Default value:

- No (disabled) if SOC_WIFI_HE_SUPPORT_5G && *CONFIG_ESP_HOST_WIFI_ENABLED*

WPS Configuration Options Contains:

- *CONFIG_ESP_WIFI_WPS_PASSPHRASE*
- *CONFIG_ESP_WIFI_WPS_STRICT*

CONFIG_ESP_WIFI_WPS_STRICT

Strictly validate all WPS attributes

Found in: Component config > Wi-Fi > WPS Configuration Options

Select this option to enable validate each WPS attribute rigorously. Disabling this add the workarounds with various APs. Enabling this may cause inter operability issues with some APs.

Default value:

- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_WPS_PASSPHRASE

Get WPA2 passphrase in WPS config

Found in: Component config > Wi-Fi > WPS Configuration Options

Select this option to get passphrase during WPS configuration. This option fakes the virtual display capabilities to get the configuration in passphrase mode. Not recommended to be used since WPS credentials should not be shared to other devices, making it in readable format increases that risk, also passphrase requires pbkdf2 to convert in psk.

Default value:

- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_DEBUG_PRINT

Print debug messages from WPA Supplicant

Found in: Component config > Wi-Fi

Select this option to print logging information from WPA supplicant, this includes handshake information and key hex dumps depending on the project logging level.

Enabling this could increase the build size ~60kb depending on the project logging level.

Default value:

- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_TESTING_OPTIONS

Add DPP testing code

Found in: Component config > Wi-Fi

Select this to enable unity test for DPP.

Default value:

- No (disabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENTERPRISE_SUPPORT

Enable enterprise option

Found in: Component config > Wi-Fi

Select this to enable/disable enterprise connection support.

disabling this will reduce binary size. disabling this will disable the use of any esp_wifi_sta_wpa2_ent_* (as APIs will be meaningless)

Note that when using bigger certificates on low-power chips without crypto hardware acceleration, it is recommended to adjust the task watchdog timer (TWDT) if it is enabled. For precise information on timing requirements, you can check performance numbers at https://github.com/espressif/mbedtls/wiki/Performance-Numbers.

Default value:

- Yes (enabled) if *CONFIG_ESP_HOST_WIFI_ENABLED*

CONFIG_ESP_WIFI_ENT_FREE_DYNAMIC_BUFFER

Free dynamic buffers during WiFi enterprise connection

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_ENTERPRISE_SUPPORT

Select this configuration to free dynamic buffers during WiFi enterprise connection. This will enable chip to reduce heap consumption during WiFi enterprise connection.

Default value:

- No (disabled) if *CONFIG_ESP_WIFI_ENTERPRISE_SUPPORT* && *CONFIG_ESP_HOST_WIFI_ENABLED*

Core dump

Contains:

- *CONFIG_ESP_COREDUMP_CHECK_BOOT*
- *CONFIG_ESP_COREDUMP_DATA_FORMAT*
- *CONFIG_ESP_COREDUMP_CHECKSUM*
- *CONFIG_ESP_COREDUMP_TO_FLASH_OR_UART*
- *CONFIG_ESP_COREDUMP_UART_DELAY*
- *CONFIG_ESP_COREDUMP_FLASH_NO_OVERWRITE*
- *CONFIG_ESP_COREDUMP_LOGS*
- *CONFIG_ESP_COREDUMP_DECODE*
- *CONFIG_ESP_COREDUMP_CAPTURE_DRAM*
- *CONFIG_ESP_COREDUMP_MAX_TASKS_NUM*
- *CONFIG_ESP_COREDUMP_STACK_SIZE*
- *CONFIG_ESP_COREDUMP_SUMMARY_STACKDUMP_SIZE*

CONFIG_ESP_COREDUMP_TO_FLASH_OR_UART

Data destination

Found in: Component config > Core dump

Select place to store core dump: flash, uart or none (to disable core dumps generation).

Core dumps to Flash are not available if PSRAM is used for task stacks.

If core dump is configured to be stored in flash and custom partition table is used add corresponding entry to your CSV. For examples, please see predefined partition table CSV descriptions in the components/partition_table directory.

Available options:

- Flash (CONFIG_ESP_COREDUMP_ENABLE_TO_FLASH)
- UART (CONFIG_ESP_COREDUMP_ENABLE_TO_UART)
- None (CONFIG_ESP_COREDUMP_ENABLE_TO_NONE)

CONFIG_ESP_COREDUMP_DATA_FORMAT

Core dump data format

Found in: Component config > Core dump

Select the data format for core dump.

Available options:

- Binary format (CONFIG_ESP_COREDUMP_DATA_FORMAT_BIN)
- ELF format (CONFIG_ESP_COREDUMP_DATA_FORMAT_ELF)

CONFIG_ESP_COREDUMP_CHECKSUM

Core dump data integrity check

Found in: Component config > Core dump

Select the integrity check for the core dump.

Available options:

- Use CRC32 for integrity verification (CONFIG_ESP_COREDUMP_CHECKSUM_CRC32)
- Use SHA256 for integrity verification (CONFIG_ESP_COREDUMP_CHECKSUM_SHA256)

CONFIG_ESP_COREDUMP_CAPTURE_DRAM

Include whole .bss and .data sections and heap data into core dump file

Found in: Component config > Core dump

Storing these sections can help with easier debugging and troubleshooting. However, additional storage space will be required in the core dump partition. At least 128KB should be reserved, but the actual amount required may vary based on the application's DRAM usage. Note that sections located in external RAM will not be stored.

Default value:

- No (disabled) if *CONFIG_ESP_COREDUMP_DATA_FORMAT_ELF*

CONFIG_ESP_COREDUMP_CHECK_BOOT

Check core dump data integrity on boot

Found in: Component config > Core dump

When enabled, if any data are found on the flash core dump partition, they will be checked by calculating their checksum.

Default value:

- Yes (enabled) if *CONFIG_ESP_COREDUMP_ENABLE_TO_FLASH*

CONFIG_ESP_COREDUMP_LOGS

Enable coredump logs for debugging

Found in: Component config > Core dump

Enable/disable coredump logs. Logs strings from espcoredump component are placed in DRAM. Disabling these helps to save ~5KB of internal memory.

CONFIG_ESP_COREDUMP_MAX_TASKS_NUM

Maximum number of tasks

Found in: Component config > Core dump

Maximum number of tasks snapshots in core dump.

CONFIG_ESP_COREDUMP_UART_DELAY

Delay before print to UART

Found in: Component config > Core dump

Config delay (in ms) before printing core dump to UART. Delay can be interrupted by pressing Enter key.

Default value:

- 0 if *CONFIG_ESP_COREDUMP_ENABLE_TO_UART*

CONFIG_ESP_COREDUMP_FLASH_NO_OVERWRITE

Don't overwrite existing core dump

Found in: Component config > Core dump

Don't overwrite an existing core dump already present in flash. Enable this option to only keep the first of multiple core dumps.

If enabled, the core dump partition must be erased before the first core dump can be written.

Default value:

- No (disabled) if *CONFIG_ESP_COREDUMP_ENABLE_TO_FLASH*

CONFIG_ESP_COREDUMP_STACK_SIZE

Reserved stack size

Found in: Component config > Core dump

Size of the memory to be reserved for core dump stack. If 0 core dump process will run on the stack of crashed task/ISR, otherwise special stack will be allocated. To ensure that core dump itself will not overflow task/ISR stack set this to the value around 1300-1800 depending on the chosen checksum calculation method. SHA256 method needs more stack space than CRC32. NOTE: It eats DRAM.

CONFIG_ESP_COREDUMP_SUMMARY_STACKDUMP_SIZE

Size of the stack dump buffer

Found in: Component config > Core dump

Size of the buffer that would be reserved for extracting backtrace info summary. This buffer will contain the stack dump of the crashed task. This dump is useful in generating backtrace

Range:

- from 512 to 4096 if *CONFIG_ESP_COREDUMP_DATA_FORMAT_ELF* && *CONFIG_ESP_COREDUMP_ENABLE_TO_FLASH*

Default value:

- 1024 if *CONFIG_ESP_COREDUMP_DATA_FORMAT_ELF* && *CONFIG_ESP_COREDUMP_ENABLE_TO_FLASH*

CONFIG_ESP_COREDUMP_DECODE

Handling of UART core dumps in IDF Monitor

Found in: Component config > *Core dump*

Available options:

- Decode and show summary (info_corefile) (CONFIG_ESP_COREDUMP_DECODE_INFO)
- Don't decode (CONFIG_ESP_COREDUMP_DECODE_DISABLE)

FAT Filesystem support Contains:

- *CONFIG_FATFS_API_ENCODING*
- *CONFIG_FATFS_VFS_FSTAT_BLKSIZE*
- *CONFIG_FATFS_IMMEDIATE_FSYNC*
- *CONFIG_FATFS_USE_FASTSEEK*
- *CONFIG_FATFS_LONG_FILENAMES*
- *CONFIG_FATFS_MAX_LFN*
- *CONFIG_FATFS_FS_LOCK*
- *CONFIG_FATFS_VOLUME_COUNT*
- *CONFIG_FATFS_CHOOSE_CODEPAGE*
- *CONFIG_FATFS_LINK_LOCK*
- *CONFIG_FATFS_ALLOC_PREFER_EXTRAM*
- *CONFIG_FATFS_SECTOR_SIZE*
- *CONFIG_FATFS_TIMEOUT_MS*
- *CONFIG_FATFS_USE_DYN_BUFFERS*
- *CONFIG_FATFS_USE_LABEL*
- *CONFIG_FATFS_PER_FILE_CACHE*

CONFIG_FATFS_VOLUME_COUNT

Number of volumes

Found in: Component config > *FAT Filesystem support*

Number of volumes (logical drives) to use.

Range:

- from 1 to 10

Default value:

- 2

CONFIG_FATFS_LONG_FILENAMES

Long filename support

Found in: Component config > *FAT Filesystem support*

Support long filenames in FAT. Long filename data increases memory usage. FATFS can be configured to store the buffer for long filename data in stack or heap.

Available options:

- No long filenames (CONFIG_FATFS_LFN_NONE)
- Long filename buffer in heap (CONFIG_FATFS_LFN_HEAP)
- Long filename buffer on stack (CONFIG_FATFS_LFN_STACK)

CONFIG_FATFS_SECTOR_SIZE

Sector size

Found in: Component config > FAT Filesystem support

Specify the size of the sector in bytes for FATFS partition generator.

Available options:

- 512 (CONFIG_FATFS_SECTOR_512)
- 4096 (CONFIG_FATFS_SECTOR_4096)

CONFIG_FATFS_CHOOSE_CODEPAGE

OEM Code Page

Found in: Component config > FAT Filesystem support

OEM code page used for file name encodings.

If ”Dynamic” is selected, code page can be chosen at runtime using f_setcp function. Note that choosing this option will increase application size by ~480kB.

Available options:

- Dynamic (all code pages supported) (CONFIG_FATFS_CODEPAGE_DYNAMIC)
- US (CP437) (CONFIG_FATFS_CODEPAGE_437)
- Arabic (CP720) (CONFIG_FATFS_CODEPAGE_720)
- Greek (CP737) (CONFIG_FATFS_CODEPAGE_737)
- KBL (CP771) (CONFIG_FATFS_CODEPAGE_771)
- Baltic (CP775) (CONFIG_FATFS_CODEPAGE_775)
- Latin 1 (CP850) (CONFIG_FATFS_CODEPAGE_850)
- Latin 2 (CP852) (CONFIG_FATFS_CODEPAGE_852)
- Cyrillic (CP855) (CONFIG_FATFS_CODEPAGE_855)
- Turkish (CP857) (CONFIG_FATFS_CODEPAGE_857)
- Portuguese (CP860) (CONFIG_FATFS_CODEPAGE_860)
- Icelandic (CP861) (CONFIG_FATFS_CODEPAGE_861)
- Hebrew (CP862) (CONFIG_FATFS_CODEPAGE_862)
- Canadian French (CP863) (CONFIG_FATFS_CODEPAGE_863)
- Arabic (CP864) (CONFIG_FATFS_CODEPAGE_864)
- Nordic (CP865) (CONFIG_FATFS_CODEPAGE_865)
- Russian (CP866) (CONFIG_FATFS_CODEPAGE_866)
- Greek 2 (CP869) (CONFIG_FATFS_CODEPAGE_869)
- Japanese (DBCS) (CP932) (CONFIG_FATFS_CODEPAGE_932)
- Simplified Chinese (DBCS) (CP936) (CONFIG_FATFS_CODEPAGE_936)
- Korean (DBCS) (CP949) (CONFIG_FATFS_CODEPAGE_949)
- Traditional Chinese (DBCS) (CP950) (CONFIG_FATFS_CODEPAGE_950)

CONFIG_FATFS_MAX_LFN

Max long filename length

Found in: Component config > FAT Filesystem support

Maximum long filename length. Can be reduced to save RAM.

CONFIG_FATFS_API_ENCODING

API character encoding

Found in: Component config > FAT Filesystem support

Choose encoding for character and string arguments/returns when using FATFS APIs. The encoding of arguments will usually depend on text editor settings.

Available options:

- API uses ANSI/OEM encoding (CONFIG_FATFS_API_ENCODING_ANSI_OEM)
- API uses UTF-8 encoding (CONFIG_FATFS_API_ENCODING_UTF_8)

CONFIG_FATFS_FS_LOCK

Number of simultaneously open files protected by lock function

Found in: Component config > FAT Filesystem support

This option sets the FATFS configuration value _FS_LOCK. The option _FS_LOCK switches file lock function to control duplicated file open and illegal operation to open objects.

* 0: Disable file lock function. To avoid volume corruption, application should avoid illegal open, remove and rename to the open objects.

* >0: Enable file lock function. The value defines how many files/sub-directories can be opened simultaneously under file lock control.

Note that the file lock control is independent of re-entrancy.

Range:
- from 0 to 65535

Default value:
- 0

CONFIG_FATFS_TIMEOUT_MS

Timeout for acquiring a file lock, ms

Found in: Component config > FAT Filesystem support

This option sets FATFS configuration value _FS_TIMEOUT, scaled to milliseconds. Sets the number of milliseconds FATFS will wait to acquire a mutex when operating on an open file. For example, if one task is performing a lengthy operation, another task will wait for the first task to release the lock, and time out after amount of time set by this option.

Default value:
- 10000

CONFIG_FATFS_PER_FILE_CACHE

Use separate cache for each file

Found in: Component config > FAT Filesystem support

This option affects FATFS configuration value _FS_TINY.

If this option is set, _FS_TINY is 0, and each open file has its own cache, size of the cache is equal to the _MAX_SS variable (512 or 4096 bytes). This option uses more RAM if more than 1 file is open, but needs less reads and writes to the storage for some operations.

If this option is not set, _FS_TINY is 1, and single cache is used for all open files, size is also equal to _MAX_SS variable. This reduces the amount of heap used when multiple files are open, but increases the number of read and write operations which FATFS needs to make.

Default value:

- Yes (enabled)

CONFIG_FATFS_ALLOC_PREFER_EXTRAM

Prefer external RAM when allocating FATFS buffers

Found in: Component config > FAT Filesystem support

When the option is enabled, internal buffers used by FATFS will be allocated from external RAM. If the allocation from external RAM fails, the buffer will be allocated from the internal RAM. Disable this option if optimizing for performance. Enable this option if optimizing for internal memory size.

Default value:

- Yes (enabled) if *CONFIG_SPIRAM_USE_CAPS_ALLOC* || *CONFIG_SPIRAM_USE_MALLOC*

CONFIG_FATFS_USE_FASTSEEK

Enable fast seek algorithm when using lseek function through VFS FAT

Found in: Component config > FAT Filesystem support

The fast seek feature enables fast backward/long seek operations without FAT access by using an in-memory CLMT (cluster link map table). Please note, fast-seek is only allowed for read-mode files, if a file is opened in write-mode, the seek mechanism will automatically fallback to the default implementation.

Default value:

- No (disabled)

CONFIG_FATFS_FAST_SEEK_BUFFER_SIZE

Fast seek CLMT buffer size

Found in: Component config > FAT Filesystem support > CONFIG_FATFS_USE_FASTSEEK

If fast seek algorithm is enabled, this defines the size of CLMT buffer used by this algorithm in 32-bit word units. This value should be chosen based on prior knowledge of maximum elements of each file entry would store.

Default value:

- 64 if *CONFIG_FATFS_USE_FASTSEEK*

CONFIG_FATFS_VFS_FSTAT_BLKSIZE

Default block size

Found in: Component config > FAT Filesystem support

If set to 0, the 'newlib' library's default size (BLKSIZ) is used (128 B). If set to a non-zero value, the value is used as the block size. Default file buffer size is set to this value and the buffer is allocated when first attempt of reading/writing to a file is made. Increasing this value improves fread() speed, however the heap usage is increased as well.

NOTE: The block size value is shared by all the filesystem functions accessing target media for given file descriptor! See 'Improving I/O performance' section of 'Maximizing Execution Speed' documentation page for more details.

Default value:

- 0

CONFIG_FATFS_IMMEDIATE_FSYNC

Enable automatic f_sync

Found in: Component config > FAT Filesystem support

Enables automatic calling of f_sync() to flush recent file changes after each call of vfs_fat_write(), vfs_fat_pwrite(), vfs_fat_link(), vfs_fat_truncate() and vfs_fat_ftruncate() functions. This feature improves file-consistency and size reporting accuracy for the FatFS, at a price on decreased performance due to frequent disk operations

Default value:

- No (disabled)

CONFIG_FATFS_USE_LABEL

Use FATFS volume label

Found in: Component config > FAT Filesystem support

Allows FATFS volume label to be specified using f_setlabel

Default value:

- No (disabled)

CONFIG_FATFS_LINK_LOCK

Perform the whole link operation under lock

Found in: Component config > FAT Filesystem support

If enabled, the whole link operation (including file copying) is performed under lock. This ensures that the link operation is atomic, but may cause performance for large files. It may create less fragmented file copy.

Default value:

- Yes (enabled)

CONFIG_FATFS_USE_DYN_BUFFERS

Use dynamic buffers

Found in: Component config > FAT Filesystem support

If enabled, the buffers used by FATFS will be allocated separately from the rest of the structure. This option is useful when using multiple FATFS instances with different sector sizes, as the buffers will be allocated according to the sector size. If disabled, the greatest sector size will be used for all FATFS instances. (In most cases, this would be the sector size of Wear Levelling library) This might cause more memory to be used than necessary.

Default value:

- Yes (enabled) if CONFIG_WL_SECTOR_SIZE_4096

FreeRTOS

Contains:

- *Kernel*
- *Port*

Kernel

Contains:

- *CONFIG_FREERTOS_CHECK_STACKOVERFLOW*
- *CONFIG_FREERTOS_ENABLE_BACKWARD_COMPATIBILITY*
- *CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS*
- *CONFIG_FREERTOS_MAX_TASK_NAME_LEN*

- *CONFIG_FREERTOS_IDLE_TASK_STACKSIZE*
- *CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS*
- *CONFIG_FREERTOS_QUEUE_REGISTRY_SIZE*
- *CONFIG_FREERTOS_TASK_NOTIFICATION_ARRAY_ENTRIES*
- *CONFIG_FREERTOS_HZ*
- *CONFIG_FREERTOS_TIMER_QUEUE_LENGTH*
- *CONFIG_FREERTOS_TIMER_SERVICE_TASK_CORE_AFFINITY*
- *CONFIG_FREERTOS_TIMER_SERVICE_TASK_NAME*
- *CONFIG_FREERTOS_TIMER_TASK_PRIORITY*
- *CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH*
- *CONFIG_FREERTOS_USE_APPLICATION_TASK_TAG*
- *CONFIG_FREERTOS_USE_IDLE_HOOK*
- *CONFIG_FREERTOS_USE_LIST_DATA_INTEGRITY_CHECK_BYTES*
- *CONFIG_FREERTOS_OPTIMIZED_SCHEDULER*
- *CONFIG_FREERTOS_USE_TICK_HOOK*
- *CONFIG_FREERTOS_USE_TICKLESS_IDLE*
- *CONFIG_FREERTOS_USE_TRACE_FACILITY*
- *CONFIG_FREERTOS_VTASKLIST_INCLUDE_COREID*
- *CONFIG_FREERTOS_UNICORE*

CONFIG_FREERTOS_UNICORE

Run FreeRTOS only on first core

Found in: Component config > FreeRTOS > Kernel

This version of FreeRTOS normally takes control of all cores of the CPU. Select this if you only want to start it on the first core. This is needed when e.g. another process needs complete control over the second core.

CONFIG_FREERTOS_HZ

configTICK_RATE_HZ

Found in: Component config > FreeRTOS > Kernel

Sets the FreeRTOS tick interrupt frequency in Hz (see configTICK_RATE_HZ documentation for more details).

Range:
- from 1 to 1000

Default value:
- 100

CONFIG_FREERTOS_OPTIMIZED_SCHEDULER

configUSE_PORT_OPTIMISED_TASK_SELECTION

Found in: Component config > FreeRTOS > Kernel

Enables port specific task selection method. This option can speed up the search of ready tasks when scheduling (see configUSE_PORT_OPTIMISED_TASK_SELECTION documentation for more details).

Default value:
- Yes (enabled) if *CONFIG_FREERTOS_UNICORE*

CONFIG_FREERTOS_CHECK_STACKOVERFLOW

configCHECK_FOR_STACK_OVERFLOW

Found in: Component config > FreeRTOS > Kernel

Enables FreeRTOS to check for stack overflows (see configCHECK_FOR_STACK_OVERFLOW documentation for more details).

Note: If users do not provide their own `vApplicationStackOverflowHook()` function, a default function will be provided by ESP-IDF.

Available options:

- No checking (CONFIG_FREERTOS_CHECK_STACKOVERFLOW_NONE)
  Do not check for stack overflows (configCHECK_FOR_STACK_OVERFLOW = 0)
- Check by stack pointer value (Method 1) (CONFIG_FREERTOS_CHECK_STACKOVERFLOW_PTRVAL)
  Check for stack overflows on each context switch by checking if the stack pointer is in a valid range. Quick but does not detect stack overflows that happened between context switches (configCHECK_FOR_STACK_OVERFLOW = 1)
- Check using canary bytes (Method 2) (CONFIG_FREERTOS_CHECK_STACKOVERFLOW_CANARY)
  Places some magic bytes at the end of the stack area and on each context switch, check if these bytes are still intact. More thorough than just checking the pointer, but also slightly slower. (configCHECK_FOR_STACK_OVERFLOW = 2)

CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS

configNUM_THREAD_LOCAL_STORAGE_POINTERS

Found in: Component config > FreeRTOS > Kernel

Set the number of thread local storage pointers in each task (see configNUM_THREAD_LOCAL_STORAGE_POINTERS documentation for more details).

Note: In ESP-IDF, this value must be at least 1. Index 0 is reserved for use by the pthreads API thread-local-storage. Other indexes can be used for any desired purpose.

Range:
- from 1 to 256

Default value:
- 1

CONFIG_FREERTOS_IDLE_TASK_STACKSIZE

configMINIMAL_STACK_SIZE (Idle task stack size)

Found in: Component config > FreeRTOS > Kernel

Sets the idle task stack size in bytes (see configMINIMAL_STACK_SIZE documentation for more details).

Note:

- ESP-IDF specifies stack sizes in bytes instead of words.
- The default size is enough for most use cases.
- The stack size may need to be increased above the default if the app installs idle or thread local storage cleanup hooks that use a lot of stack memory.
- Conversely, the stack size can be reduced to the minimum if non of the idle features are used.

Range:
- from 768 to 32768

Default value:
- 1536

CONFIG_FREERTOS_USE_IDLE_HOOK

configUSE_IDLE_HOOK

Found in: Component config > FreeRTOS > Kernel

Enables the idle task application hook (see configUSE_IDLE_HOOK documentation for more details).

Note:

- The application must provide the hook function `void vApplicationIdleHook( void );`
- `vApplicationIdleHook()` is called from FreeRTOS idle task(s)
- The FreeRTOS idle hook is NOT the same as the ESP-IDF Idle Hook, but both can be enabled simultaneously.

Default value:

- No (disabled)

CONFIG_FREERTOS_USE_TICK_HOOK

configUSE_TICK_HOOK

Found in: Component config > FreeRTOS > Kernel

Enables the tick hook (see configUSE_TICK_HOOK documentation for more details).

Note:

- The application must provide the hook function `void vApplicationTickHook( void );`
- `vApplicationTickHook()` is called from FreeRTOS's tick handling function `xTaskIncrementTick()`
- The FreeRTOS tick hook is NOT the same as the ESP-IDF Tick Interrupt Hook, but both can be enabled simultaneously.

Default value:

- No (disabled)

CONFIG_FREERTOS_MAX_TASK_NAME_LEN

configMAX_TASK_NAME_LEN

Found in: Component config > FreeRTOS > Kernel

Sets the maximum number of characters for task names (see configMAX_TASK_NAME_LEN documentation for more details).

Note: For most uses, the default of 16 characters is sufficient.

Range:

- from 1 to 256

Default value:

- 16

CONFIG_FREERTOS_ENABLE_BACKWARD_COMPATIBILITY

configENABLE_BACKWARD_COMPATIBILITY

Found in: Component config > FreeRTOS > Kernel

Enable backward compatibility with APIs prior to FreeRTOS v8.0.0. (see configENABLE_BACKWARD_COMPATIBILITY documentation for more details).

Default value:

- No (disabled)

CONFIG_FREERTOS_TIMER_SERVICE_TASK_NAME

configTIMER_SERVICE_TASK_NAME

Found in: Component config > FreeRTOS > Kernel

Sets the timer task's name (see configTIMER_SERVICE_TASK_NAME documentation for more details).

Default value:

- "Tmr Svc"

CONFIG_FREERTOS_TIMER_SERVICE_TASK_CORE_AFFINITY

configTIMER_SERVICE_TASK_CORE_AFFINITY

Found in: Component config > FreeRTOS > Kernel

Sets the timer task's core affinity (see configTIMER_SERVICE_TASK_CORE_AFFINITY documentation for more details).

Available options:

- CPU0 (CONFIG_FREERTOS_TIMER_TASK_AFFINITY_CPU0)
- CPU1 (CONFIG_FREERTOS_TIMER_TASK_AFFINITY_CPU1)
- No affinity (CONFIG_FREERTOS_TIMER_TASK_NO_AFFINITY)

CONFIG_FREERTOS_TIMER_TASK_PRIORITY

configTIMER_TASK_PRIORITY

Found in: Component config > FreeRTOS > Kernel

Sets the timer task's priority (see configTIMER_TASK_PRIORITY documentation for more details).

Range:

- from 1 to 25

Default value:

- 1

CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH

configTIMER_TASK_STACK_DEPTH

Found in: Component config > FreeRTOS > Kernel

Set the timer task's stack size (see configTIMER_TASK_STACK_DEPTH documentation for more details).

Range:

- from 1536 to 32768

Default value:

- 2048

CONFIG_FREERTOS_TIMER_QUEUE_LENGTH

configTIMER_QUEUE_LENGTH

Found in: Component config > FreeRTOS > Kernel

Set the timer task's command queue length (see configTIMER_QUEUE_LENGTH documentation for more details).

Range:

- from 5 to 20

Default value:

- 10

CONFIG_FREERTOS_QUEUE_REGISTRY_SIZE

configQUEUE_REGISTRY_SIZE

Found in: Component config > FreeRTOS > Kernel

Set the size of the queue registry (see configQUEUE_REGISTRY_SIZE documentation for more details).

Note: A value of 0 will disable queue registry functionality

Range:

- from 0 to 20

Default value:

- 0

CONFIG_FREERTOS_TASK_NOTIFICATION_ARRAY_ENTRIES

configTASK_NOTIFICATION_ARRAY_ENTRIES

Found in: Component config > FreeRTOS > Kernel

Set the size of the task notification array of each task. When increasing this value, keep in mind that this means additional memory for each and every task on the system. However, task notifications in general are more light weight compared to alternatives such as semaphores.

Range:

- from 1 to 32

Default value:

- 1

CONFIG_FREERTOS_USE_TRACE_FACILITY

configUSE_TRACE_FACILITY

Found in: Component config > FreeRTOS > Kernel

Enables additional structure members and functions to assist with execution visualization and tracing (see configUSE_TRACE_FACILITY documentation for more details).

Default value:

- No (disabled)

CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS

configUSE_STATS_FORMATTING_FUNCTIONS

Found in: Component config > FreeRTOS > Kernel > CONFIG_FREERTOS_USE_TRACE_FACILITY

Set configUSE_TRACE_FACILITY and configUSE_STATS_FORMATTING_FUNCTIONS to 1 to include the `vTaskList()` and `vTaskGetRunTimeStats()` functions in the build (see configUSE_STATS_FORMATTING_FUNCTIONS documentation for more details).

Default value:

- No (disabled) if *CONFIG_FREERTOS_USE_TRACE_FACILITY*

CONFIG_FREERTOS_USE_LIST_DATA_INTEGRITY_CHECK_BYTES

configUSE_LIST_DATA_INTEGRITY_CHECK_BYTES

Found in: Component config > FreeRTOS > Kernel

Enable list integrity checker (see configUSE_LIST_DATA_INTEGRITY_CHECK_BYTES documentation for more details).

Default value:

- No (disabled)

CONFIG_FREERTOS_VTASKLIST_INCLUDE_COREID

Enable display of xCoreID in vTaskList

Found in: Component config > FreeRTOS > Kernel

If enabled, this will include an extra column when vTaskList is called to display the CoreID the task is pinned to (0,1) or -1 if not pinned.

Default value:

- No (disabled) if *CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS*

CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS

configGENERATE_RUN_TIME_STATS

Found in: Component config > FreeRTOS > Kernel

Enables collection of run time statistics for each task (see configGENERATE_RUN_TIME_STATS documentation for more details).

Note: The clock used for run time statistics can be configured in FREERTOS_RUN_TIME_STATS_CLK.

Default value:

- No (disabled)

CONFIG_FREERTOS_RUN_TIME_COUNTER_TYPE

configRUN_TIME_COUNTER_TYPE

Found in: Component config > FreeRTOS > Kernel > CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS

Sets the data type used for the FreeRTOS run time stats. A larger data type can be used to reduce the frequency of the counter overflowing.

Available options:

- uint32_t (CONFIG_FREERTOS_RUN_TIME_COUNTER_TYPE_U32)
  configRUN_TIME_COUNTER_TYPE is set to uint32_t
- uint64_t (CONFIG_FREERTOS_RUN_TIME_COUNTER_TYPE_U64)
  configRUN_TIME_COUNTER_TYPE is set to uint64_t

CONFIG_FREERTOS_USE_TICKLESS_IDLE

configUSE_TICKLESS_IDLE

Found in: Component config > FreeRTOS > Kernel

If power management support is enabled, FreeRTOS will be able to put the system into light sleep mode when no tasks need to run for a number of ticks. This number can be set using FREERTOS_IDLE_TIME_BEFORE_SLEEP option. This feature is also known as ”automatic light sleep”.

Note that timers created using esp_timer APIs may prevent the system from entering sleep mode, even when no tasks need to run. To skip unnecessary wake-up initialize a timer with the ”skip_unhandled_events” option as true.

If disabled, automatic light sleep support will be disabled.

Default value:

- No (disabled) if *CONFIG_PM_ENABLE*

CONFIG_FREERTOS_IDLE_TIME_BEFORE_SLEEP

configEXPECTED_IDLE_TIME_BEFORE_SLEEP

Found in: Component config > FreeRTOS > Kernel > CONFIG_FREERTOS_USE_TICKLESS_IDLE

FreeRTOS will enter light sleep mode if no tasks need to run for this number of ticks. You can enable PM_PROFILING feature in esp_pm components and dump the sleep status with esp_pm_dump_locks, if the proportion of rejected sleeps is too high, please increase this value to improve scheduling efficiency

Range:

- from 2 to 4294967295 if *CONFIG_FREERTOS_USE_TICKLESS_IDLE*

Default value:

- 3 if *CONFIG_FREERTOS_USE_TICKLESS_IDLE*

CONFIG_FREERTOS_USE_APPLICATION_TASK_TAG

configUSE_APPLICATION_TASK_TAG

Found in: Component config > FreeRTOS > Kernel

Enables task tagging functionality and its associated API (see configUSE_APPLICATION_TASK_TAG documentation for more details).

Default value:

- No (disabled)

Port Contains:

- *CONFIG_FREERTOS_CHECK_MUTEX_GIVEN_BY_OWNER*
- *CONFIG_FREERTOS_RUN_TIME_STATS_CLK*
- *CONFIG_FREERTOS_INTERRUPT_BACKTRACE*
- *CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK*
- *CONFIG_FREERTOS_ENABLE_STATIC_TASK_CLEAN_UP*
- *CONFIG_FREERTOS_TASK_PRE_DELETION_HOOK*
- *CONFIG_FREERTOS_TLSP_DELETION_CALLBACKS*
- *CONFIG_FREERTOS_ISR_STACKSIZE*
- *CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH*
- *CONFIG_FREERTOS_CHECK_PORT_CRITICAL_COMPLIANCE*
- *CONFIG_FREERTOS_CORETIMER*
- *CONFIG_FREERTOS_TASK_FUNCTION_WRAPPER*

CONFIG_FREERTOS_TASK_FUNCTION_WRAPPER

Wrap task functions

Found in: Component config > FreeRTOS > Port

If enabled, all FreeRTOS task functions will be enclosed in a wrapper function. If a task function mistakenly returns (i.e. does not delete), the call flow will return to the wrapper function. The wrapper

function will then log an error and abort the application. This option is also required for GDB backtraces and C++ exceptions to work correctly inside top-level task functions.

Default value:

- Yes (enabled)

CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK

Enable stack overflow debug watchpoint

Found in: Component config > FreeRTOS > Port

FreeRTOS can check if a stack has overflown its bounds by checking either the value of the stack pointer or by checking the integrity of canary bytes. (See FREERTOS_CHECK_STACKOVERFLOW for more information.) These checks only happen on a context switch, and the situation that caused the stack overflow may already be long gone by then. This option will use the last debug memory watchpoint to allow breaking into the debugger (or panic'ing) as soon as any of the last 32 bytes on the stack of a task are overwritten. The side effect is that using gdb, you effectively have one hardware watchpoint less because the last one is overwritten as soon as a task switch happens.

Another consequence is that due to alignment requirements of the watchpoint, the usable stack size decreases by up to 60 bytes. This is because the watchpoint region has to be aligned to its size and the size for the stack watchpoint in IDF is 32 bytes.

This check only triggers if the stack overflow writes within 32 bytes near the end of the stack, rather than overshooting further, so it is worth combining this approach with one of the other stack overflow check methods.

When this watchpoint is hit, gdb will stop with a SIGTRAP message. When no JTAG OCD is attached, esp-idf will panic on an unhandled debug exception.

Default value:

- No (disabled)

CONFIG_FREERTOS_TLSP_DELETION_CALLBACKS

Enable thread local storage pointers deletion callbacks

Found in: Component config > FreeRTOS > Port

ESP-IDF provides users with the ability to free TLSP memory by registering TLSP deletion callbacks. These callbacks are automatically called by FreeRTOS when a task is deleted. When this option is turned on, the memory reserved for TLSPs in the TCB is doubled to make space for storing the deletion callbacks. If the user does not wish to use TLSP deletion callbacks then this option could be turned off to save space in the TCB memory.

Default value:

- Yes (enabled)

CONFIG_FREERTOS_TASK_PRE_DELETION_HOOK

Enable task pre-deletion hook

Found in: Component config > FreeRTOS > Port

Enable this option to make FreeRTOS call a user provided hook function right before it deletes a task (i.e., frees/releases a dynamically/statically allocated task's memory). This is useful if users want to know when a task is actually deleted (in case the task's deletion is delegated to the IDLE task).

If this config option is enabled, users must define a `void vTaskPreDeletionHook( void \* pxTCB )` hook function in their application.

CONFIG_FREERTOS_ENABLE_STATIC_TASK_CLEAN_UP

Enable static task clean up hook (DEPRECATED)

Found in: Component config > FreeRTOS > Port

THIS OPTION IS DEPRECATED. Use FREERTOS_TASK_PRE_DELETION_HOOK instead.

Enable this option to make FreeRTOS call the static task clean up hook when a task is deleted.

Note: Users will need to provide a `void vPortCleanUpTCB ( void \*pxTCB )` callback

Default value:

- No (disabled)

CONFIG_FREERTOS_CHECK_MUTEX_GIVEN_BY_OWNER

Check that mutex semaphore is given by owner task

Found in: Component config > FreeRTOS > Port

If enabled, assert that when a mutex semaphore is given, the task giving the semaphore is the task which is currently holding the mutex.

Default value:

- Yes (enabled)

CONFIG_FREERTOS_ISR_STACKSIZE

ISR stack size

Found in: Component config > FreeRTOS > Port

The interrupt handlers have their own stack. The size of the stack can be defined here. Each processor has its own stack, so the total size occupied will be twice this.

Range:

- from 2096 to 32768 if *CONFIG_ESP_COREDUMP_DATA_FORMAT_ELF*
- from 1536 to 32768

Default value:

- 2096 if *CONFIG_ESP_COREDUMP_DATA_FORMAT_ELF*
- 1536

CONFIG_FREERTOS_INTERRUPT_BACKTRACE

Enable backtrace from interrupt to task context

Found in: Component config > FreeRTOS > Port

If this option is enabled, interrupt stack frame will be modified to point to the code of the interrupted task as its return address. This helps the debugger (or the panic handler) show a backtrace from the interrupt to the task which was interrupted. This also works for nested interrupts: higher level interrupt stack can be traced back to the lower level interrupt. This option adds 4 instructions to the interrupt dispatching code.

Default value:

- Yes (enabled)

CONFIG_FREERTOS_CORETIMER

Tick timer source (Xtensa Only)

Found in: Component config > FreeRTOS > Port

FreeRTOS needs a timer with an associated interrupt to use as the main tick source to increase counters, run timers and do pre-emptive multitasking with. There are multiple timers available to do this, with different interrupt priorities.

Available options:

- Timer 0 (int 6, level 1) (CONFIG_FREERTOS_CORETIMER_0)
  Select this to use timer 0
- Timer 1 (int 15, level 3) (CONFIG_FREERTOS_CORETIMER_1)
  Select this to use timer 1
- SYSTIMER 0 (level 1) (CONFIG_FREERTOS_CORETIMER_SYSTIMER_LVL1)
  Select this to use systimer with the 1 interrupt priority.
- SYSTIMER 0 (level 3) (CONFIG_FREERTOS_CORETIMER_SYSTIMER_LVL3)
  Select this to use systimer with the 3 interrupt priority.

CONFIG_FREERTOS_RUN_TIME_STATS_CLK

Choose the clock source for run time stats

Found in: Component config > FreeRTOS > Port

Choose the clock source for FreeRTOS run time stats. Options are CPU0's CPU Clock or the ESP Timer. Both clock sources are 32 bits. The CPU Clock can run at a higher frequency hence provide a finer resolution but will overflow much quicker. Note that run time stats are only valid until the clock source overflows.

Available options:

- Use ESP TIMER for run time stats (CONFIG_FREERTOS_RUN_TIME_STATS_USING_ESP_TIMER)
  ESP Timer will be used as the clock source for FreeRTOS run time stats. The ESP Timer runs at a frequency of 1MHz regardless of Dynamic Frequency Scaling. Therefore the ESP Timer will overflow in approximately 4290 seconds.
- Use CPU Clock for run time stats (CONFIG_FREERTOS_RUN_TIME_STATS_USING_CPU_CLK)
  CPU Clock will be used as the clock source for the generation of run time stats. The CPU Clock has a frequency dependent on ESP_DEFAULT_CPU_FREQ_MHZ and Dynamic Frequency Scaling (DFS). Therefore the CPU Clock frequency can fluctuate between 80 to 240MHz. Run time stats generated using the CPU Clock represents the number of CPU cycles each task is allocated and DOES NOT reflect the amount of time each task runs for (as CPU clock frequency can change). If the CPU clock consistently runs at the maximum frequency of 240MHz, it will overflow in approximately 17 seconds.

CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH

Place FreeRTOS functions into Flash

Found in: Component config > FreeRTOS > Port

When enabled the selected Non-ISR FreeRTOS functions will be placed into Flash memory instead of IRAM. This saves up to 8KB of IRAM depending on which functions are used.

Default value:

- No (disabled)

CONFIG_FREERTOS_CHECK_PORT_CRITICAL_COMPLIANCE

Tests compliance with Vanilla FreeRTOS port*_CRITICAL calls

Found in: Component config > FreeRTOS > Port

If enabled, context of port*_CRITICAL calls (ISR or Non-ISR) would be checked to be in compliance with Vanilla FreeRTOS. e.g Calling port*_CRITICAL from ISR context would cause assert failure

Default value:

- No (disabled)

Hardware Abstraction Layer (HAL) and Low Level (LL) Contains:

- *CONFIG_HAL_DEFAULT_ASSERTION_LEVEL*
- *CONFIG_HAL_ECDSA_GEN_SIG_CM*
- *CONFIG_HAL_LOG_LEVEL*
- *CONFIG_HAL_SYSTIMER_USE_ROM_IMPL*
- *CONFIG_HAL_WDT_USE_ROM_IMPL*

CONFIG_HAL_DEFAULT_ASSERTION_LEVEL

Default HAL assertion level

Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)

Set the assert behavior / level for HAL component. HAL component assert level can be set separately, but the level can't exceed the system assertion level. e.g. If the system assertion is disabled, then the HAL assertion can't be enabled either. If the system assertion is enable, then the HAL assertion can still be disabled by this Kconfig option.

Available options:

- Same as system assertion level (CONFIG_HAL_ASSERTION_EQUALS_SYSTEM)
- Disabled (CONFIG_HAL_ASSERTION_DISABLE)
- Silent (CONFIG_HAL_ASSERTION_SILENT)
- Enabled (CONFIG_HAL_ASSERTION_ENABLE)

CONFIG_HAL_LOG_LEVEL

HAL layer log verbosity

Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)

Specify how much output to see in HAL logs.

Available options:

- No output (CONFIG_HAL_LOG_LEVEL_NONE)
- Error (CONFIG_HAL_LOG_LEVEL_ERROR)
- Warning (CONFIG_HAL_LOG_LEVEL_WARN)
- Info (CONFIG_HAL_LOG_LEVEL_INFO)
- Debug (CONFIG_HAL_LOG_LEVEL_DEBUG)
- Verbose (CONFIG_HAL_LOG_LEVEL_VERBOSE)

CONFIG_HAL_SYSTIMER_USE_ROM_IMPL

Use ROM implementation of SysTimer HAL driver

Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)

Enable this flag to use HAL functions from ROM instead of ESP-IDF.

If keeping this as "n" in your project, you will have less free IRAM. If making this as "y" in your project, you will increase free IRAM, but you will lose the possibility to debug this module, and some new features will be added and bugs will be fixed in the IDF source but cannot be synced to ROM.

Default value:

- Yes (enabled)

CONFIG_HAL_WDT_USE_ROM_IMPL

Use ROM implementation of WDT HAL driver

Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)

Enable this flag to use HAL functions from ROM instead of ESP-IDF.

If keeping this as ”n” in your project, you will have less free IRAM. If making this as ”y” in your project, you will increase free IRAM, but you will lose the possibility to debug this module, and some new features will be added and bugs will be fixed in the IDF source but cannot be synced to ROM.

Default value:

- Yes (enabled)

CONFIG_HAL_ECDSA_GEN_SIG_CM

Enable countermeasure for ECDSA signature generation

Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)

Enable this option to apply the countermeasure for ECDSA signature operation This countermeasure masks the real ECDSA sign operation under dummy sign operations to add randomness in the generated power signature.

Default value:

- No (disabled)

Heap memory debugging Contains:

- *CONFIG_HEAP_ABORT_WHEN_ALLOCATION_FAILS*
- *CONFIG_HEAP_TASK_TRACKING*
- *CONFIG_HEAP_PLACE_FUNCTION_INTO_FLASH*
- *CONFIG_HEAP_CORRUPTION_DETECTION*
- *CONFIG_HEAP_TRACING_DEST*
- *CONFIG_HEAP_TRACING_STACK_DEPTH*
- *CONFIG_HEAP_USE_HOOKS*
- *CONFIG_HEAP_TRACE_HASH_MAP*
- *CONFIG_HEAP_TLSF_USE_ROM_IMPL*

CONFIG_HEAP_CORRUPTION_DETECTION

Heap corruption detection

Found in: Component config > Heap memory debugging

Enable heap poisoning features to detect heap corruption caused by out-of-bounds access to heap memory.

See the ”Heap Memory Debugging” page of the IDF documentation for a description of each level of heap corruption detection.

Available options:

- Basic (no poisoning) (CONFIG_HEAP_POISONING_DISABLED)
- Light impact (CONFIG_HEAP_POISONING_LIGHT)
- Comprehensive (CONFIG_HEAP_POISONING_COMPREHENSIVE)

CONFIG_HEAP_TRACING_DEST

Heap tracing

Found in: Component config > Heap memory debugging

Enables the heap tracing API defined in esp_heap_trace.h.

This function causes a moderate increase in IRAM code side and a minor increase in heap function (malloc/free/realloc) CPU overhead, even when the tracing feature is not used. So it's best to keep it disabled unless tracing is being used.

Available options:

- Disabled (CONFIG_HEAP_TRACING_OFF)
- Standalone (CONFIG_HEAP_TRACING_STANDALONE)
- Host-based (CONFIG_HEAP_TRACING_TOHOST)

CONFIG_HEAP_TRACING_STACK_DEPTH

Heap tracing stack depth

Found in: Component config > Heap memory debugging

Number of stack frames to save when tracing heap operation callers.

More stack frames uses more memory in the heap trace buffer (and slows down allocation), but can provide useful information.

CONFIG_HEAP_USE_HOOKS

Use allocation and free hooks

Found in: Component config > Heap memory debugging

Enable the user to implement function hooks triggered for each successful allocation and free.

CONFIG_HEAP_TASK_TRACKING

Enable heap task tracking

Found in: Component config > Heap memory debugging

Enables tracking the task responsible for each heap allocation.

This function depends on heap poisoning being enabled and adds four more bytes of overhead for each block allocated.

CONFIG_HEAP_TRACE_HASH_MAP

Use hash map mechanism to access heap trace records

Found in: Component config > Heap memory debugging

Enable this flag to use a hash map to increase performance in handling heap trace records.

Heap trace standalone supports storing records as a list, or a list + hash map.

Using only a list takes less memory, but calls to 'free' will get slower as the list grows. This is particularly affected when using HEAP_TRACE_ALL mode.

By using a list + hash map, calls to 'free' remain fast, at the cost of additional memory to store the hash map.

Default value:

- No (disabled) if *CONFIG_HEAP_TRACING_STANDALONE*

CONFIG_HEAP_TRACE_HASH_MAP_IN_EXT_RAM

Place hash map in external RAM

Found in: Component config > Heap memory debugging > CONFIG_HEAP_TRACE_HASH_MAP

When enabled this configuration forces the hash map to be placed in external RAM.

Default value:

- No (disabled) if *CONFIG_HEAP_TRACE_HASH_MAP*

CONFIG_HEAP_TRACE_HASH_MAP_SIZE

The number of entries in the hash map

Found in: Component config > Heap memory debugging > CONFIG_HEAP_TRACE_HASH_MAP

Defines the number of entries in the heap trace hashmap. Each entry takes 8 bytes. The bigger this number is, the better the performance. Recommended range: 200 - 2000.

Default value:

- 512 if *CONFIG_HEAP_TRACE_HASH_MAP*

CONFIG_HEAP_ABORT_WHEN_ALLOCATION_FAILS

Abort if memory allocation fails

Found in: Component config > Heap memory debugging

When enabled, if a memory allocation operation fails it will cause a system abort.

Default value:

- No (disabled)

CONFIG_HEAP_TLSF_USE_ROM_IMPL

Use ROM implementation of heap tlsf library

Found in: Component config > Heap memory debugging

Enable this flag to use heap functions from ROM instead of ESP-IDF.

If keeping this as ”n” in your project, you will have less free IRAM. If making this as ”y” in your project, you will increase free IRAM, but you will lose the possibility to debug this module, and some new features will be added and bugs will be fixed in the IDF source but cannot be synced to ROM.

Default value:

- Yes (enabled) if ESP_ROM_HAS_HEAP_TLSF

CONFIG_HEAP_PLACE_FUNCTION_INTO_FLASH

Force the entire heap component to be placed in flash memory

Found in: Component config > Heap memory debugging

Enable this flag to save up RAM space by placing the heap component in the flash memory

Note that it is only safe to enable this configuration if no functions from esp_heap_caps.h or esp_heap_trace.h are called from ISR.

IEEE 802.15.4 Contains:

- *CONFIG_IEEE802154_ENABLED*

CONFIG_IEEE802154_ENABLED

IEEE802154 Enable

Found in: Component config > IEEE 802.15.4

Default value:

- Yes (enabled) if SOC_IEEE802154_SUPPORTED

CONFIG_IEEE802154_RX_BUFFER_SIZE

The number of 802.15.4 receive buffers

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED

The number of 802.15.4 receive buffers

Range:

- from 2 to 100 if *CONFIG_IEEE802154_ENABLED*

Default value:

- 20 if *CONFIG_IEEE802154_ENABLED*

CONFIG_IEEE802154_CCA_MODE

Clear Channel Assessment (CCA) mode

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED

configure the CCA mode

Available options:

- Carrier sense only (CONFIG_IEEE802154_CCA_CARRIER)
  configure the CCA mode to Energy above threshold
- Energy above threshold (CONFIG_IEEE802154_CCA_ED)
  configure the CCA mode to Energy above threshold
- Carrier sense OR energy above threshold (CONFIG_IEEE802154_CCA_CARRIER_OR_ED)
  configure the CCA mode to Carrier sense OR energy above threshold
- Carrier sense AND energy above threshold (CONFIG_IEEE802154_CCA_CARRIER_AND_ED)
  configure the CCA mode to Carrier sense AND energy above threshold

CONFIG_IEEE802154_CCA_THRESHOLD

CCA detection threshold

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED

set the CCA threshold, in dB

Range:

- from -120 to 0 if *CONFIG_IEEE802154_ENABLED*

Default value:

- ”-60” if *CONFIG_IEEE802154_ENABLED*

CONFIG_IEEE802154_PENDING_TABLE_SIZE

Pending table size

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED

set the pending table size

Range:
- from 1 to 100 if *CONFIG_IEEE802154_ENABLED*

Default value:
- 20 if *CONFIG_IEEE802154_ENABLED*

CONFIG_IEEE802154_MULTI_PAN_ENABLE

Enable multi-pan feature for frame filter

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED

Enable IEEE802154 multi-pan

Default value:
- No (disabled) if *CONFIG_IEEE802154_ENABLED*

CONFIG_IEEE802154_TIMING_OPTIMIZATION

Enable throughput optimization

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED

Enabling this option increases throughput by ~5% at the expense of ~2.1k IRAM code size increase.

Default value:
- No (disabled) if *CONFIG_IEEE802154_ENABLED*

CONFIG_IEEE802154_SLEEP_ENABLE

Enable IEEE802154 light sleep

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED

Enabling this option allows the IEEE802.15.4 module to be powered down during automatic light sleep, which reduces current consumption.

Default value:
- No (disabled) if *CONFIG_PM_ENABLE* && *CONFIG_IEEE802154_ENABLED*

CONFIG_IEEE802154_DEBUG

Enable IEEE802154 Debug

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED

Enabling this option allows different kinds of IEEE802154 debug output. All IEEE802154 debug features increase the size of the final binary.

Default value:
- No (disabled) if *CONFIG_IEEE802154_ENABLED*

Contains:

- *CONFIG_IEEE802154_RECORD_ABORT*
- *CONFIG_IEEE802154_RECORD_CMD*
- *CONFIG_IEEE802154_RECORD_EVENT*
- *CONFIG_IEEE802154_RECORD_STATE*
- *CONFIG_IEEE802154_TXRX_STATISTIC*
- *CONFIG_IEEE802154_ASSERT*

CONFIG_IEEE802154_ASSERT

Enrich the assert information with IEEE802154 state and event

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED > CONFIG_IEEE802154_DEBUG

Enabling this option to add some probe codes in the driver, and these informations will be printed when assert.

Default value:

- No (disabled) if *CONFIG_IEEE802154_DEBUG*

CONFIG_IEEE802154_RECORD_EVENT

Enable record event information for debugging

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED > CONFIG_IEEE802154_DEBUG

Enabling this option to record event, when assert, the recorded event will be printed.

Default value:

- No (disabled) if *CONFIG_IEEE802154_DEBUG*

CONFIG_IEEE802154_RECORD_EVENT_SIZE

Record event table size

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED > CONFIG_IEEE802154_DEBUG > CONFIG_IEEE802154_RECORD_EVENT

set the record event table size

Range:

- from 1 to 50 if *CONFIG_IEEE802154_RECORD_EVENT*

Default value:

- 30 if *CONFIG_IEEE802154_RECORD_EVENT*

CONFIG_IEEE802154_RECORD_STATE

Enable record state information for debugging

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED > CONFIG_IEEE802154_DEBUG

Enabling this option to record state, when assert, the recorded state will be printed.

Default value:

- No (disabled) if *CONFIG_IEEE802154_DEBUG*

CONFIG_IEEE802154_RECORD_STATE_SIZE

Record state table size

Found in: Component config > IEEE 802.15.4 > CONFIG_IEEE802154_ENABLED > CONFIG_IEEE802154_DEBUG > CONFIG_IEEE802154_RECORD_STATE

set the record state table size

Range:

- from 1 to 50 if *CONFIG_IEEE802154_RECORD_STATE*

Default value:

- 10 if *CONFIG_IEEE802154_RECORD_STATE*

CONFIG_IEEE802154_RECORD_CMD

Enable record command information for debugging

Found in: *Component config* > *IEEE 802.15.4* > *CONFIG_IEEE802154_ENABLED* > *CONFIG_IEEE802154_DEBUG*

Enabling this option to record the command, when assert, the recorded command will be printed.

Default value:

- No (disabled) if *CONFIG_IEEE802154_DEBUG*

CONFIG_IEEE802154_RECORD_CMD_SIZE

Record command table size

Found in: *Component config* > *IEEE 802.15.4* > *CONFIG_IEEE802154_ENABLED* > *CONFIG_IEEE802154_DEBUG* > *CONFIG_IEEE802154_RECORD_CMD*

set the record command table size

Range:

- from 1 to 50 if *CONFIG_IEEE802154_RECORD_CMD*

Default value:

- 10 if *CONFIG_IEEE802154_RECORD_CMD*

CONFIG_IEEE802154_RECORD_ABORT

Enable record abort information for debugging

Found in: *Component config* > *IEEE 802.15.4* > *CONFIG_IEEE802154_ENABLED* > *CONFIG_IEEE802154_DEBUG*

Enabling this option to record the abort, when assert, the recorded abort will be printed.

Default value:

- No (disabled) if *CONFIG_IEEE802154_DEBUG*

CONFIG_IEEE802154_RECORD_ABORT_SIZE

Record abort table size

Found in: *Component config* > *IEEE 802.15.4* > *CONFIG_IEEE802154_ENABLED* > *CONFIG_IEEE802154_DEBUG* > *CONFIG_IEEE802154_RECORD_ABORT*

set the record abort table size

Range:

- from 1 to 50 if *CONFIG_IEEE802154_RECORD_ABORT*

Default value:

- 10 if *CONFIG_IEEE802154_RECORD_ABORT*

CONFIG_IEEE802154_TXRX_STATISTIC

Enable record tx/rx packets information for debugging

Found in: *Component config* > *IEEE 802.15.4* > *CONFIG_IEEE802154_ENABLED* > *CONFIG_IEEE802154_DEBUG*

Enabling this option to record the tx and rx

Default value:

- No (disabled) if *CONFIG_IEEE802154_DEBUG*

Log output Contains:

- *CONFIG_LOG_DEFAULT_LEVEL*
- *CONFIG_LOG_MASTER_LEVEL*
- *CONFIG_LOG_TIMESTAMP_SOURCE*
- *CONFIG_LOG_MAXIMUM_LEVEL*
- *CONFIG_LOG_COLORS*

CONFIG_LOG_DEFAULT_LEVEL

Default log verbosity

Found in: Component config > Log output

Specify how much output to see in logs by default. You can set lower verbosity level at runtime using esp_log_level_set function.

By default, this setting limits which log statements are compiled into the program. For example, selecting "Warning" would mean that changing log level to "Debug" at runtime will not be possible. To allow increasing log level above the default at runtime, see the next option.

Available options:

- No output (CONFIG_LOG_DEFAULT_LEVEL_NONE)
- Error (CONFIG_LOG_DEFAULT_LEVEL_ERROR)
- Warning (CONFIG_LOG_DEFAULT_LEVEL_WARN)
- Info (CONFIG_LOG_DEFAULT_LEVEL_INFO)
- Debug (CONFIG_LOG_DEFAULT_LEVEL_DEBUG)
- Verbose (CONFIG_LOG_DEFAULT_LEVEL_VERBOSE)

CONFIG_LOG_MAXIMUM_LEVEL

Maximum log verbosity

Found in: Component config > Log output

This config option sets the highest log verbosity that it's possible to select at runtime by calling esp_log_level_set(). This level may be higher than the default verbosity level which is set when the app starts up.

This can be used enable debugging output only at a critical point, for a particular tag, or to minimize startup time but then enable more logs once the firmware has loaded.

Note that increasing the maximum available log level will increase the firmware binary size.

This option only applies to logging from the app, the bootloader log level is fixed at compile time to the separate "Bootloader log verbosity" setting.

Available options:

- Same as default (CONFIG_LOG_MAXIMUM_EQUALS_DEFAULT)
- Error (CONFIG_LOG_MAXIMUM_LEVEL_ERROR)
- Warning (CONFIG_LOG_MAXIMUM_LEVEL_WARN)
- Info (CONFIG_LOG_MAXIMUM_LEVEL_INFO)
- Debug (CONFIG_LOG_MAXIMUM_LEVEL_DEBUG)
- Verbose (CONFIG_LOG_MAXIMUM_LEVEL_VERBOSE)

CONFIG_LOG_MASTER_LEVEL

Enable global master log level

Found in: Component config > Log output

Enables an additional global ”master” log level check that occurs before a log tag cache lookup. This is useful if you want to compile in a lot of logs that are selectable at runtime, but avoid the performance hit during periods where you don't want log output. Examples include remote log forwarding, or disabling logs during a time-critical or CPU-intensive section and re-enabling them later. Results in larger program size depending on number of logs compiled in.

If enabled, defaults to LOG_DEFAULT_LEVEL and can be set using esp_log_set_level_master(). This check takes precedence over ESP_LOG_LEVEL_LOCAL.

Default value:

- No (disabled)

CONFIG_LOG_COLORS

Use ANSI terminal colors in log output

Found in: Component config > Log output

Enable ANSI terminal color codes in bootloader output.

In order to view these, your terminal program must support ANSI color codes.

Default value:

- Yes (enabled)

CONFIG_LOG_TIMESTAMP_SOURCE

Log Timestamps

Found in: Component config > Log output

Choose what sort of timestamp is displayed in the log output:

- Milliseconds since boot is calulated from the RTOS tick count multiplied by the tick period. This time will reset after a software reboot. e.g. (90000)
- System time is taken from POSIX time functions which use the chip's RTC and high resoultion timers to maintain an accurate time. The system time is initialized to 0 on startup, it can be set with an SNTP sync, or with POSIX time functions. This time will not reset after a software reboot. e.g. (00:01:30.000)
- NOTE: Currently this will not get used in logging from binary blobs (i.e WiFi & Bluetooth libraries), these will always print milliseconds since boot.

Available options:

- Milliseconds Since Boot (CONFIG_LOG_TIMESTAMP_SOURCE_RTOS)
- System Time (CONFIG_LOG_TIMESTAMP_SOURCE_SYSTEM)

LWIP

Contains:

- *CONFIG_LWIP_CHECK_THREAD_SAFETY*
- *Checksums*
- *CONFIG_LWIP_DHCP_COARSE_TIMER_SECS*
- *DHCP server*
- *CONFIG_LWIP_DHCP_OPTIONS_LEN*
- *CONFIG_LWIP_DHCP_DISABLE_CLIENT_ID*
- *CONFIG_LWIP_DHCP_DISABLE_VENDOR_CLASS_ID*
- *CONFIG_LWIP_DHCP_DOES_ARP_CHECK*

- *CONFIG_LWIP_DHCP_RESTORE_LAST_IP*
- *DNS*
- *CONFIG_LWIP_PPP_CHAP_SUPPORT*
- *CONFIG_LWIP_L2_TO_L3_COPY*
- *CONFIG_LWIP_IPV6_DHCP6*
- *CONFIG_LWIP_IP4_FRAG*
- *CONFIG_LWIP_IP6_FRAG*
- *CONFIG_LWIP_IP_FORWARD*
- *CONFIG_LWIP_NETBUF_RECVINFO*
- *CONFIG_LWIP_IPV4*
- *CONFIG_LWIP_AUTOIP*
- *CONFIG_LWIP_IPV6*
- *CONFIG_LWIP_ENABLE_LCP_ECHO*
- *CONFIG_LWIP_ESP_LWIP_ASSERT*
- *CONFIG_LWIP_DEBUG*
- *CONFIG_LWIP_IRAM_OPTIMIZATION*
- *CONFIG_LWIP_EXTRA_IRAM_OPTIMIZATION*
- *CONFIG_LWIP_ENABLE*
- *CONFIG_LWIP_STATS*
- *CONFIG_LWIP_TIMERS_ONDEMAND*
- *CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES*
- *CONFIG_LWIP_PPP_MPPE_SUPPORT*
- *CONFIG_LWIP_PPP_MSCHAP_SUPPORT*
- *CONFIG_LWIP_PPP_NOTIFY_PHASE_SUPPORT*
- *CONFIG_LWIP_PPP_PAP_SUPPORT*
- *CONFIG_LWIP_PPP_DEBUG_ON*
- *CONFIG_LWIP_PPP_SERVER_SUPPORT*
- *CONFIG_LWIP_PPP_SUPPORT*
- *CONFIG_LWIP_IP4_REASSEMBLY*
- *CONFIG_LWIP_IP6_REASSEMBLY*
- *CONFIG_LWIP_SLIP_SUPPORT*
- *CONFIG_LWIP_SO_LINGER*
- *CONFIG_LWIP_SO_RCVBUF*
- *CONFIG_LWIP_SO_REUSE*
- *CONFIG_LWIP_NETIF_STATUS_CALLBACK*
- *CONFIG_LWIP_TCPIP_CORE_LOCKING*
- *CONFIG_LWIP_NETIF_API*
- *CONFIG_LWIP_PPP_VJ_HEADER_COMPRESSION*
- *Hooks*
- *ICMP*
- *CONFIG_LWIP_LOCAL_HOSTNAME*
- *CONFIG_LWIP_ND6*
- *LWIP RAW API*
- *CONFIG_LWIP_TCPIP_TASK_PRIO*
- *CONFIG_LWIP_IPV6_ND6_NUM_ROUTERS*
- *CONFIG_LWIP_IPV6_ND6_NUM_DESTINATIONS*
- *CONFIG_LWIP_IPV6_ND6_NUM_NEIGHBORS*
- *CONFIG_LWIP_IPV6_ND6_NUM_PREFIXES*
- *CONFIG_LWIP_IPV6_MEMP_NUM_ND6_QUEUE*
- *CONFIG_LWIP_MAX_SOCKETS*
- *CONFIG_LWIP_BRIDGEIF_MAX_PORTS*
- *CONFIG_LWIP_NUM_NETIF_CLIENT_DATA*
- *CONFIG_LWIP_ESP_GRATUITOUS_ARP*
- *CONFIG_LWIP_ESP_MLDV6_REPORT*
- *SNTP*
- *CONFIG_LWIP_USE_ONLY_LWIP_SELECT*
- *CONFIG_LWIP_NETIF_LOOPBACK*
- *TCP*

- *CONFIG_LWIP_TCPIP_TASK_AFFINITY*
- *CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*
- *CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*
- *CONFIG_LWIP_IP_REASS_MAX_PBUFS*
- *CONFIG_LWIP_IP_DEFAULT_TTL*
- *UDP*
- *CONFIG_LWIP_IPV6_RDNSS_MAX_DNS_SERVERS*
- *CONFIG_LWIP_USE_EXTERNAL_MBEDTLS*

CONFIG_LWIP_ENABLE

Enable LwIP stack

Found in: Component config > LWIP

Builds normally if selected. Excludes LwIP from build if unselected, even if it is a dependency of a component or application. Some applications can switch their IP stacks, e.g., when switching between chip and Linux targets (LwIP stack vs. Linux IP stack). Since the LwIP dependency cannot easily be excluded based on a Kconfig option, it has to be a dependency in all cases. This switch allows the LwIP stack to be built selectively, even if it is a dependency.

Default value:

- Yes (enabled)

CONFIG_LWIP_LOCAL_HOSTNAME

Local netif hostname

Found in: Component config > LWIP

The default name this device will report to other devices on the network. Could be updated at runtime with esp_netif_set_hostname()

Default value:

- ”espressif”

CONFIG_LWIP_NETIF_API

Enable usage of standard POSIX APIs in LWIP

Found in: Component config > LWIP

If this feature is enabled, standard POSIX APIs: if_indextoname(), if_nametoindex() could be used to convert network interface index to name instead of IDF specific esp-netif APIs (such as esp_netif_get_netif_impl_name())

Default value:

- No (disabled)

CONFIG_LWIP_TCPIP_TASK_PRIO

LWIP TCP/IP Task Priority

Found in: Component config > LWIP

LWIP tcpip task priority. In case of high throughput, this parameter could be changed up to (config-MAX_PRIORITIES-1).

Range:

- from 1 to 24

Default value:

- 18

CONFIG_LWIP_TCPIP_CORE_LOCKING

Enable tcpip core locking

Found in: Component config > LWIP

If Enable tcpip core locking,Creates a global mutex that is held during TCPIP thread operations.Can be locked by client code to perform lwIP operations without changing into TCPIP thread using callbacks. See LOCK_TCPIP_CORE() and UNLOCK_TCPIP_CORE().

If disable tcpip core locking,TCP IP will perform tasks through context switching

Default value:

- No (disabled)

CONFIG_LWIP_TCPIP_CORE_LOCKING_INPUT

Enable tcpip core locking input

Found in: Component config > LWIP > CONFIG_LWIP_TCPIP_CORE_LOCKING

when LWIP_TCPIP_CORE_LOCKING is enabled, this lets tcpip_input() grab the mutex for input packets as well, instead of allocating a message and passing it to tcpip_thread.

Default value:

- No (disabled) if *CONFIG_LWIP_TCPIP_CORE_LOCKING*

CONFIG_LWIP_CHECK_THREAD_SAFETY

Checks that lwip API runs in expected context

Found in: Component config > LWIP

Enable to check that the project does not violate lwip thread safety. If enabled, all lwip functions that require thread awareness run an assertion to verify that the TCP/IP core functionality is either locked or accessed from the correct thread.

Default value:

- No (disabled)

CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES

Enable mDNS queries in resolving host name

Found in: Component config > LWIP

If this feature is enabled, standard API such as gethostbyname support .local addresses by sending one shot multicast mDNS query

Default value:

- Yes (enabled)

CONFIG_LWIP_L2_TO_L3_COPY

Enable copy between Layer2 and Layer3 packets

Found in: Component config > LWIP

If this feature is enabled, all traffic from layer2(WIFI Driver) will be copied to a new buffer before sending it to layer3(LWIP stack), freeing the layer2 buffer. Please be notified that the total layer2 receiving buffer is fixed and ESP32 currently supports 25 layer2 receiving buffer, when layer2 buffer runs out of memory, then the incoming packets will be dropped in hardware. The layer3 buffer is allocated from the heap, so the total layer3 receiving buffer depends on the available heap size, when heap runs out of memory, no copy will be sent to layer3 and packet will be dropped in layer2. Please make sure you fully understand the impact of this feature before enabling it.

Default value:

- No (disabled)

CONFIG_LWIP_IRAM_OPTIMIZATION

Enable LWIP IRAM optimization

Found in: Component config > LWIP

If this feature is enabled, some functions relating to RX/TX in LWIP will be put into IRAM, it can improve UDP/TCP throughput by >10% for single core mode, it doesn't help too much for dual core mode. On the other hand, it needs about 10KB IRAM for these optimizations.

If this feature is disabled, all lwip functions will be put into FLASH.

Default value:

- No (disabled)

CONFIG_LWIP_EXTRA_IRAM_OPTIMIZATION

Enable LWIP IRAM optimization for TCP part

Found in: Component config > LWIP

If this feature is enabled, some tcp part functions relating to RX/TX in LWIP will be put into IRAM, it can improve TCP throughput. On the other hand, it needs about 17KB IRAM for these optimizations.

Default value:

- No (disabled)

CONFIG_LWIP_TIMERS_ONDEMAND

Enable LWIP Timers on demand

Found in: Component config > LWIP

If this feature is enabled, IGMP and MLD6 timers will be activated only when joining groups or receiving QUERY packets.

This feature will reduce the power consumption for applications which do not use IGMP and MLD6.

Default value:

- Yes (enabled)

CONFIG_LWIP_ND6

LWIP NDP6 Enable/Disable

Found in: Component config > LWIP

This option is used to disable the Network Discovery Protocol (NDP) if it is not required. Please use this option with caution, as the NDP is essential for IPv6 functionality within a local network.

Default value:

- Yes (enabled)

CONFIG_LWIP_FORCE_ROUTER_FORWARDING

LWIP Force Router Forwarding Enable/Disable

Found in: Component config > LWIP > CONFIG_LWIP_ND6

This option is used to set the the router flag for the NA packets. When enabled, the router flag in NA packet will always set to 1, otherwise, never set router flag for NA packets.

Default value:

- No (disabled)

CONFIG_LWIP_MAX_SOCKETS

Max number of open sockets

Found in: Component config > LWIP

The practical maximum limit is determined by available heap memory at runtime.

Sockets take up a certain amount of memory, and allowing fewer sockets to be open at the same time conserves memory. Specify the maximum amount of sockets here. The valid value is from 1 to 253. If using value above 61, update CMakeLists defining FD_SETSIZE to the number of sockets used plus the expected open files (minimum of +3 for stdout, stderr and stdin).

Range:
- from 1 to 253

Default value:
- 10

CONFIG_LWIP_USE_ONLY_LWIP_SELECT

Support LWIP socket select() only (DEPRECATED)

Found in: Component config > LWIP

This option is deprecated. Do not use this option, use VFS_SUPPORT_SELECT instead.

Default value:
- No (disabled)

CONFIG_LWIP_SO_LINGER

Enable SO_LINGER processing

Found in: Component config > LWIP

Enabling this option allows SO_LINGER processing. l_onoff = 1,l_linger can set the timeout.

If l_linger=0, When a connection is closed, TCP will terminate the connection. This means that TCP will discard any data packets stored in the socket send buffer and send an RST to the peer.

If l_linger!=0,Then closesocket() calls to block the process until the remaining data packets has been sent or timed out.

Default value:
- No (disabled)

CONFIG_LWIP_SO_REUSE

Enable SO_REUSEADDR option

Found in: Component config > LWIP

Enabling this option allows binding to a port which remains in TIME_WAIT.

Default value:
- Yes (enabled)

CONFIG_LWIP_SO_REUSE_RXTOALL

SO_REUSEADDR copies broadcast/multicast to all matches

Found in: Component config > LWIP > CONFIG_LWIP_SO_REUSE

Enabling this option means that any incoming broadcast or multicast packet will be copied to all of the local sockets that it matches (may be more than one if SO_REUSEADDR is set on the socket.)

This increases memory overhead as the packets need to be copied, however they are only copied per matching socket. You can safely disable it if you don't plan to receive broadcast or multicast traffic on more than one socket at a time.

Default value:

- Yes (enabled)

CONFIG_LWIP_SO_RCVBUF

Enable SO_RCVBUF option

Found in: Component config > LWIP

Enabling this option allows checking for available data on a netconn.

Default value:

- No (disabled)

CONFIG_LWIP_NETBUF_RECVINFO

Enable IP_PKTINFO option

Found in: Component config > LWIP

Enabling this option allows checking for the destination address of a received IPv4 Packet.

Default value:

- No (disabled)

CONFIG_LWIP_IP_DEFAULT_TTL

The value for Time-To-Live used by transport layers

Found in: Component config > LWIP

Set value for Time-To-Live used by transport layers.

Range:

- from 1 to 255

Default value:

- 64

CONFIG_LWIP_IP4_FRAG

Enable fragment outgoing IP4 packets

Found in: Component config > LWIP

Enabling this option allows fragmenting outgoing IP4 packets if their size exceeds MTU.

Default value:

- Yes (enabled)

CONFIG_LWIP_IP6_FRAG

Enable fragment outgoing IP6 packets

Found in: Component config > LWIP

Enabling this option allows fragmenting outgoing IP6 packets if their size exceeds MTU.

Default value:

- Yes (enabled)

CONFIG_LWIP_IP4_REASSEMBLY

Enable reassembly incoming fragmented IP4 packets

Found in: Component config > LWIP

Enabling this option allows reassemblying incoming fragmented IP4 packets.

Default value:

- No (disabled)

CONFIG_LWIP_IP6_REASSEMBLY

Enable reassembly incoming fragmented IP6 packets

Found in: Component config > LWIP

Enabling this option allows reassemblying incoming fragmented IP6 packets.

Default value:

- No (disabled)

CONFIG_LWIP_IP_REASS_MAX_PBUFS

The maximum amount of pbufs waiting to be reassembled

Found in: Component config > LWIP

Set the maximum amount of pbufs waiting to be reassembled.

Range:

- from 10 to 100

Default value:

- 10

CONFIG_LWIP_IP_FORWARD

Enable IP forwarding

Found in: Component config > LWIP

Enabling this option allows packets forwarding across multiple interfaces.

Default value:

- No (disabled)

CONFIG_LWIP_IPV4_NAPT

Enable NAT

Found in: Component config > LWIP > CONFIG_LWIP_IP_FORWARD

Enabling this option allows Network Address and Port Translation.

Default value:

- No (disabled) if *CONFIG_LWIP_IP_FORWARD*

CONFIG_LWIP_IPV4_NAPT_PORTMAP

Enable NAT Port Mapping

Found in: Component config > LWIP > CONFIG_LWIP_IP_FORWARD > CONFIG_LWIP_IPV4_NAPT

Enabling this option allows Port Forwarding or Port mapping.

Default value:

- Yes (enabled) if *CONFIG_LWIP_IPV4_NAPT*

CONFIG_LWIP_STATS

Enable LWIP statistics

Found in: Component config > LWIP

Enabling this option allows LWIP statistics

Default value:

- No (disabled)

CONFIG_LWIP_ESP_GRATUITOUS_ARP

Send gratuitous ARP periodically

Found in: Component config > LWIP

Enable this option allows to send gratuitous ARP periodically.

This option solve the compatibility issues.If the ARP table of the AP is old, and the AP doesn't send ARP request to update it's ARP table, this will lead to the STA sending IP packet fail. Thus we send gratuitous ARP periodically to let AP update it's ARP table.

Default value:

- Yes (enabled)

CONFIG_LWIP_GARP_TMR_INTERVAL

GARP timer interval(seconds)

Found in: Component config > LWIP > CONFIG_LWIP_ESP_GRATUITOUS_ARP

Set the timer interval for gratuitous ARP. The default value is 60s

Default value:

- 60

CONFIG_LWIP_ESP_MLDV6_REPORT

Send mldv6 report periodically

Found in: Component config > LWIP

Enable this option allows to send mldv6 report periodically.

This option solve the issue that failed to receive multicast data. Some routers fail to forward multicast packets. To solve this problem, send multicast mdlv6 report to routers regularly.

Default value:

- Yes (enabled)

CONFIG_LWIP_MLDV6_TMR_INTERVAL

mldv6 report timer interval(seconds)

Found in: Component config > LWIP > CONFIG_LWIP_ESP_MLDV6_REPORT

Set the timer interval for mldv6 report. The default value is 30s

Default value:

- 40

CONFIG_LWIP_TCPIP_RECVMBOX_SIZE

TCPIP task receive mail box size

Found in: Component config > LWIP

Set TCPIP task receive mail box size. Generally bigger value means higher throughput but more memory. The value should be bigger than UDP/TCP mail box size.

Range:

- from 6 to 1024 if *CONFIG_LWIP_WND_SCALE*

Default value:

- 32

CONFIG_LWIP_DHCP_DOES_ARP_CHECK

DHCP: Perform ARP check on any offered address

Found in: Component config > LWIP

Enabling this option performs a check (via ARP request) if the offered IP address is not already in use by another host on the network.

Default value:

- Yes (enabled)

CONFIG_LWIP_DHCP_DISABLE_CLIENT_ID

DHCP: Disable Use of HW address as client identification

Found in: Component config > LWIP

This option could be used to disable DHCP client identification with its MAC address. (Client id is used by DHCP servers to uniquely identify clients and are included in the DHCP packets as an option 61) Set this option to ”y” in order to exclude option 61 from DHCP packets.

Default value:

- No (disabled)

CONFIG_LWIP_DHCP_DISABLE_VENDOR_CLASS_ID

DHCP: Disable Use of vendor class identification

Found in: Component config > LWIP

This option could be used to disable DHCP client vendor class identification. Set this option to ”y” in order to exclude option 60 from DHCP packets.

Default value:

- Yes (enabled)

CONFIG_LWIP_DHCP_RESTORE_LAST_IP

DHCP: Restore last IP obtained from DHCP server

Found in: Component config > LWIP

When this option is enabled, DHCP client tries to re-obtain last valid IP address obtained from DHCP server. Last valid DHCP configuration is stored in nvs and restored after reset/power-up. If IP is still available, there is no need for sending discovery message to DHCP server and save some time.

Default value:

- No (disabled)

CONFIG_LWIP_DHCP_OPTIONS_LEN

DHCP total option length

Found in: Component config > LWIP

Set total length of outgoing DHCP option msg. Generally bigger value means it can carry more options and values. If your code meets LWIP_ASSERT due to option value is too long. Please increase the LWIP_DHCP_OPTIONS_LEN value.

Range:

- from 68 to 255

Default value:

- 68

CONFIG_LWIP_NUM_NETIF_CLIENT_DATA

Number of clients store data in netif

Found in: Component config > LWIP

Number of clients that may store data in client_data member array of struct netif.

Range:

- from 0 to 256

Default value:

- 0

CONFIG_LWIP_DHCP_COARSE_TIMER_SECS

DHCP coarse timer interval(s)

Found in: Component config > LWIP

Set DHCP coarse interval in seconds. A higher value will be less precise but cost less power consumption.

Range:

- from 1 to 10

Default value:

- 1

DHCP server Contains:

- *CONFIG_LWIP_DHCPS*

CONFIG_LWIP_DHCPS

DHCPS: Enable IPv4 Dynamic Host Configuration Protocol Server (DHCPS)

Found in: Component config > LWIP > DHCP server

Enabling this option allows the device to run the DHCP server (to dynamically assign IPv4 addresses to clients).

Default value:

- Yes (enabled)

CONFIG_LWIP_DHCPS_LEASE_UNIT

Multiplier for lease time, in seconds

Found in: Component config > LWIP > DHCP server > CONFIG_LWIP_DHCPS

The DHCP server is calculating lease time multiplying the sent and received times by this number of seconds per unit. The default is 60, that equals one minute.

Range:

- from 1 to 3600

Default value:

- 60

CONFIG_LWIP_DHCPS_MAX_STATION_NUM

Maximum number of stations

Found in: Component config > LWIP > DHCP server > CONFIG_LWIP_DHCPS

The maximum number of DHCP clients that are connected to the server. After this number is exceeded, DHCP server removes of the oldest device from it's address pool, without notification.

Range:

- from 1 to 64

Default value:

- 8

CONFIG_LWIP_DHCPS_STATIC_ENTRIES

Enable ARP static entries

Found in: Component config > LWIP > DHCP server > CONFIG_LWIP_DHCPS

Enabling this option allows DHCP server to support temporary static ARP entries for DHCP Client. This will help the DHCP server to send the DHCP OFFER and DHCP ACK using IP unicast.

Default value:

- Yes (enabled)

CONFIG_LWIP_AUTOIP

Enable IPV4 Link-Local Addressing (AUTOIP)

Found in: Component config > LWIP

Enabling this option allows the device to self-assign an address in the 169.256/16 range if none is assigned statically or via DHCP.

See RFC 3927.

Default value:

- No (disabled)

Contains:

- *CONFIG_LWIP_AUTOIP_TRIES*
- *CONFIG_LWIP_AUTOIP_MAX_CONFLICTS*
- *CONFIG_LWIP_AUTOIP_RATE_LIMIT_INTERVAL*

CONFIG_LWIP_AUTOIP_TRIES

DHCP Probes before self-assigning IPv4 LL address

Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP

DHCP client will send this many probes before self-assigning a link local address.

From LWIP help: "This can be set as low as 1 to get an AutoIP address very quickly, but you should be prepared to handle a changing IP address when DHCP overrides AutoIP." (In the case of ESP-IDF, this means multiple SYSTEM_EVENT_STA_GOT_IP events.)

Range:
- from 1 to 100 if *CONFIG_LWIP_AUTOIP*

Default value:
- 2 if *CONFIG_LWIP_AUTOIP*

CONFIG_LWIP_AUTOIP_MAX_CONFLICTS

Max IP conflicts before rate limiting

Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP

If the AUTOIP functionality detects this many IP conflicts while self-assigning an address, it will go into a rate limited mode.

Range:
- from 1 to 100 if *CONFIG_LWIP_AUTOIP*

Default value:
- 9 if *CONFIG_LWIP_AUTOIP*

CONFIG_LWIP_AUTOIP_RATE_LIMIT_INTERVAL

Rate limited interval (seconds)

Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP

If rate limiting self-assignment requests, wait this long between each request.

Range:
- from 5 to 120 if *CONFIG_LWIP_AUTOIP*

Default value:
- 20 if *CONFIG_LWIP_AUTOIP*

CONFIG_LWIP_IPV4

Enable IPv4

Found in: Component config > LWIP

Enable IPv4 stack. If you want to use IPv6 only TCP/IP stack, disable this.

Default value:
- Yes (enabled)

CONFIG_LWIP_IPV6

Enable IPv6

Found in: Component config > LWIP

Enable IPv6 function. If not use IPv6 function, set this option to n. If disabling LWIP_IPV6 then some other components (asio) will no longer be available.

Default value:

- Yes (enabled)

CONFIG_LWIP_IPV6_AUTOCONFIG

Enable IPV6 stateless address autoconfiguration (SLAAC)

Found in: Component config > LWIP > CONFIG_LWIP_IPV6

Enabling this option allows the devices to IPV6 stateless address autoconfiguration (SLAAC).

See RFC 4862.

Default value:

- No (disabled)

CONFIG_LWIP_IPV6_NUM_ADDRESSES

Number of IPv6 addresses on each network interface

Found in: Component config > LWIP > CONFIG_LWIP_IPV6

The maximum number of IPv6 addresses on each interface. Any additional addresses will be discarded.

Default value:

- 3

CONFIG_LWIP_IPV6_FORWARD

Enable IPv6 forwarding between interfaces

Found in: Component config > LWIP > CONFIG_LWIP_IPV6

Forwarding IPv6 packets between interfaces is only required when acting as a router.

Default value:

- No (disabled)

CONFIG_LWIP_IPV6_RDNSS_MAX_DNS_SERVERS

Use IPv6 Router Advertisement Recursive DNS Server Option

Found in: Component config > LWIP

Use IPv6 Router Advertisement Recursive DNS Server Option (as per RFC 6106) to copy a defined maximum number of DNS servers to the DNS module. Set this option to a number of desired DNS servers advertised in the RA protocol. This feature is disabled when set to 0.

Default value:

- 0 if *CONFIG_LWIP_IPV6_AUTOCONFIG*

CONFIG_LWIP_IPV6_DHCP6

Enable DHCPv6 stateless address autoconfiguration

Found in: Component config > LWIP

Enable DHCPv6 for IPv6 stateless address autoconfiguration. Note that the dhcpv6 client has to be started using dhcp6_enable_stateless(netif); Note that the stateful address autoconfiguration is not supported.

Default value:

- No (disabled) if *CONFIG_LWIP_IPV6_AUTOCONFIG*

CONFIG_LWIP_NETIF_STATUS_CALLBACK

Enable status callback for network interfaces

Found in: Component config > LWIP

Enable callbacks when the network interface is up/down and addresses are changed.

Default value:

- No (disabled)

CONFIG_LWIP_NETIF_LOOPBACK

Support per-interface loopback

Found in: Component config > LWIP

Enabling this option means that if a packet is sent with a destination address equal to the interface's own IP address, it will "loop back" and be received by this interface. Disabling this option disables support of loopback interface in lwIP

Default value:

- Yes (enabled)

Contains:

- *CONFIG_LWIP_LOOPBACK_MAX_PBUFS*

CONFIG_LWIP_LOOPBACK_MAX_PBUFS

Max queued loopback packets per interface

Found in: Component config > LWIP > CONFIG_LWIP_NETIF_LOOPBACK

Configure the maximum number of packets which can be queued for loopback on a given interface. Reducing this number may cause packets to be dropped, but will avoid filling memory with queued packet data.

Range:

- from 0 to 16

Default value:

- 8

TCP

Contains:

- *CONFIG_LWIP_TCP_WND_DEFAULT*
- *CONFIG_LWIP_TCP_SND_BUF_DEFAULT*
- *CONFIG_LWIP_TCP_ACCEPTMBOX_SIZE*
- *CONFIG_LWIP_TCP_RECVMBOX_SIZE*
- *CONFIG_LWIP_TCP_RTO_TIME*
- *CONFIG_LWIP_MAX_ACTIVE_TCP*
- *CONFIG_LWIP_TCP_FIN_WAIT_TIMEOUT*

- *CONFIG_LWIP_MAX_LISTENING_TCP*
- *CONFIG_LWIP_TCP_MAXRTX*
- *CONFIG_LWIP_TCP_SYNMAXRTX*
- *CONFIG_LWIP_TCP_MSL*
- *CONFIG_LWIP_TCP_MSS*
- *CONFIG_LWIP_TCP_OVERSIZE*
- *CONFIG_LWIP_TCP_QUEUE_OOSEQ*
- *CONFIG_LWIP_WND_SCALE*
- *CONFIG_LWIP_TCP_HIGH_SPEED_RETRANSMISSION*
- *CONFIG_LWIP_TCP_TMR_INTERVAL*

CONFIG_LWIP_MAX_ACTIVE_TCP

Maximum active TCP Connections

Found in: Component config > LWIP > TCP

The maximum number of simultaneously active TCP connections. The practical maximum limit is determined by available heap memory at runtime.

Changing this value by itself does not substantially change the memory usage of LWIP, except for preventing new TCP connections after the limit is reached.

Range:
- from 1 to 1024

Default value:
- 16

CONFIG_LWIP_MAX_LISTENING_TCP

Maximum listening TCP Connections

Found in: Component config > LWIP > TCP

The maximum number of simultaneously listening TCP connections. The practical maximum limit is determined by available heap memory at runtime.

Changing this value by itself does not substantially change the memory usage of LWIP, except for preventing new listening TCP connections after the limit is reached.

Range:
- from 1 to 1024

Default value:
- 16

CONFIG_LWIP_TCP_HIGH_SPEED_RETRANSMISSION

TCP high speed retransmissions

Found in: Component config > LWIP > TCP

Speed up the TCP retransmission interval. If disabled, it is recommended to change the number of SYN retransmissions to 6, and TCP initial rto time to 3000.

Default value:
- Yes (enabled)

CONFIG_LWIP_TCP_MAXRTX

Maximum number of retransmissions of data segments

Found in: Component config > LWIP > TCP

Set maximum number of retransmissions of data segments.

Range:
- from 3 to 12

Default value:
- 12

CONFIG_LWIP_TCP_SYNMAXRTX

Maximum number of retransmissions of SYN segments

Found in: Component config > LWIP > TCP

Set maximum number of retransmissions of SYN segments.

Range:
- from 3 to 12

Default value:
- 12

CONFIG_LWIP_TCP_MSS

Maximum Segment Size (MSS)

Found in: Component config > LWIP > TCP

Set maximum segment size for TCP transmission.

Can be set lower to save RAM, the default value 1460(ipv4)/1440(ipv6) will give best throughput. IPv4 TCP_MSS Range: 576 <= TCP_MSS <= 1460 IPv6 TCP_MSS Range: 1220<= TCP_MSS <= 1440

Range:
- from 536 to 1460

Default value:
- 1440

CONFIG_LWIP_TCP_TMR_INTERVAL

TCP timer interval(ms)

Found in: Component config > LWIP > TCP

Set TCP timer interval in milliseconds.

Can be used to speed connections on bad networks. A lower value will redeliver unacked packets faster.

Default value:
- 250

CONFIG_LWIP_TCP_MSL

Maximum segment lifetime (MSL)

Found in: Component config > LWIP > TCP

Set maximum segment lifetime in milliseconds.

Default value:
- 60000

CONFIG_LWIP_TCP_FIN_WAIT_TIMEOUT

Maximum FIN segment lifetime

Found in: Component config > LWIP > TCP

Set maximum segment lifetime in milliseconds.

Default value:

- 20000

CONFIG_LWIP_TCP_SND_BUF_DEFAULT

Default send buffer size

Found in: Component config > LWIP > TCP

Set default send buffer size for new TCP sockets.

Per-socket send buffer size can be changed at runtime with lwip_setsockopt(s, TCP_SNDBUF, ...).

This value must be at least 2x the MSS size, and the default is 4x the default MSS size.

Setting a smaller default SNDBUF size can save some RAM, but will decrease performance.

Range:

- from 2440 to 1024000 if *CONFIG_LWIP_WND_SCALE*

Default value:

- 5760

CONFIG_LWIP_TCP_WND_DEFAULT

Default receive window size

Found in: Component config > LWIP > TCP

Set default TCP receive window size for new TCP sockets.

Per-socket receive window size can be changed at runtime with lwip_setsockopt(s, TCP_WINDOW, ...).

Setting a smaller default receive window size can save some RAM, but will significantly decrease performance.

Range:

- from 2440 to 1024000 if *CONFIG_LWIP_WND_SCALE*

Default value:

- 5760

CONFIG_LWIP_TCP_RECVMBOX_SIZE

Default TCP receive mail box size

Found in: Component config > LWIP > TCP

Set TCP receive mail box size. Generally bigger value means higher throughput but more memory. The recommended value is: LWIP_TCP_WND_DEFAULT/TCP_MSS + 2, e.g. if LWIP_TCP_WND_DEFAULT=14360, TCP_MSS=1436, then the recommended receive mail box size is (14360/1436 + 2) = 12.

TCP receive mail box is a per socket mail box, when the application receives packets from TCP socket, LWIP core firstly posts the packets to TCP receive mail box and the application then fetches the packets from mail box. It means LWIP can caches maximum LWIP_TCP_RECCVMBOX_SIZE packets for each TCP socket, so the maximum possible cached TCP packets for all TCP sockets is LWIP_TCP_RECCVMBOX_SIZE multiples the maximum TCP socket number. In other words, the bigger LWIP_TCP_RECVMBOX_SIZE means more memory. On the other hand, if the receive mail box is too small, the mail box may be full. If the mail box is full, the LWIP drops the packets. So generally we need to make sure the TCP receive mail box is big enough to avoid packet drop between LWIP core and application.

Range:

- from 6 to 1024 if *CONFIG_LWIP_WND_SCALE*

Default value:

- 6

CONFIG_LWIP_TCP_ACCEPTMBOX_SIZE

Default TCP accept mail box size

Found in: Component config > LWIP > TCP

Set TCP accept mail box size. Generally bigger value means supporting larger backlogs but more memory. The recommended value is 6, but applications can set it to a lower value if listening servers are meant to have a smaller backlog.

TCP accept mail box is a per socket mail box, when the application listens for connections with a given listening TCP socket. If the mailbox is full, LWIP will send a RST packet and the client will fail to connect.

Range:
- from 1 to 255 if *CONFIG_LWIP_WND_SCALE*

Default value:
- 6

CONFIG_LWIP_TCP_QUEUE_OOSEQ

Queue incoming out-of-order segments

Found in: Component config > LWIP > TCP

Queue incoming out-of-order segments for later use.

Disable this option to save some RAM during TCP sessions, at the expense of increased retransmissions if segments arrive out of order.

Default value:
- Yes (enabled)

CONFIG_LWIP_TCP_OOSEQ_TIMEOUT

Timeout for each pbuf queued in TCP OOSEQ, in RTOs.

Found in: Component config > LWIP > TCP > CONFIG_LWIP_TCP_QUEUE_OOSEQ

The timeout value is TCP_OOSEQ_TIMEOUT * RTO.

Range:
- from 1 to 30

Default value:
- 6

CONFIG_LWIP_TCP_OOSEQ_MAX_PBUFS

The maximum number of pbufs queued on OOSEQ per pcb

Found in: Component config > LWIP > TCP > CONFIG_LWIP_TCP_QUEUE_OOSEQ

If LWIP_TCP_OOSEQ_MAX_PBUFS = 0, TCP will not control the number of OOSEQ pbufs.

In a poor network environment, many out-of-order tcp pbufs will be received. These out-of-order pbufs will be cached in the TCP out-of-order queue which will cause Wi-Fi/Ethernet fail to release RX buffer in time. It is possible that all RX buffers for MAC layer are used by OOSEQ.

Control the number of out-of-order pbufs to ensure that the MAC layer has enough RX buffer to receive packets.

In the Wi-Fi scenario, recommended OOSEQ PBUFS Range: 0 <= TCP_OOSEQ_MAX_PBUFS <= CONFIG_ESP_WIFI_DYNAMIC_RX_BUFFER_NUM/(MAX_TCP_NUMBER + 1)

In the Ethernet scenario,recommended Ethernet OOSEQ PBUFS Range: 0 <= TCP_OOSEQ_MAX_PBUFS <= CONFIG_ETH_DMA_RX_BUFFER_NUM/(MAX_TCP_NUMBER + 1)

Within the recommended value range, the larger the value, the better the performance.

MAX_TCP_NUMBER represent Maximum number of TCP connections in Wi-Fi(STA+SoftAP) and Ethernet scenario.

Range:

- from 0 to 12

Default value:

- 0 if *CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP* && *CONFIG_LWIP_TCP_QUEUE_OOSEQ*

CONFIG_LWIP_TCP_SACK_OUT

Support sending selective acknowledgements

Found in: Component config > LWIP > TCP > CONFIG_LWIP_TCP_QUEUE_OOSEQ

TCP will support sending selective acknowledgements (SACKs).

Default value:

- No (disabled)

CONFIG_LWIP_TCP_OVERSIZE

Pre-allocate transmit PBUF size

Found in: Component config > LWIP > TCP

Allows enabling ”oversize” allocation of TCP transmission pbufs ahead of time, which can reduce the length of pbuf chains used for transmission.

This will not make a difference to sockets where Nagle’s algorithm is disabled.

Default value of MSS is fine for most applications, 25% MSS may save some RAM when only transmitting small amounts of data. Disabled will have worst performance and fragmentation characteristics, but uses least RAM overall.

Available options:

- MSS (CONFIG_LWIP_TCP_OVERSIZE_MSS)
- 25% MSS (CONFIG_LWIP_TCP_OVERSIZE_QUARTER_MSS)
- Disabled (CONFIG_LWIP_TCP_OVERSIZE_DISABLE)

CONFIG_LWIP_WND_SCALE

Support TCP window scale

Found in: Component config > LWIP > TCP

Enable this feature to support TCP window scaling.

Default value:

- No (disabled) if *CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP*

CONFIG_LWIP_TCP_RCV_SCALE

Set TCP receiving window scaling factor

Found in: Component config > LWIP > TCP > CONFIG_LWIP_WND_SCALE

Enable this feature to support TCP window scaling.

Range:
- from 0 to 14 if *CONFIG_LWIP_WND_SCALE*

Default value:
- 0 if *CONFIG_LWIP_WND_SCALE*

CONFIG_LWIP_TCP_RTO_TIME

Default TCP rto time

Found in: Component config > LWIP > TCP

Set default TCP rto time for a reasonable initial rto. In bad network environment, recommend set value of rto time to 1500.

Default value:
- 1500

UDP

Contains:

- *CONFIG_LWIP_UDP_RECVMBOX_SIZE*
- *CONFIG_LWIP_MAX_UDP_PCBS*

CONFIG_LWIP_MAX_UDP_PCBS

Maximum active UDP control blocks

Found in: Component config > LWIP > UDP

The maximum number of active UDP "connections" (ie UDP sockets sending/receiving data). The practical maximum limit is determined by available heap memory at runtime.

Range:
- from 1 to 1024

Default value:
- 16

CONFIG_LWIP_UDP_RECVMBOX_SIZE

Default UDP receive mail box size

Found in: Component config > LWIP > UDP

Set UDP receive mail box size. The recommended value is 6.

UDP receive mail box is a per socket mail box, when the application receives packets from UDP socket, LWIP core firstly posts the packets to UDP receive mail box and the application then fetches the packets from mail box. It means LWIP can caches maximum UDP_RECCVMBOX_SIZE packets for each UDP socket, so the maximum possible cached UDP packets for all UDP sockets is UDP_RECCVMBOX_SIZE multiples the maximum UDP socket number. In other words, the bigger UDP_RECVMBOX_SIZE means more memory. On the other hand, if the receive mail box is too small, the mail box may be full. If the mail box is full, the LWIP drops the packets. So generally we need to make sure the UDP receive mail box is big enough to avoid packet drop between LWIP core and application.

Range:
- from 6 to 64

Default value:

- 6

Checksums Contains:

- *CONFIG_LWIP_CHECKSUM_CHECK_ICMP*
- *CONFIG_LWIP_CHECKSUM_CHECK_IP*
- *CONFIG_LWIP_CHECKSUM_CHECK_UDP*

CONFIG_LWIP_CHECKSUM_CHECK_IP

Enable LWIP IP checksums

Found in: Component config > LWIP > Checksums

Enable checksum checking for received IP messages

Default value:

- No (disabled)

CONFIG_LWIP_CHECKSUM_CHECK_UDP

Enable LWIP UDP checksums

Found in: Component config > LWIP > Checksums

Enable checksum checking for received UDP messages

Default value:

- No (disabled)

CONFIG_LWIP_CHECKSUM_CHECK_ICMP

Enable LWIP ICMP checksums

Found in: Component config > LWIP > Checksums

Enable checksum checking for received ICMP messages

Default value:

- Yes (enabled)

CONFIG_LWIP_TCPIP_TASK_STACK_SIZE

TCP/IP Task Stack Size

Found in: Component config > LWIP

Configure TCP/IP task stack size, used by LWIP to process multi-threaded TCP/IP operations. Setting this stack too small will result in stack overflow crashes.

Range:

- from 2048 to 65536

Default value:

- 3072

CONFIG_LWIP_TCPIP_TASK_AFFINITY

TCP/IP task affinity

Found in: Component config > LWIP

Allows setting LwIP tasks affinity, i.e. whether the task is pinned to CPU0, pinned to CPU1, or allowed to run on any CPU. Currently this applies to ”TCP/IP” task and ”Ping” task.

Available options:

- No affinity (CONFIG_LWIP_TCPIP_TASK_AFFINITY_NO_AFFINITY)
- CPU0 (CONFIG_LWIP_TCPIP_TASK_AFFINITY_CPU0)
- CPU1 (CONFIG_LWIP_TCPIP_TASK_AFFINITY_CPU1)

CONFIG_LWIP_IPV6_ND6_NUM_PREFIXES

Max number of entries in IPv6 on-link prefixes cache

Found in: Component config > LWIP

Maximum number of entries in IPv6 on-link prefixes cache

Default value:

- 5

CONFIG_LWIP_IPV6_ND6_NUM_ROUTERS

Max number of entries in IPv6 default routers cache

Found in: Component config > LWIP

Maximum number of entries in IPv6 default routers cache

Default value:

- 3

CONFIG_LWIP_IPV6_ND6_NUM_DESTINATIONS

Max number of entries in IPv6 destinations cache

Found in: Component config > LWIP

Maximum number of entries in IPv6 destinations cache

Default value:

- 10

CONFIG_LWIP_PPP_SUPPORT

Enable PPP support

Found in: Component config > LWIP

Enable PPP stack. Now only PPP over serial is possible.

Default value:

- No (disabled)

Contains:

- *CONFIG_LWIP_PPP_ENABLE_IPV6*

CONFIG_LWIP_PPP_ENABLE_IPV6

Enable IPV6 support for PPP connections (IPV6CP)

Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT

Enable IPV6 support in PPP for the local link between the DTE (processor) and DCE (modem). There are some modems which do not support the IPV6 addressing in the local link. If they are requested for IPV6CP negotiation, they may time out. This would in turn fail the configuration for the whole link. If your modem is not responding correctly to PPP Phase Network, try to disable IPV6 support.

Default value:

- Yes (enabled) if *CONFIG_LWIP_PPP_SUPPORT* && *CONFIG_LWIP_IPV6*

CONFIG_LWIP_IPV6_MEMP_NUM_ND6_QUEUE

Max number of IPv6 packets to queue during MAC resolution

Found in: Component config > LWIP

Config max number of IPv6 packets to queue during MAC resolution.

Range:

- from 3 to 20

Default value:

- 3

CONFIG_LWIP_IPV6_ND6_NUM_NEIGHBORS

Max number of entries in IPv6 neighbor cache

Found in: Component config > LWIP

Config max number of entries in IPv6 neighbor cache

Range:

- from 3 to 10

Default value:

- 5

CONFIG_LWIP_PPP_NOTIFY_PHASE_SUPPORT

Enable Notify Phase Callback

Found in: Component config > LWIP

Enable to set a callback which is called on change of the internal PPP state machine.

Default value:

- No (disabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_PPP_PAP_SUPPORT

Enable PAP support

Found in: Component config > LWIP

Enable Password Authentication Protocol (PAP) support

Default value:

- No (disabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_PPP_CHAP_SUPPORT

Enable CHAP support

Found in: Component config > LWIP

Enable Challenge Handshake Authentication Protocol (CHAP) support

Default value:

- No (disabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_PPP_MSCHAP_SUPPORT

Enable MSCHAP support

Found in: Component config > LWIP

Enable Microsoft version of the Challenge-Handshake Authentication Protocol (MSCHAP) support

Default value:

- No (disabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_PPP_MPPE_SUPPORT

Enable MPPE support

Found in: Component config > LWIP

Enable Microsoft Point-to-Point Encryption (MPPE) support

Default value:

- No (disabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_PPP_SERVER_SUPPORT

Enable PPP server support

Found in: Component config > LWIP

Enable to use PPP server

Default value:

- No (disabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_PPP_VJ_HEADER_COMPRESSION

Enable VJ IP Header compression

Found in: Component config > LWIP

Enable support for VJ header compression. Please disable this if you're using NAPT on PPP interface, since the compressed IP header might not be correctly interpreted in NAT causing the compressed packet to be dropped.

Default value:

- Yes (enabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_ENABLE_LCP_ECHO

Enable LCP ECHO

Found in: Component config > LWIP

Enable LCP echo keepalive requests

Default value:

- No (disabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_LCP_ECHOINTERVAL

Echo interval (s)

Found in: Component config > LWIP > CONFIG_LWIP_ENABLE_LCP_ECHO

Interval in seconds between keepalive LCP echo requests, 0 to disable.

Range:

- from 0 to 1000000 if *CONFIG_LWIP_ENABLE_LCP_ECHO*

Default value:

- 3 if *CONFIG_LWIP_ENABLE_LCP_ECHO*

CONFIG_LWIP_LCP_MAXECHOFAILS

Maximum echo failures

Found in: Component config > LWIP > CONFIG_LWIP_ENABLE_LCP_ECHO

Number of consecutive unanswered echo requests before failure is indicated.

Range:

- from 0 to 100000 if *CONFIG_LWIP_ENABLE_LCP_ECHO*

Default value:

- 3 if *CONFIG_LWIP_ENABLE_LCP_ECHO*

CONFIG_LWIP_PPP_DEBUG_ON

Enable PPP debug log output

Found in: Component config > LWIP

Enable PPP debug log output

Default value:

- No (disabled) if *CONFIG_LWIP_PPP_SUPPORT*

CONFIG_LWIP_USE_EXTERNAL_MBEDTLS

Use mbedTLS instead of internal polarSSL

Found in: Component config > LWIP

This option uses mbedTLS crypto functions (instead of internal PolarSSL implementation) for PPP authentication modes (PAP, CHAP, etc.). You can use this option to address symbol duplication issues, since the internal functions are not namespaced (e.g. md5_init()).

CONFIG_LWIP_SLIP_SUPPORT

Enable SLIP support (new/experimental)

Found in: Component config > LWIP

Enable SLIP stack. Now only SLIP over serial is possible.

SLIP over serial support is experimental and unsupported.

Default value:

- No (disabled)

Contains:

- *CONFIG_LWIP_SLIP_DEBUG_ON*

CONFIG_LWIP_SLIP_DEBUG_ON

Enable SLIP debug log output

Found in: Component config > LWIP > CONFIG_LWIP_SLIP_SUPPORT

Enable SLIP debug log output

Default value:

- No (disabled) if *CONFIG_LWIP_SLIP_SUPPORT*

ICMP Contains:

- *CONFIG_LWIP_ICMP*
- *CONFIG_LWIP_BROADCAST_PING*
- *CONFIG_LWIP_MULTICAST_PING*

CONFIG_LWIP_ICMP

ICMP: Enable ICMP

Found in: Component config > LWIP > ICMP

Enable ICMP module for check network stability

Default value:

- Yes (enabled)

CONFIG_LWIP_MULTICAST_PING

Respond to multicast pings

Found in: Component config > LWIP > ICMP

Default value:

- No (disabled)

CONFIG_LWIP_BROADCAST_PING

Respond to broadcast pings

Found in: Component config > LWIP > ICMP

Default value:

- No (disabled)

LWIP RAW API Contains:

- *CONFIG_LWIP_MAX_RAW_PCBS*

CONFIG_LWIP_MAX_RAW_PCBS

Maximum LWIP RAW PCBs

Found in: Component config > LWIP > LWIP RAW API

The maximum number of simultaneously active LWIP RAW protocol control blocks. The practical maximum limit is determined by available heap memory at runtime.

Range:

- from 1 to 1024

Default value:

- 16

SNTP Contains:

- *CONFIG_LWIP_SNTP_STARTUP_DELAY*
- *CONFIG_LWIP_SNTP_MAX_SERVERS*
- *CONFIG_LWIP_SNTP_UPDATE_DELAY*
- *CONFIG_LWIP_DHCP_GET_NTP_SRV*

CONFIG_LWIP_SNTP_MAX_SERVERS

Maximum number of NTP servers

Found in: Component config > LWIP > SNTP

Set maximum number of NTP servers used by LwIP SNTP module. First argument of sntp_setserver/sntp_setservername functions is limited to this value.

Range:
- from 1 to 16

Default value:
- 1

CONFIG_LWIP_DHCP_GET_NTP_SRV

Request NTP servers from DHCP

Found in: Component config > LWIP > SNTP

If enabled, LWIP will add 'NTP' to Parameter-Request Option sent via DHCP-request. DHCP server might reply with an NTP server address in option 42. SNTP callback for such replies should be set accordingly (see sntp_servermode_dhcp() func.)

Default value:
- No (disabled)

CONFIG_LWIP_DHCP_MAX_NTP_SERVERS

Maximum number of NTP servers acquired via DHCP

Found in: Component config > LWIP > SNTP > CONFIG_LWIP_DHCP_GET_NTP_SRV

Set maximum number of NTP servers acquired via DHCP-offer. Should be less or equal to "Maximum number of NTP servers", any extra servers would be just ignored.

Range:
- from 1 to 16 if *CONFIG_LWIP_DHCP_GET_NTP_SRV*

Default value:
- 1 if *CONFIG_LWIP_DHCP_GET_NTP_SRV*

CONFIG_LWIP_SNTP_UPDATE_DELAY

Request interval to update time (ms)

Found in: Component config > LWIP > SNTP

This option allows you to set the time update period via SNTP. Default is 1 hour. Must not be below 15 seconds by specification. (SNTPv4 RFC 4330 enforces a minimum update time of 15 seconds).

Range:
- from 15000 to 4294967295

Default value:
- 3600000

CONFIG_LWIP_SNTP_STARTUP_DELAY

Enable SNTP startup delay

Found in: Component config > LWIP > SNTP

It is recommended (RFC 4330) to delay the initial request after by a random timeout from 1 to 5 minutes to reduce potential load of NTP servers after simultaneous power-up of many devices. This option disables this initial delay. Please use this option with care, it could improve a single device responsiveness

but might cause peaks on the network after reset. Another option to address responsiveness of devices while using the initial random delay is to adjust LWIP_SNTP_MAXIMUM_STARTUP_DELAY.

Default value:

- Yes (enabled)

CONFIG_LWIP_SNTP_MAXIMUM_STARTUP_DELAY

Maximum startup delay (ms)

Found in: Component config > LWIP > SNTP > CONFIG_LWIP_SNTP_STARTUP_DELAY

RFC 4330 recommends a startup delay before sending the initial request. LWIP calculates this delay to a random number of milliseconds between 0 and this value.

Range:

- from 100 to 300000

Default value:

- 5000

DNS Contains:

- *CONFIG_LWIP_FALLBACK_DNS_SERVER_SUPPORT*
- *CONFIG_LWIP_DNS_SETSERVER_WITH_NETIF*
- *CONFIG_LWIP_DNS_MAX_SERVERS*
- *CONFIG_LWIP_DNS_MAX_HOST_IP*

CONFIG_LWIP_DNS_MAX_HOST_IP

Maximum number of IP addresses per host

Found in: Component config > LWIP > DNS

Maximum number of IP addresses that can be returned by DNS queries for a single host.

Default value:

- 1

CONFIG_LWIP_DNS_MAX_SERVERS

Maximum number of DNS servers

Found in: Component config > LWIP > DNS

Set maximum number of DNS servers. If fallback DNS servers are supported, the number of DNS servers needs to be greater than or equal to 3.

Range:

- from 1 to 4

Default value:

- 3

CONFIG_LWIP_FALLBACK_DNS_SERVER_SUPPORT

Enable DNS fallback server support

Found in: Component config > LWIP > DNS

Enable this feature to support DNS fallback server.

Default value:

- No (disabled)

CONFIG_LWIP_FALLBACK_DNS_SERVER_ADDRESS

DNS fallback server address

Found in: Component config > LWIP > DNS > CONFIG_LWIP_FALLBACK_DNS_SERVER_SUPPORT

This option allows you to config dns fallback server address.

Default value:

- "114.114.114.114" if *CONFIG_LWIP_FALLBACK_DNS_SERVER_SUPPORT*

CONFIG_LWIP_DNS_SETSERVER_WITH_NETIF

Enable DNS server settings with netif

Found in: Component config > LWIP > DNS

This option allows collecting DNS server settings per netif using configurable callback function. It's typically used with CONFIG_ESP_NETIF_SET_DNS_PER_DEFAULT_NETIF which configures a callback to collect the DNS info on esp_netif layer.

Default value:

- No (disabled)

CONFIG_LWIP_BRIDGEIF_MAX_PORTS

Maximum number of bridge ports

Found in: Component config > LWIP

Set maximum number of ports a bridge can consists of.

Range:

- from 1 to 63

Default value:

- 7

CONFIG_LWIP_ESP_LWIP_ASSERT

Enable LWIP ASSERT checks

Found in: Component config > LWIP

Enable this option keeps LWIP assertion checks enabled. It is recommended to keep this option enabled.

If asserts are disabled for the entire project, they are also disabled for LWIP and this option is ignored.

Hooks Contains:

- *CONFIG_LWIP_HOOK_DNS_EXTERNAL_RESOLVE*
- *CONFIG_LWIP_HOOK_ND6_GET_GW*
- *CONFIG_LWIP_HOOK_IP6_INPUT*
- *CONFIG_LWIP_HOOK_IP6_ROUTE*
- *CONFIG_LWIP_HOOK_IP6_SELECT_SRC_ADDR*
- *CONFIG_LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE*
- *CONFIG_LWIP_HOOK_TCP_ISN*

CONFIG_LWIP_HOOK_TCP_ISN

TCP ISN Hook

Found in: Component config > LWIP > Hooks

Enables to define a TCP ISN hook to randomize initial sequence number in TCP connection. The default TCP ISN algorithm used in IDF (standardized in RFC 6528) produces ISN by combining an MD5 of

the new TCP id and a stable secret with the current time. This is because the lwIP implementation (*tcp_next_iss*) is not very strong, as it does not take into consideration any platform specific entropy source.

Set to LWIP_HOOK_TCP_ISN_CUSTOM to provide custom implementation. Set to LWIP_HOOK_TCP_ISN_NONE to use lwIP implementation.

Available options:

- No hook declared (CONFIG_LWIP_HOOK_TCP_ISN_NONE)
- Default implementation (CONFIG_LWIP_HOOK_TCP_ISN_DEFAULT)
- Custom implementation (CONFIG_LWIP_HOOK_TCP_ISN_CUSTOM)

CONFIG_LWIP_HOOK_IP6_ROUTE

IPv6 route Hook

Found in: Component config > LWIP > Hooks

Enables custom IPv6 route hook. Setting this to ”default” provides weak implementation stub that could be overwritten in application code. Setting this to ”custom” provides hook’s declaration only and expects the application to implement it.

Available options:

- No hook declared (CONFIG_LWIP_HOOK_IP6_ROUTE_NONE)
- Default (weak) implementation (CONFIG_LWIP_HOOK_IP6_ROUTE_DEFAULT)
- Custom implementation (CONFIG_LWIP_HOOK_IP6_ROUTE_CUSTOM)

CONFIG_LWIP_HOOK_ND6_GET_GW

IPv6 get gateway Hook

Found in: Component config > LWIP > Hooks

Enables custom IPv6 route hook. Setting this to ”default” provides weak implementation stub that could be overwritten in application code. Setting this to ”custom” provides hook’s declaration only and expects the application to implement it.

Available options:

- No hook declared (CONFIG_LWIP_HOOK_ND6_GET_GW_NONE)
- Default (weak) implementation (CONFIG_LWIP_HOOK_ND6_GET_GW_DEFAULT)
- Custom implementation (CONFIG_LWIP_HOOK_ND6_GET_GW_CUSTOM)

CONFIG_LWIP_HOOK_IP6_SELECT_SRC_ADDR

IPv6 source address selection Hook

Found in: Component config > LWIP > Hooks

Enables custom IPv6 source address selection. Setting this to ”default” provides weak implementation stub that could be overwritten in application code. Setting this to ”custom” provides hook’s declaration only and expects the application to implement it.

Available options:

- No hook declared (CONFIG_LWIP_HOOK_IP6_SELECT_SRC_ADDR_NONE)
- Default (weak) implementation (CONFIG_LWIP_HOOK_IP6_SELECT_SRC_ADDR_DEFAULT)

- Custom implementation (CONFIG_LWIP_HOOK_IP6_SELECT_SRC_ADDR_CUSTOM)

CONFIG_LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE

Netconn external resolve Hook

Found in: Component config > LWIP > Hooks

Enables custom DNS resolve hook (without callback). Setting this to ”default” provides weak implementation stub that could be overwritten in application code. Setting this to ”custom” provides hook’s declaration only and expects the application to implement it.

Available options:

- No hook declared (CONFIG_LWIP_HOOK_NETCONN_EXT_RESOLVE_NONE)
- Default (weak) implementation (CONFIG_LWIP_HOOK_NETCONN_EXT_RESOLVE_DEFAULT)
- Custom implementation (CONFIG_LWIP_HOOK_NETCONN_EXT_RESOLVE_CUSTOM)

CONFIG_LWIP_HOOK_DNS_EXTERNAL_RESOLVE

DNS external resolve Hook

Found in: Component config > LWIP > Hooks

Enables custom DNS resolve hook (with callback). Setting this to ”custom” provides hook’s declaration only and expects the application to implement it.

Available options:

- No hook declared (CONFIG_LWIP_HOOK_DNS_EXT_RESOLVE_NONE)
- Custom implementation (CONFIG_LWIP_HOOK_DNS_EXT_RESOLVE_CUSTOM)

CONFIG_LWIP_HOOK_IP6_INPUT

IPv6 packet input

Found in: Component config > LWIP > Hooks

Enables custom IPv6 packet input. Setting this to ”default” provides weak IDF implementation, which drops all incoming IPv6 traffic if the interface has no link local address. (this default implementation is ”weak” and could be still overwritten in the application if some additional IPv6 input packet filtering is needed) Setting this to ”none” removes this default filter and conforms to the lwIP implementation (which accepts multicasts even if the interface has no link local address) Setting this to ”custom” provides hook’s declaration only and expects the application to implement it.

Available options:

- No hook declared (CONFIG_LWIP_HOOK_IP6_INPUT_NONE)
- Default (weak) implementation (CONFIG_LWIP_HOOK_IP6_INPUT_DEFAULT)
- Custom implementation (CONFIG_LWIP_HOOK_IP6_INPUT_CUSTOM)

CONFIG_LWIP_DEBUG

Enable LWIP Debug

Found in: Component config > LWIP

Enabling this option allows different kinds of lwIP debug output.

All lwIP debug features increase the size of the final binary.

Default value:

- No (disabled)

Contains:

- *CONFIG_LWIP_API_LIB_DEBUG*
- *CONFIG_LWIP_BRIDGEIF_FDB_DEBUG*
- *CONFIG_LWIP_BRIDGEIF_FW_DEBUG*
- *CONFIG_LWIP_BRIDGEIF_DEBUG*
- *CONFIG_LWIP_DHCP_DEBUG*
- *CONFIG_LWIP_DHCP_STATE_DEBUG*
- *CONFIG_LWIP_DNS_DEBUG*
- *CONFIG_LWIP_ETHARP_DEBUG*
- *CONFIG_LWIP_ICMP_DEBUG*
- *CONFIG_LWIP_ICMP6_DEBUG*
- *CONFIG_LWIP_IP_DEBUG*
- *CONFIG_LWIP_IP6_DEBUG*
- *CONFIG_LWIP_NAPT_DEBUG*
- *CONFIG_LWIP_NETIF_DEBUG*
- *CONFIG_LWIP_PBUF_DEBUG*
- *CONFIG_LWIP_SNTP_DEBUG*
- *CONFIG_LWIP_SOCKETS_DEBUG*
- *CONFIG_LWIP_TCP_DEBUG*
- *CONFIG_LWIP_UDP_DEBUG*
- *CONFIG_LWIP_DEBUG_ESP_LOG*

CONFIG_LWIP_DEBUG_ESP_LOG

Route LWIP debugs through ESP_LOG interface

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Enabling this option routes all enabled LWIP debugs through ESP_LOGD.

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_NETIF_DEBUG

Enable netif debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_PBUF_DEBUG

Enable pbuf debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_ETHARP_DEBUG

Enable etharp debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_API_LIB_DEBUG

Enable api lib debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_SOCKETS_DEBUG

Enable socket debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_IP_DEBUG

Enable IP debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_ICMP_DEBUG

Enable ICMP debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG* && *CONFIG_LWIP_ICMP*

CONFIG_LWIP_DHCP_STATE_DEBUG

Enable DHCP state tracking

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_DHCP_DEBUG

Enable DHCP debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_IP6_DEBUG

Enable IP6 debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_ICMP6_DEBUG

Enable ICMP6 debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_TCP_DEBUG

Enable TCP debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_UDP_DEBUG

Enable UDP debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_SNTP_DEBUG

Enable SNTP debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_DNS_DEBUG

Enable DNS debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_NAPT_DEBUG

Enable NAPT debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG* && *CONFIG_LWIP_IPV4_NAPT*

CONFIG_LWIP_BRIDGEIF_DEBUG

Enable bridge generic debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_BRIDGEIF_FDB_DEBUG

Enable bridge FDB debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

CONFIG_LWIP_BRIDGEIF_FW_DEBUG

Enable bridge forwarding debug messages

Found in: Component config > LWIP > CONFIG_LWIP_DEBUG

Default value:

- No (disabled) if *CONFIG_LWIP_DEBUG*

mbedTLS Contains:

- *CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN*
- *Certificate Bundle*
- *Certificates*
- *CONFIG_MBEDTLS_CHACHA20_C*
- *CONFIG_MBEDTLS_DHM_C*
- *CONFIG_MBEDTLS_ECP_C*
- *CONFIG_MBEDTLS_ECDH_C*
- *CONFIG_MBEDTLS_ECJPAKE_C*
- *CONFIG_MBEDTLS_ECP_DP_BP256R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_BP384R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_BP512R1_ENABLED*
- *CONFIG_MBEDTLS_CMAC_C*
- *CONFIG_MBEDTLS_ECP_DP_CURVE25519_ENABLED*
- *CONFIG_MBEDTLS_ECDSA_DETERMINISTIC*
- *CONFIG_MBEDTLS_HARDWARE_ECDSA_VERIFY*
- *CONFIG_MBEDTLS_HARDWARE_ECDSA_SIGN*
- *CONFIG_MBEDTLS_ERROR_STRINGS*
- *CONFIG_MBEDTLS_ECP_FIXED_POINT_OPTIM*
- *CONFIG_MBEDTLS_FS_IO*
- *CONFIG_MBEDTLS_HARDWARE_AES*
- *CONFIG_MBEDTLS_HARDWARE_ECC*
- *CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN*
- *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*
- *CONFIG_MBEDTLS_HARDWARE_MPI*
- *CONFIG_MBEDTLS_HARDWARE_SHA*
- *CONFIG_MBEDTLS_DEBUG*
- *CONFIG_MBEDTLS_ECP_RESTARTABLE*
- *CONFIG_MBEDTLS_HAVE_TIME*
- *CONFIG_MBEDTLS_RIPEMD160_C*
- *CONFIG_MBEDTLS_ECP_DP_SECP192K1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP192R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP224K1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP224R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP256K1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP256R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP384R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP521R1_ENABLED*
- *CONFIG_MBEDTLS_SHA512_C*
- *CONFIG_MBEDTLS_SHA3_C*
- *CONFIG_MBEDTLS_THREADING_C*

- *CONFIG_MBEDTLS_HKDF_C*
- *mbedTLS v3.x related*
- *CONFIG_MBEDTLS_MEM_ALLOC_MODE*
- *CONFIG_MBEDTLS_ECP_NIST_OPTIM*
- *CONFIG_MBEDTLS_POLY1305_C*
- *CONFIG_MBEDTLS_SSL_ALPN*
- *CONFIG_MBEDTLS_SSL_PROTO_DTLS*
- *CONFIG_MBEDTLS_SSL_PROTO_GMTSSL1_1*
- *CONFIG_MBEDTLS_SSL_PROTO_TLS1_2*
- *CONFIG_MBEDTLS_SSL_RENEGOTIATION*
- *Symmetric Ciphers*
- *TLS Key Exchange Methods*
- *CONFIG_MBEDTLS_SSL_MAX_CONTENT_LEN*
- *CONFIG_MBEDTLS_TLS_MODE*
- *CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS*
- *CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS*
- *CONFIG_MBEDTLS_ROM_MD5*
- *CONFIG_MBEDTLS_USE_CRYPTO_ROM_IMPL*
- *CONFIG_MBEDTLS_DYNAMIC_BUFFER*

CONFIG_MBEDTLS_MEM_ALLOC_MODE

Memory allocation strategy

Found in: Component config > mbedTLS

Allocation strategy for mbedTLS, essentially provides ability to allocate all required dynamic allocations from,

- Internal DRAM memory only
- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF
- Custom allocation mode, by overwriting calloc()/free() using mbedtls_platform_set_calloc_free() function
- Internal IRAM memory wherever applicable else internal DRAM

Recommended mode here is always internal (*), since that is most preferred from security perspective. But if application requirement does not allow sufficient free internal memory then alternate mode can be selected.

(*) In case of ESP32-S2/ESP32-S3, hardware allows encryption of external SPIRAM contents provided hardware flash encryption feature is enabled. In that case, using external SPIRAM allocation strategy is also safe choice from security perspective.

Available options:

- Internal memory (CONFIG_MBEDTLS_INTERNAL_MEM_ALLOC)
- External SPIRAM (CONFIG_MBEDTLS_EXTERNAL_MEM_ALLOC)
- Default alloc mode (CONFIG_MBEDTLS_DEFAULT_MEM_ALLOC)
- Custom alloc mode (CONFIG_MBEDTLS_CUSTOM_MEM_ALLOC)
- Internal IRAM (CONFIG_MBEDTLS_IRAM_8BIT_MEM_ALLOC)
  Allows to use IRAM memory region as 8bit accessible region.
  TLS input and output buffers will be allocated in IRAM section which is 32bit aligned memory. Every unaligned (8bit or 16bit) access will result in an exception and incur penalty of certain clock cycles per unaligned read/write.

CONFIG_MBEDTLS_SSL_MAX_CONTENT_LEN

TLS maximum message content length

Found in: Component config > mbedTLS

Maximum TLS message length (in bytes) supported by mbedTLS.

16384 is the default and this value is required to comply fully with TLS standards.

However you can set a lower value in order to save RAM. This is safe if the other end of the connection supports Maximum Fragment Length Negotiation Extension (max_fragment_length, see RFC6066) or you know for certain that it will never send a message longer than a certain number of bytes.

If the value is set too low, symptoms are a failed TLS handshake or a return value of MBEDTLS_ERR_SSL_INVALID_RECORD (-0x7200).

CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN

Asymmetric in/out fragment length

Found in: Component config > mbedTLS

If enabled, this option allows customizing TLS in/out fragment length in asymmetric way. Please note that enabling this with default values saves 12KB of dynamic memory per TLS connection.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_SSL_IN_CONTENT_LEN

TLS maximum incoming fragment length

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN

This defines maximum incoming fragment length, overriding default maximum content length (MBEDTLS_SSL_MAX_CONTENT_LEN).

Range:

- from 512 to 16384

Default value:

- 16384

CONFIG_MBEDTLS_SSL_OUT_CONTENT_LEN

TLS maximum outgoing fragment length

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN

This defines maximum outgoing fragment length, overriding default maximum content length (MBEDTLS_SSL_MAX_CONTENT_LEN).

Range:

- from 512 to 16384

Default value:

- 4096

CONFIG_MBEDTLS_DYNAMIC_BUFFER

Using dynamic TX/RX buffer

Found in: Component config > mbedTLS

Using dynamic TX/RX buffer. After enabling this option, mbedTLS will allocate TX buffer when need to send data and then free it if all data is sent, allocate RX buffer when need to receive data and then free it when all data is used or read by upper layer.

By default, when SSL is initialized, mbedTLS also allocate TX and RX buffer with the default value of ”MBEDTLS_SSL_OUT_CONTENT_LEN” or ”MBEDTLS_SSL_IN_CONTENT_LEN”, so to save more heap, users can set the options to be an appropriate value.

CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA

Free private key and DHM data after its usage

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_DYNAMIC_BUFFER

Free private key and DHM data after its usage in handshake process.

The option will decrease heap cost when handshake, but also lead to problem:

Because all certificate, private key and DHM data are freed so users should register certificate and private key to ssl config object again.

Default value:

- No (disabled) if *CONFIG_MBEDTLS_DYNAMIC_BUFFER*

CONFIG_MBEDTLS_DYNAMIC_FREE_CA_CERT

Free SSL CA certificate after its usage

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_DYNAMIC_BUFFER > CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA

Free CA certificate after its usage in the handshake process. This option will decrease the heap footprint for the TLS handshake, but may lead to a problem: If the respective ssl object needs to perform the TLS handshake again, the CA certificate should once again be registered to the ssl object.

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA*

CONFIG_MBEDTLS_DEBUG

Enable mbedTLS debugging

Found in: Component config > mbedTLS

Enable mbedTLS debugging functions at compile time.

If this option is enabled, you can include ”mbedtls/esp_debug.h” and call mbedtls_esp_enable_debug_log() at runtime in order to enable mbedTLS debug output via the ESP log mechanism.

Default value:

- No (disabled)

CONFIG_MBEDTLS_DEBUG_LEVEL

Set mbedTLS debugging level

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_DEBUG

Set mbedTLS debugging level

Available options:

- Warning (CONFIG_MBEDTLS_DEBUG_LEVEL_WARN)
- Info (CONFIG_MBEDTLS_DEBUG_LEVEL_INFO)
- Debug (CONFIG_MBEDTLS_DEBUG_LEVEL_DEBUG)
- Verbose (CONFIG_MBEDTLS_DEBUG_LEVEL_VERBOSE)

mbedTLS v3.x related Contains:

- *DTLS-based configurations*
- *CONFIG_MBEDTLS_PKCS7_C*
- *CONFIG_MBEDTLS_SSL_CONTEXT_SERIALIZATION*
- *CONFIG_MBEDTLS_X509_TRUSTED_CERT_CALLBACK*
- *CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE*
- *CONFIG_MBEDTLS_SSL_CID_PADDING_GRANULARITY*
- *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3*
- *CONFIG_MBEDTLS_ECDH_LEGACY_CONTEXT*
- *CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH*

CONFIG_MBEDTLS_SSL_PROTO_TLS1_3

Support TLS 1.3 protocol

Found in: Component config > mbedTLS > mbedTLS v3.x related

TLS 1.3 related configurations Contains:

- *CONFIG_MBEDTLS_SSL_TLS1_3_KEXM_EPHEMERAL*
- *CONFIG_MBEDTLS_SSL_TLS1_3_COMPATIBILITY_MODE*
- *CONFIG_MBEDTLS_SSL_TLS1_3_KEXM_PSK_EPHEMERAL*
- *CONFIG_MBEDTLS_SSL_TLS1_3_KEXM_PSK*

CONFIG_MBEDTLS_SSL_TLS1_3_COMPATIBILITY_MODE

TLS 1.3 middlebox compatibility mode

Found in: Component config > mbedTLS > mbedTLS v3.x related > CONFIG_MBEDTLS_SSL_PROTO_TLS1_3 > TLS 1.3 related configurations

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3*

CONFIG_MBEDTLS_SSL_TLS1_3_KEXM_PSK

TLS 1.3 PSK key exchange mode

Found in: Component config > mbedTLS > mbedTLS v3.x related > CONFIG_MBEDTLS_SSL_PROTO_TLS1_3 > TLS 1.3 related configurations

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3*

CONFIG_MBEDTLS_SSL_TLS1_3_KEXM_EPHEMERAL

TLS 1.3 ephemeral key exchange mode

Found in: Component config > mbedTLS > mbedTLS v3.x related > CONFIG_MBEDTLS_SSL_PROTO_TLS1_3 > TLS 1.3 related configurations

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3*

CONFIG_MBEDTLS_SSL_TLS1_3_KEXM_PSK_EPHEMERAL

TLS 1.3 PSK ephemeral key exchange mode

Found in: Component config > mbedTLS > mbedTLS v3.x related > CONFIG_MBEDTLS_SSL_PROTO_TLS1_3 > TLS 1.3 related configurations

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3*

CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH

Variable SSL buffer length

Found in: Component config > mbedTLS > mbedTLS v3.x related

This enables the SSL buffer to be resized automatically based on the negotiated maximum fragment length in each direction.

Default value:

- No (disabled)

CONFIG_MBEDTLS_ECDH_LEGACY_CONTEXT

Use a backward compatible ECDH context (Experimental)

Found in: Component config > mbedTLS > mbedTLS v3.x related

Use the legacy ECDH context format. Define this option only if you enable MBEDTLS_ECP_RESTARTABLE or if you want to access ECDH context fields directly.

Default value:

- No (disabled) if *CONFIG_MBEDTLS_ECDH_C* && *CONFIG_MBEDTLS_ECP_RESTARTABLE*

CONFIG_MBEDTLS_X509_TRUSTED_CERT_CALLBACK

Enable trusted certificate callbacks

Found in: Component config > mbedTLS > mbedTLS v3.x related

Enables users to configure the set of trusted certificates through a callback instead of a linked list.

See mbedTLS documentation for required API and more details.

Default value:

- No (disabled)

CONFIG_MBEDTLS_SSL_CONTEXT_SERIALIZATION

Enable serialization of the TLS context structures

Found in: Component config > mbedTLS > mbedTLS v3.x related

Enable serialization of the TLS context structures This is a local optimization in handling a single, potentially long-lived connection.

See mbedTLS documentation for required API and more details. Disabling this option will save some code size.

Default value:

- No (disabled)

CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE

Keep peer certificate after handshake completion

Found in: Component config > mbedTLS > mbedTLS v3.x related

Keep the peer's certificate after completion of the handshake. Disabling this option will save about 4kB of heap and some code size.

See mbedTLS documentation for required API and more details.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_PKCS7_C

Enable PKCS #7

Found in: Component config > mbedTLS > mbedTLS v3.x related

Enable PKCS #7 core for using PKCS #7-formatted signatures.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_SSL_CID_PADDING_GRANULARITY

Record plaintext padding

Found in: Component config > mbedTLS > mbedTLS v3.x related

Controls the use of record plaintext padding in TLS 1.3 and when using the Connection ID extension in DTLS 1.2.

The padding will always be chosen so that the length of the padded plaintext is a multiple of the value of this option.

Notes: A value of 1 means that no padding will be used for outgoing records. On systems lacking division instructions, a power of two should be preferred.

Range:

- from 0 to 32 if *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3* || *CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID*

Default value:

- 16 if *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3* || *CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID*

DTLS-based configurations Contains:

- *CONFIG_MBEDTLS_SSL_DTLS_SRTP*
- *CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID*

CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID

Support for the DTLS Connection ID extension

Found in: Component config > mbedTLS > mbedTLS v3.x related > DTLS-based configurations

Enable support for the DTLS Connection ID extension which allows to identify DTLS connections across changes in the underlying transport.

Default value:

- No (disabled) if *CONFIG_MBEDTLS_SSL_PROTO_DTLS*

CONFIG_MBEDTLS_SSL_CID_IN_LEN_MAX

Maximum length of CIDs used for incoming DTLS messages

Found in: Component config > mbedTLS > mbedTLS v3.x related > DTLS-based configurations > CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID

Maximum length of CIDs used for incoming DTLS messages

Range:

- from 0 to 32 if *CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID* && *CONFIG_MBEDTLS_SSL_PROTO_DTLS*

Default value:

- 32 if *CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID* && *CONFIG_MBEDTLS_SSL_PROTO_DTLS*

CONFIG_MBEDTLS_SSL_CID_OUT_LEN_MAX

Maximum length of CIDs used for outgoing DTLS messages

Found in: Component config > *mbedTLS* > *mbedTLS v3.x related* > *DTLS-based configurations* > *CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID*

Maximum length of CIDs used for outgoing DTLS messages

Range:

- from 0 to 32 if *CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID* && *CONFIG_MBEDTLS_SSL_PROTO_DTLS*

Default value:

- 32 if *CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID* && *CONFIG_MBEDTLS_SSL_PROTO_DTLS*

CONFIG_MBEDTLS_SSL_DTLS_SRTP

Enable support for negotiation of DTLS-SRTP (RFC 5764)

Found in: Component config > *mbedTLS* > *mbedTLS v3.x related* > *DTLS-based configurations*

Enable support for negotiation of DTLS-SRTP (RFC 5764) through the use_srtp extension.

See mbedTLS documentation for required API and more details. Disabling this option will save some code size.

Default value:

- No (disabled) if *CONFIG_MBEDTLS_SSL_PROTO_DTLS*

Certificate Bundle Contains:

- *CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*

CONFIG_MBEDTLS_CERTIFICATE_BUNDLE

Enable trusted root certificate bundle

Found in: Component config > *mbedTLS* > *Certificate Bundle*

Enable support for large number of default root certificates

When enabled this option allows user to store default as well as customer specific root certificates in compressed format rather than storing full certificate. For the root certificates the public key and the subject name will be stored.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_DEFAULT_CERTIFICATE_BUNDLE

Default certificate bundle options

Found in: Component config > *mbedTLS* > *Certificate Bundle* > *CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*

Available options:

- Use the full default certificate bundle (CONFIG_MBEDTLS_CERTIFICATE_BUNDLE_DEFAULT_FULL)

- Use only the most common certificates from the default bundles (CONFIG_MBEDTLS_CERTIFICATE_BUNDLE_DEFAULT_CMN)
  Use only the most common certificates from the default bundles, reducing the size with 50%, while still having around 99% coverage.
- Do not use the default certificate bundle (CONFIG_MBEDTLS_CERTIFICATE_BUNDLE_DEFAULT_NONE)

CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE

Add custom certificates to the default bundle

Found in: Component config > mbedTLS > Certificate Bundle > CONFIG_MBEDTLS_CERTIFICATE_BUNDLE

Default value:

- No (disabled)

CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE_PATH

Custom certificate bundle path

Found in: Component config > mbedTLS > Certificate Bundle > CONFIG_MBEDTLS_CERTIFICATE_BUNDLE > CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE

Name of the custom certificate directory or file. This path is evaluated relative to the project root directory.

CONFIG_MBEDTLS_CERTIFICATE_BUNDLE_DEPRECATED_LIST

Add deprecated root certificates

Found in: Component config > mbedTLS > Certificate Bundle > CONFIG_MBEDTLS_CERTIFICATE_BUNDLE

Include the deprecated list of root certificates in the bundle. This list gets updated when a certificate is removed from the Mozilla's NSS root certificate store. This config can be enabled if you would like to ensure that none of the certificates that were deployed in the product are affected because of the update to bundle. In turn, enabling this config keeps expired, retracted certificates in the bundle and it may pose a security risk.

- Deprecated cert list may grow based based on sync with upstream bundle
- Deprecated certs would be be removed in ESP-IDF (next) major release

CONFIG_MBEDTLS_CERTIFICATE_BUNDLE_MAX_CERTS

Maximum no of certificates allowed in certificate bundle

Found in: Component config > mbedTLS > Certificate Bundle > CONFIG_MBEDTLS_CERTIFICATE_BUNDLE

Default value:

- 200

CONFIG_MBEDTLS_ECP_RESTARTABLE

Enable mbedTLS ecp restartable

Found in: Component config > mbedTLS

Enable "non-blocking" ECC operations that can return early and be resumed.

Default value:

- No (disabled)

CONFIG_MBEDTLS_CMAC_C

Enable CMAC mode for block ciphers

Found in: Component config > mbedTLS

Enable the CMAC (Cipher-based Message Authentication Code) mode for block ciphers.

Default value:

- No (disabled)

CONFIG_MBEDTLS_HARDWARE_AES

Enable hardware AES acceleration

Found in: Component config > mbedTLS

Enable hardware accelerated AES encryption & decryption.

Note that if the ESP32 CPU is running at 240MHz, hardware AES does not offer any speed boost over software AES.

CONFIG_MBEDTLS_AES_USE_INTERRUPT

Use interrupt for long AES operations

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_AES

Use an interrupt to coordinate long AES operations.

This allows other code to run on the CPU while an AES operation is pending. Otherwise the CPU busy-waits.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_AES_INTERRUPT_LEVEL

AES hardware interrupt level

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_AES > CONFIG_MBEDTLS_AES_USE_INTERRUPT

This config helps to set the interrupt priority level for the AES peripheral. Value 0 (default) means that there is no preference regarding the interrupt priority level and any level from 1 to 3 can be selected (based on the availability). Note: Higher value indicates high interrupt priority.

Range:

- from 0 to 3

Default value:

- 0

CONFIG_MBEDTLS_HARDWARE_GCM

Enable partially hardware accelerated GCM

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_AES

Enable partially hardware accelerated GCM. GHASH calculation is still done in software.

If MBEDTLS_HARDWARE_GCM is disabled and MBEDTLS_HARDWARE_AES is enabled then mbedTLS will still use the hardware accelerated AES block operation, but on a single block at a time.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_GCM_SUPPORT_NON_AES_CIPHER

Enable support for non-AES ciphers in GCM operation

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_AES

Enable this config to support fallback to software definitions for a non-AES cipher GCM operation as we support hardware acceleration only for AES cipher. Some of the non-AES ciphers used in a GCM operation are DES, ARIA, CAMELLIA, CHACHA20, BLOWFISH.

If this config is disabled, performing a non-AES cipher GCM operation with the config MBEDTLS_HARDWARE_AES enabled will result in calculation of an AES-GCM operation instead for the given input values and thus could lead to failure in certificate validation which would ultimately lead to a SSL handshake failure.

This config being by-default enabled leads to an increase in binary size footprint of ~2.5KB. In case you are sure that your use case (for example, client and server configurations in case of a TLS handshake) would not involve any GCM operations using a non-AES cipher, you can safely disable this config, leading to reduction in binary size footprint.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_HARDWARE_MPI

Enable hardware MPI (bignum) acceleration

Found in: Component config > mbedTLS

Enable hardware accelerated multiple precision integer operations.

Hardware accelerated multiplication, modulo multiplication, and modular exponentiation for up to SOC_RSA_MAX_BIT_LEN bit results.

These operations are used by RSA.

CONFIG_MBEDTLS_LARGE_KEY_SOFTWARE_MPI

Fallback to software implementation for larger MPI values

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_MPI

Fallback to software implementation for RSA key lengths larger than SOC_RSA_MAX_BIT_LEN. If this is not active then the ESP will be unable to process keys greater than SOC_RSA_MAX_BIT_LEN.

Default value:

- No (disabled)

CONFIG_MBEDTLS_MPI_USE_INTERRUPT

Use interrupt for MPI exp-mod operations

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_MPI

Use an interrupt to coordinate long MPI operations.

This allows other code to run on the CPU while an MPI operation is pending. Otherwise the CPU busy-waits.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_MPI_INTERRUPT_LEVEL

MPI hardware interrupt level

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_MPI > CONFIG_MBEDTLS_MPI_USE_INTERRUPT

This config helps to set the interrupt priority level for the MPI peripheral. Value 0 (default) means that there is no preference regarding the interrupt priority level and any level from 1 to 3 can be selected (based on the availability). Note: Higher value indicates high interrupt priority.

Range:
- from 0 to 3

Default value:
- 0

CONFIG_MBEDTLS_HARDWARE_SHA

Enable hardware SHA acceleration

Found in: Component config > mbedTLS

Enable hardware accelerated SHA1, SHA256, SHA384 & SHA512 in mbedTLS.

Due to a hardware limitation, on the ESP32 hardware acceleration is only guaranteed if SHA digests are calculated one at a time. If more than one SHA digest is calculated at the same time, one will be calculated fully in hardware and the rest will be calculated (at least partially calculated) in software. This happens automatically.

SHA hardware acceleration is faster than software in some situations but slower in others. You should benchmark to find the best setting for you.

CONFIG_MBEDTLS_HARDWARE_ECC

Enable hardware ECC acceleration

Found in: Component config > mbedTLS

Enable hardware accelerated ECC point multiplication and point verification for points on curve SECP192R1 and SECP256R1 in mbedTLS

Default value:
- Yes (enabled)

CONFIG_MBEDTLS_ECC_OTHER_CURVES_SOFT_FALLBACK

Fallback to software implementation for curves not supported in hardware

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_ECC

Fallback to software implementation of ECC point multiplication and point verification for curves not supported in hardware.

Default value:
- Yes (enabled)

CONFIG_MBEDTLS_ROM_MD5

Use MD5 implementation in ROM

Found in: Component config > mbedTLS

Use ROM MD5 in mbedTLS.

Default value:
- Yes (enabled)

CONFIG_MBEDTLS_HARDWARE_ECDSA_SIGN

Enable ECDSA signing using on-chip ECDSA peripheral

Found in: Component config > mbedTLS

Enable hardware accelerated ECDSA peripheral to sign data on curve SECP192R1 and SECP256R1 in mbedTLS.

Note that for signing, the private key has to be burnt in an efuse key block with key purpose set to ECDSA_KEY. If no key is burnt, it will report an error

The key should be burnt in little endian format. espefuse.py utility handles it internally but care needs to be taken while burning using esp_efuse APIs

Enable Software Countermeasure for ECDSA signing using on-chip ECDSA peripheral

CONFIG_MBEDTLS_HARDWARE_ECDSA_VERIFY

Enable ECDSA signature verification using on-chip ECDSA peripheral

Found in: Component config > mbedTLS

Enable hardware accelerated ECDSA peripheral to verify signature on curve SECP192R1 and SECP256R1 in mbedTLS.

CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN

Enable hardware ECDSA sign acceleration when using ATECC608A

Found in: Component config > mbedTLS

This option enables hardware acceleration for ECDSA sign function, only when using ATECC608A cryptoauth chip.

Default value:

- No (disabled)

CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY

Enable hardware ECDSA verify acceleration when using ATECC608A

Found in: Component config > mbedTLS

This option enables hardware acceleration for ECDSA sign function, only when using ATECC608A cryptoauth chip.

Default value:

- No (disabled)

CONFIG_MBEDTLS_HAVE_TIME

Enable mbedtls time support

Found in: Component config > mbedTLS

Enable use of time.h functions (time() and gmtime()) by mbedTLS.

This option doesn't require the system time to be correct, but enables functionality that requires relative timekeeping - for example periodic expiry of TLS session tickets or session cache entries.

Disabling this option will save some firmware size, particularly if the rest of the firmware doesn't call any standard timekeeeping functions.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_PLATFORM_TIME_ALT

Enable mbedtls time support: platform-specific

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HAVE_TIME

Enabling this config will provide users with a function ”mbedtls_platform_set_time()” that allows to set an alternative time function pointer.

Default value:

- No (disabled)

CONFIG_MBEDTLS_HAVE_TIME_DATE

Enable mbedtls certificate expiry check

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HAVE_TIME

Enables X.509 certificate expiry checks in mbedTLS.

If this option is disabled (default) then X.509 certificate ”valid from” and ”valid to” timestamp fields are ignored.

If this option is enabled, these fields are compared with the current system date and time. The time is retrieved using the standard time() and gmtime() functions. If the certificate is not valid for the current system time then verification will fail with code MBEDTLS_X509_BADCERT_FUTURE or MBEDTLS_X509_BADCERT_EXPIRED.

Enabling this option requires adding functionality in the firmware to set the system clock to a valid timestamp before using TLS. The recommended way to do this is via ESP-IDF's SNTP functionality, but any method can be used.

In the case where only a small number of certificates are trusted by the device, please carefully consider the tradeoffs of enabling this option. There may be undesired consequences, for example if all trusted certificates expire while the device is offline and a TLS connection is required to update. Or if an issue with the SNTP server means that the system time is invalid for an extended period after a reset.

Default value:

- No (disabled)

CONFIG_MBEDTLS_ECDSA_DETERMINISTIC

Enable deterministic ECDSA

Found in: Component config > mbedTLS

Standard ECDSA is ”fragile” in the sense that lack of entropy when signing may result in a compromise of the long-term signing key.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_SHA512_C

Enable the SHA-384 and SHA-512 cryptographic hash algorithms

Found in: Component config > mbedTLS

Enable MBEDTLS_SHA512_C adds support for SHA-384 and SHA-512.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_SHA3_C

Enable the SHA3 cryptographic hash algorithm

Found in: Component config > mbedTLS

Enabling MBEDTLS_SHA3_C adds support for SHA3. Enabling this configuration option increases the flash footprint by almost 4KB.

Default value:

- No (disabled)

CONFIG_MBEDTLS_TLS_MODE

TLS Protocol Role

Found in: Component config > mbedTLS

mbedTLS can be compiled with protocol support for the TLS server, TLS client, or both server and client.

Reducing the number of TLS roles supported saves code size.

Available options:

- Server & Client (CONFIG_MBEDTLS_TLS_SERVER_AND_CLIENT)
- Server (CONFIG_MBEDTLS_TLS_SERVER_ONLY)
- Client (CONFIG_MBEDTLS_TLS_CLIENT_ONLY)
- None (CONFIG_MBEDTLS_TLS_DISABLED)

TLS Key Exchange Methods

Contains:

- *CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_RSA*
- *CONFIG_MBEDTLS_KEY_EXCHANGE_ECJPAKE*
- *CONFIG_MBEDTLS_PSK_MODES*
- *CONFIG_MBEDTLS_KEY_EXCHANGE_RSA*
- *CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE*

CONFIG_MBEDTLS_PSK_MODES

Enable pre-shared-key ciphersuites

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to show configuration for different types of pre-shared-key TLS authentatication methods.

Leaving this options disabled will save code size if they are not used.

Default value:

- No (disabled)

CONFIG_MBEDTLS_KEY_EXCHANGE_PSK

Enable PSK based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES

Enable to support symmetric key PSK (pre-shared-key) TLS key exchange modes.

Default value:

- No (disabled) if *CONFIG_MBEDTLS_PSK_MODES*

CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_PSK

Enable DHE-PSK based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES

Enable to support Diffie-Hellman PSK (pre-shared-key) TLS authentication modes.

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_PSK_MODES* && *CONFIG_MBEDTLS_DHM_C*

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_PSK

Enable ECDHE-PSK based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES

Enable to support Elliptic-Curve-Diffie-Hellman PSK (pre-shared-key) TLS authentication modes.

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_PSK_MODES* && *CONFIG_MBEDTLS_ECDH_C*

CONFIG_MBEDTLS_KEY_EXCHANGE_RSA_PSK

Enable RSA-PSK based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES

Enable to support RSA PSK (pre-shared-key) TLS authentication modes.

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_PSK_MODES*

CONFIG_MBEDTLS_KEY_EXCHANGE_RSA

Enable RSA-only based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to support ciphersuites with prefix TLS-RSA-WITH-

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_RSA

Enable DHE-RSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to support ciphersuites with prefix TLS-DHE-RSA-WITH-

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_DHM_C*

CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Support Elliptic Curve based ciphersuites

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to show Elliptic Curve based ciphersuite mode options.

Disabling all Elliptic Curve ciphersuites saves code size and can give slightly faster TLS handshakes, provided the server supports RSA-only ciphersuite modes.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_RSA

Enable ECDHE-RSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_ECDSA

Enable ECDHE-ECDSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDH_ECDSA

Enable ECDH-ECDSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDH_RSA

Enable ECDH-RSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_KEY_EXCHANGE_ECJPAKE

Enable ECJPAKE based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to support ciphersuites with prefix TLS-ECJPAKE-WITH-

Default value:

- No (disabled) if *CONFIG_MBEDTLS_ECJPAKE_C* && *CONFIG_MBEDTLS_ECP_DP_SECP256R1_ENABLED*

CONFIG_MBEDTLS_SSL_RENEGOTIATION

Support TLS renegotiation

Found in: Component config > mbedTLS

The two main uses of renegotiation are (1) refresh keys on long-lived connections and (2) client authentication after the initial handshake. If you don't need renegotiation, disabling it will save code size and reduce the possibility of abuse/vulnerability.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_SSL_PROTO_TLS1_2

Support TLS 1.2 protocol

Found in: Component config > mbedTLS

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_SSL_PROTO_GMTSSL1_1

Support GM/T SSL 1.1 protocol

Found in: Component config > mbedTLS

Provisions for GM/T SSL 1.1 support

Default value:

- No (disabled)

CONFIG_MBEDTLS_SSL_PROTO_DTLS

Support DTLS protocol (all versions)

Found in: Component config > mbedTLS

Requires TLS 1.2 to be enabled for DTLS 1.2

Default value:

- No (disabled)

CONFIG_MBEDTLS_SSL_ALPN

Support ALPN (Application Layer Protocol Negotiation)

Found in: Component config > mbedTLS

Disabling this option will save some code size if it is not needed.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS

TLS: Client Support for RFC 5077 SSL session tickets

Found in: Component config > mbedTLS

Client support for RFC 5077 session tickets. See mbedTLS documentation for more details. Disabling this option will save some code size.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS

TLS: Server Support for RFC 5077 SSL session tickets

Found in: Component config > mbedTLS

Server support for RFC 5077 session tickets. See mbedTLS documentation for more details. Disabling this option will save some code size.

Default value:

- Yes (enabled)

Symmetric Ciphers Contains:

- *CONFIG_MBEDTLS_AES_C*
- *CONFIG_MBEDTLS_BLOWFISH_C*
- *CONFIG_MBEDTLS_CAMELLIA_C*
- *CONFIG_MBEDTLS_CCM_C*
- *CONFIG_MBEDTLS_DES_C*
- *CONFIG_MBEDTLS_GCM_C*
- *CONFIG_MBEDTLS_NIST_KW_C*
- *CONFIG_MBEDTLS_XTEA_C*

CONFIG_MBEDTLS_AES_C

AES block cipher

Found in: Component config > mbedTLS > Symmetric Ciphers

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_CAMELLIA_C

Camellia block cipher

Found in: Component config > mbedTLS > Symmetric Ciphers

Default value:

- No (disabled)

CONFIG_MBEDTLS_DES_C

DES block cipher (legacy, insecure)

Found in: Component config > mbedTLS > Symmetric Ciphers

Enables the DES block cipher to support 3DES-based TLS ciphersuites.

3DES is vulnerable to the Sweet32 attack and should only be enabled if absolutely necessary.

Default value:

- No (disabled)

CONFIG_MBEDTLS_BLOWFISH_C

Blowfish block cipher (read help)

Found in: Component config > mbedTLS > Symmetric Ciphers

Enables the Blowfish block cipher (not used for TLS sessions.)

The Blowfish cipher is not used for mbedTLS TLS sessions but can be used for other purposes. Read up on the limitations of Blowfish (including Sweet32) before enabling.

Default value:

- No (disabled)

CONFIG_MBEDTLS_XTEA_C

XTEA block cipher

Found in: Component config > mbedTLS > Symmetric Ciphers

Enables the XTEA block cipher.

Default value:

- No (disabled)

CONFIG_MBEDTLS_CCM_C

CCM (Counter with CBC-MAC) block cipher modes

Found in: Component config > mbedTLS > Symmetric Ciphers

Enable Counter with CBC-MAC (CCM) modes for AES and/or Camellia ciphers.

Disabling this option saves some code size.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_GCM_C

GCM (Galois/Counter) block cipher modes

Found in: Component config > mbedTLS > Symmetric Ciphers

Enable Galois/Counter Mode for AES and/or Camellia ciphers.

This option is generally faster than CCM.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_NIST_KW_C

NIST key wrapping (KW) and KW padding (KWP)

Found in: Component config > mbedTLS > Symmetric Ciphers

Enable NIST key wrapping and key wrapping padding.

Default value:

- No (disabled)

CONFIG_MBEDTLS_RIPEMD160_C

Enable RIPEMD-160 hash algorithm

Found in: Component config > mbedTLS

Enable the RIPEMD-160 hash algorithm.

Default value:

- No (disabled)

Certificates Contains:

- *CONFIG_MBEDTLS_PEM_PARSE_C*
- *CONFIG_MBEDTLS_PEM_WRITE_C*
- *CONFIG_MBEDTLS_X509_CRL_PARSE_C*
- *CONFIG_MBEDTLS_X509_CSR_PARSE_C*

CONFIG_MBEDTLS_PEM_PARSE_C

Read & Parse PEM formatted certificates

Found in: Component config > mbedTLS > Certificates

Enable decoding/parsing of PEM formatted certificates.

If your certificates are all in the simpler DER format, disabling this option will save some code size.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_PEM_WRITE_C

Write PEM formatted certificates

Found in: Component config > mbedTLS > Certificates

Enable writing of PEM formatted certificates.

If writing certificate data only in DER format, disabling this option will save some code size.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_X509_CRL_PARSE_C

X.509 CRL parsing

Found in: Component config > mbedTLS > Certificates

Support for parsing X.509 Certificate Revocation Lists.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_X509_CSR_PARSE_C

X.509 CSR parsing

Found in: Component config > mbedTLS > Certificates

Support for parsing X.509 Certificate Signing Requests

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_ECP_C

Elliptic Curve Ciphers

Found in: Component config > mbedTLS

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_DHM_C

Diffie-Hellman-Merkle key exchange (DHM)

Found in: Component config > mbedTLS

Enable DHM. Needed to use DHE-xxx TLS ciphersuites.

Note that the security of Diffie-Hellman key exchanges depends on a suitable prime being used for the exchange. Please see detailed warning text about this in file *mbedtls/dhm.h* file.

Default value:

- No (disabled)

CONFIG_MBEDTLS_ECDH_C

Elliptic Curve Diffie-Hellman (ECDH)

Found in: Component config > mbedTLS

Enable ECDH. Needed to use ECDHE-xxx TLS ciphersuites.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_ECDSA_C

Elliptic Curve DSA

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECDH_C

Enable ECDSA. Needed to use ECDSA-xxx TLS ciphersuites.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_ECJPAKE_C

Elliptic curve J-PAKE

Found in: Component config > mbedTLS

Enable ECJPAKE. Needed to use ECJPAKE-xxx TLS ciphersuites.

Default value:

- No (disabled)

CONFIG_MBEDTLS_ECP_DP_SECP192R1_ENABLED

Enable SECP192R1 curve

Found in: Component config > mbedTLS

Enable support for SECP192R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP224R1_ENABLED

Enable SECP224R1 curve

Found in: Component config > mbedTLS

Enable support for SECP224R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP256R1_ENABLED

Enable SECP256R1 curve

Found in: Component config > mbedTLS

Enable support for SECP256R1 Elliptic Curve.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_ECP_DP_SECP384R1_ENABLED

Enable SECP384R1 curve

Found in: Component config > mbedTLS

Enable support for SECP384R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP521R1_ENABLED

Enable SECP521R1 curve

Found in: Component config > mbedTLS

Enable support for SECP521R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP192K1_ENABLED

Enable SECP192K1 curve

Found in: Component config > mbedTLS

Enable support for SECP192K1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP224K1_ENABLED

Enable SECP224K1 curve

Found in: Component config > mbedTLS

Enable support for SECP224K1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP256K1_ENABLED

Enable SECP256K1 curve

Found in: Component config > mbedTLS

Enable support for SECP256K1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_BP256R1_ENABLED

Enable BP256R1 curve

Found in: Component config > mbedTLS

support for DP Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_BP384R1_ENABLED

Enable BP384R1 curve

Found in: Component config > mbedTLS

support for DP Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_BP512R1_ENABLED

Enable BP512R1 curve

Found in: Component config > mbedTLS

support for DP Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_CURVE25519_ENABLED

Enable CURVE25519 curve

Found in: Component config > mbedTLS

Enable support for CURVE25519 Elliptic Curve.

CONFIG_MBEDTLS_ECP_NIST_OPTIM

NIST 'modulo p' optimisations

Found in: Component config > mbedTLS

NIST 'modulo p' optimisations increase Elliptic Curve operation performance.

Disabling this option saves some code size.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_ECP_FIXED_POINT_OPTIM

Enable fixed-point multiplication optimisations

Found in: Component config > mbedTLS

This configuration option enables optimizations to speedup (about 3 ~ 4 times) the ECP fixed point multiplication using pre-computed tables in the flash memory. Enabling this configuration option increases the flash footprint (about 29KB if all Elliptic Curve selected) in the application binary.

end of Elliptic Curve options

Default value:

- No (disabled)

CONFIG_MBEDTLS_POLY1305_C

Poly1305 MAC algorithm

Found in: Component config > mbedTLS

Enable support for Poly1305 MAC algorithm.

Default value:

- No (disabled)

CONFIG_MBEDTLS_CHACHA20_C

Chacha20 stream cipher

Found in: Component config > mbedTLS

Enable support for Chacha20 stream cipher.

Default value:

- No (disabled)

CONFIG_MBEDTLS_CHACHAPOLY_C

ChaCha20-Poly1305 AEAD algorithm

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_CHACHA20_C

Enable support for ChaCha20-Poly1305 AEAD algorithm.

Default value:

- No (disabled) if *CONFIG_MBEDTLS_CHACHA20_C* && *CONFIG_MBEDTLS_POLY1305_C*

CONFIG_MBEDTLS_HKDF_C

HKDF algorithm (RFC 5869)

Found in: Component config > mbedTLS

Enable support for the Hashed Message Authentication Code (HMAC)-based key derivation function (HKDF).

Default value:

- No (disabled)

CONFIG_MBEDTLS_THREADING_C

Enable the threading abstraction layer

Found in: Component config > mbedTLS

If you do intend to use contexts between threads, you will need to enable this layer to prevent race conditions.

Default value:

- No (disabled)

CONFIG_MBEDTLS_THREADING_ALT

Enable threading alternate implementation

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_THREADING_C

Enable threading alt to allow your own alternate threading implementation.

Default value:

- Yes (enabled) if *CONFIG_MBEDTLS_THREADING_C*

CONFIG_MBEDTLS_THREADING_PTHREAD

Enable threading pthread implementation

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_THREADING_C

Enable the pthread wrapper layer for the threading layer.

Default value:

- No (disabled) if *CONFIG_MBEDTLS_THREADING_C*

CONFIG_MBEDTLS_ERROR_STRINGS

Enable error code to error string conversion

Found in: Component config > mbedTLS

Enables mbedtls_strerror() for converting error codes to error strings. Disabling this config can save some code/rodata size as the error string conversion implementation is replaced with an empty stub.

Default value:

- Yes (enabled)

CONFIG_MBEDTLS_USE_CRYPTO_ROM_IMPL

Use ROM implementation of the crypto algorithm

Found in: Component config > mbedTLS

Enable this flag to use mbedtls crypto algorithm from ROM instead of ESP-IDF.

This configuration option saves flash footprint in the application binary. Note that the version of mbedtls crypto algorithm library in ROM(ECO1~ECO3) is v2.16.12, and the version of mbedtls crypto algorithm library in ROM(ECO4) is v3.6.0. We have done the security analysis of the mbedtls revision in ROM (ECO1~ECO4) and ensured that affected symbols have been patched (removed). If in the future mbedtls revisions there are security issues that also affects the version in ROM (ECO1~ECO4) then we shall patch the relevant symbols. This would increase the flash footprint and hence care must be taken to keep some reserved space for the application binary in flash layout.

Default value:

- No (disabled) if ESP_ROM_HAS_MBEDTLS_CRYPTO_LIB

CONFIG_MBEDTLS_FS_IO

Enable functions that use the filesystem

Found in: Component config > mbedTLS

This option enables functions in mbedTLS that use the filesystem. It uses the default filesystem support for the target, which is added through vfs component for ESP32 based targets or by the host system when the target is Linux.

Default value:

- Yes (enabled)

ESP-MQTT Configurations

Contains:

- *CONFIG_MQTT_CUSTOM_OUTBOX*
- *CONFIG_MQTT_TRANSPORT_SSL*
- *CONFIG_MQTT_TRANSPORT_WEBSOCKET*
- *CONFIG_MQTT_PROTOCOL_311*
- *CONFIG_MQTT_PROTOCOL_5*
- *CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED*
- *CONFIG_MQTT_USE_CUSTOM_CONFIG*
- *CONFIG_MQTT_OUTBOX_EXPIRED_TIMEOUT_MS*
- *CONFIG_MQTT_REPORT_DELETED_MESSAGES*
- *CONFIG_MQTT_SKIP_PUBLISH_IF_DISCONNECTED*
- *CONFIG_MQTT_OUTBOX_DATA_ON_EXTERNAL_MEMORY*
- *CONFIG_MQTT_MSG_ID_INCREMENTAL*

CONFIG_MQTT_PROTOCOL_311

Enable MQTT protocol 3.1.1

Found in: Component config > ESP-MQTT Configurations

If not, this library will use MQTT protocol 3.1

Default value:

- Yes (enabled)

CONFIG_MQTT_PROTOCOL_5

Enable MQTT protocol 5.0

Found in: Component config > ESP-MQTT Configurations

If not, this library will not support MQTT 5.0

Default value:

- No (disabled)

CONFIG_MQTT_TRANSPORT_SSL

Enable MQTT over SSL

Found in: Component config > ESP-MQTT Configurations

Enable MQTT transport over SSL with mbedtls

Default value:

- Yes (enabled)

CONFIG_MQTT_TRANSPORT_WEBSOCKET

Enable MQTT over Websocket

Found in: Component config > ESP-MQTT Configurations

Enable MQTT transport over Websocket.

Default value:

- Yes (enabled)

CONFIG_MQTT_TRANSPORT_WEBSOCKET_SECURE

Enable MQTT over Websocket Secure

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_TRANSPORT_WEBSOCKET

Enable MQTT transport over Websocket Secure.

Default value:

- Yes (enabled)

CONFIG_MQTT_MSG_ID_INCREMENTAL

Use Incremental Message Id

Found in: Component config > ESP-MQTT Configurations

Set this to true for the message id (2.3.1 Packet Identifier) to be generated as an incremental number rather then a random value (used by default)

Default value:

- No (disabled)

CONFIG_MQTT_SKIP_PUBLISH_IF_DISCONNECTED

Skip publish if disconnected

Found in: Component config > ESP-MQTT Configurations

Set this to true to avoid publishing (enqueueing messages) if the client is disconnected. The MQTT client tries to publish all messages by default, even in the disconnected state (where the qos1 and qos2 packets are stored in the internal outbox to be published later) The

MQTT_SKIP_PUBLISH_IF_DISCONNECTED option allows applications to override this behaviour and not enqueue publish packets in the disconnected state.

Default value:

- No (disabled)

CONFIG_MQTT_REPORT_DELETED_MESSAGES

Report deleted messages

Found in: Component config > ESP-MQTT Configurations

Set this to true to post events for all messages which were deleted from the outbox before being correctly sent and confirmed.

Default value:

- No (disabled)

CONFIG_MQTT_USE_CUSTOM_CONFIG

MQTT Using custom configurations

Found in: Component config > ESP-MQTT Configurations

Custom MQTT configurations.

Default value:

- No (disabled)

CONFIG_MQTT_TCP_DEFAULT_PORT

Default MQTT over TCP port

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default MQTT over TCP port

Default value:

- 1883 if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

CONFIG_MQTT_SSL_DEFAULT_PORT

Default MQTT over SSL port

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default MQTT over SSL port

Default value:

- 8883 if *CONFIG_MQTT_USE_CUSTOM_CONFIG* && *CONFIG_MQTT_TRANSPORT_SSL*

CONFIG_MQTT_WS_DEFAULT_PORT

Default MQTT over Websocket port

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default MQTT over Websocket port

Default value:

- 80 if *CONFIG_MQTT_USE_CUSTOM_CONFIG* && *CONFIG_MQTT_TRANSPORT_WEBSOCKET*

CONFIG_MQTT_WSS_DEFAULT_PORT

Default MQTT over Websocket Secure port

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default MQTT over Websocket Secure port

Default value:

- 443 if *CONFIG_MQTT_USE_CUSTOM_CONFIG* && *CONFIG_MQTT_TRANSPORT_WEBSOCKET* && *CONFIG_MQTT_TRANSPORT_WEBSOCKET_SECURE*

CONFIG_MQTT_BUFFER_SIZE

Default MQTT Buffer Size

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

This buffer size using for both transmit and receive

Default value:

- 1024 if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

CONFIG_MQTT_TASK_STACK_SIZE

MQTT task stack size

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

MQTT task stack size

Default value:

- 6144 if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

CONFIG_MQTT_DISABLE_API_LOCKS

Disable API locks

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default config employs API locks to protect internal structures. It is possible to disable these locks if the user code doesn't access MQTT API from multiple concurrent tasks

Default value:

- No (disabled) if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

CONFIG_MQTT_TASK_PRIORITY

MQTT task priority

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

MQTT task priority. Higher number denotes higher priority.

Default value:

- 5 if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

CONFIG_MQTT_POLL_READ_TIMEOUT_MS

MQTT transport poll read timeut

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Timeout when polling underlying transport for read.

Default value:

- 1000 if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

CONFIG_MQTT_EVENT_QUEUE_SIZE

Number of queued events.

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

A value higher than 1 enables multiple queued events.

Default value:

- 1 if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED

Enable MQTT task core selection

Found in: Component config > ESP-MQTT Configurations

This will enable core selection

CONFIG_MQTT_TASK_CORE_SELECTION

Core to use ?

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED

Available options:

- Core 0 (CONFIG_MQTT_USE_CORE_0)
- Core 1 (CONFIG_MQTT_USE_CORE_1)

CONFIG_MQTT_OUTBOX_DATA_ON_EXTERNAL_MEMORY

Use external memory for outbox data

Found in: Component config > ESP-MQTT Configurations

Set to true to use external memory for outbox data.

Default value:

- No (disabled) if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

CONFIG_MQTT_CUSTOM_OUTBOX

Enable custom outbox implementation

Found in: Component config > ESP-MQTT Configurations

Set to true if a specific implementation of message outbox is needed (e.g. persistent outbox in NVM or similar). Note: Implementation of the custom outbox must be added to the mqtt component. These CMake commands could be used to append the custom implementation to lib-mqtt sources: idf_component_get_property(mqtt mqtt COMPONENT_LIB) set_property(TARGET ${mqtt} PROPERTY SOURCES ${PROJECT_DIR}/custom_outbox.c APPEND)

Default value:

- No (disabled)

CONFIG_MQTT_OUTBOX_EXPIRED_TIMEOUT_MS

Outbox message expired timeout[ms]

Found in: Component config > ESP-MQTT Configurations

Messages which stays in the outbox longer than this value before being published will be discarded.

Default value:

- 30000 if *CONFIG_MQTT_USE_CUSTOM_CONFIG*

Newlib Contains:

- *CONFIG_NEWLIB_NANO_FORMAT*
- *CONFIG_NEWLIB_STDIN_LINE_ENDING*
- *CONFIG_NEWLIB_STDOUT_LINE_ENDING*
- *CONFIG_NEWLIB_TIME_SYSCALL*

CONFIG_NEWLIB_STDOUT_LINE_ENDING

Line ending for UART output

Found in: Component config > Newlib

This option allows configuring the desired line endings sent to UART when a newline ('n', LF) appears on stdout. Three options are possible:

CRLF: whenever LF is encountered, prepend it with CR

LF: no modification is applied, stdout is sent as is

CR: each occurrence of LF is replaced with CR

This option doesn't affect behavior of the UART driver (drivers/uart.h).

Available options:

- CRLF (CONFIG_NEWLIB_STDOUT_LINE_ENDING_CRLF)
- LF (CONFIG_NEWLIB_STDOUT_LINE_ENDING_LF)
- CR (CONFIG_NEWLIB_STDOUT_LINE_ENDING_CR)

CONFIG_NEWLIB_STDIN_LINE_ENDING

Line ending for UART input

Found in: Component config > Newlib

This option allows configuring which input sequence on UART produces a newline ('n', LF) on stdin. Three options are possible:

CRLF: CRLF is converted to LF

LF: no modification is applied, input is sent to stdin as is

CR: each occurrence of CR is replaced with LF

This option doesn't affect behavior of the UART driver (drivers/uart.h).

Available options:

- CRLF (CONFIG_NEWLIB_STDIN_LINE_ENDING_CRLF)
- LF (CONFIG_NEWLIB_STDIN_LINE_ENDING_LF)
- CR (CONFIG_NEWLIB_STDIN_LINE_ENDING_CR)

CONFIG_NEWLIB_NANO_FORMAT

Enable 'nano' formatting options for printf/scanf family

Found in: Component config > Newlib

In most chips the ROM contains parts of newlib C library, including printf/scanf family of functions. These functions have been compiled with so-called "nano" formatting option. This option doesn't support 64-bit integer formats and C99 features, such as positional arguments.

For more details about "nano" formatting option, please see newlib readme file, search for '--enable-newlib-nano-formatted-io': https://sourceware.org/git/?p=newlib-cygwin.git;a=blob_plain;f=newlib/README;hb=HEAD

If this option is enabled and the ROM contains functions from newlib-nano, the build system will use functions available in ROM, reducing the application binary size. Functions available in ROM run faster than functions which run from flash. Functions available in ROM can also run when flash instruction cache is disabled.

Some chips (e.g. ESP32-C6) has the full formatting versions of printf/scanf in ROM instead of the nano versions and in this building with newlib nano might actually increase the size of the binary. Which functions are present in ROM can be seen from ROM caps: ESP_ROM_HAS_NEWLIB_NANO_FORMAT and ESP_ROM_HAS_NEWLIB_NORMAL_FORMAT.

If you need 64-bit integer formatting support or C99 features, keep this option disabled.

CONFIG_NEWLIB_TIME_SYSCALL

Timers used for gettimeofday function

Found in: Component config > Newlib

This setting defines which hardware timers are used to implement 'gettimeofday' and 'time' functions in C library.

- **If both high-resolution (systimer for all targets except ESP32)** and RTC timers are used, timekeeping will continue in deep sleep. Time will be reported at 1 microsecond resolution. This is the default, and the recommended option.
- **If only high-resolution timer (systimer) is used, gettimeofday will** provide time at microsecond resolution. Time will not be preserved when going into deep sleep mode.
- **If only RTC timer is used, timekeeping will continue in** deep sleep, but time will be measured at 6.(6) microsecond resolution. Also the gettimeofday function itself may take longer to run.
- **If no timers are used, gettimeofday and time functions** return -1 and set errno to ENOSYS; they are defined as weak, so they could be overridden. If you want to customize gettimeofday() and other time functions, please choose this option and refer to the 'time.c' source file for the exact prototypes of these functions.
- **When RTC is used for timekeeping, two RTC_STORE registers are** used to keep time in deep sleep mode.

Available options:

- RTC and high-resolution timer (CONFIG_NEWLIB_TIME_SYSCALL_USE_RTC_HRT)
- RTC (CONFIG_NEWLIB_TIME_SYSCALL_USE_RTC)
- High-resolution timer (CONFIG_NEWLIB_TIME_SYSCALL_USE_HRT)
- None (CONFIG_NEWLIB_TIME_SYSCALL_USE_NONE)

NVS Contains:

- *CONFIG_NVS_LEGACY_DUP_KEYS_COMPATIBILITY*
- *CONFIG_NVS_ENCRYPTION*
- *CONFIG_NVS_COMPATIBLE_PRE_V4_3_ENCRYPTION_FLAG*
- *CONFIG_NVS_ASSERT_ERROR_CHECK*

CONFIG_NVS_ENCRYPTION

Enable NVS encryption

Found in: Component config > NVS

This option enables encryption for NVS. When enabled, XTS-AES is used to encrypt the complete NVS data, except the page headers. It requires XTS encryption keys to be stored in an encrypted partition (enabling flash encryption is mandatory here) or to be derived from an HMAC key burnt in eFuse.

Default value:

- Yes (enabled) if *CONFIG_SECURE_FLASH_ENC_ENABLED*

CONFIG_NVS_COMPATIBLE_PRE_V4_3_ENCRYPTION_FLAG

NVS partition encrypted flag compatible with ESP-IDF before v4.3

Found in: Component config > NVS

Enabling this will ignore ”encrypted” flag for NVS partitions. NVS encryption scheme is different than hardware flash encryption and hence it is not recommended to have ”encrypted” flag for NVS partitions. This was not being checked in pre v4.3 IDF. Hence, if you have any devices where this flag is kept enabled in partition table then enabling this config will allow to have same behavior as pre v4.3 IDF.

CONFIG_NVS_ASSERT_ERROR_CHECK

Use assertions for error checking

Found in: Component config > NVS

This option switches error checking type between assertions (y) or return codes (n).

Default value:

- No (disabled)

CONFIG_NVS_LEGACY_DUP_KEYS_COMPATIBILITY

Enable legacy nvs_set function behavior when same key is reused with different data types

Found in: Component config > NVS

Enabling this option will switch the nvs_set() family of functions to the legacy mode: when called repeatedly with the same key but different data type, the existing value in the NVS remains active and the new value is just stored, actually not accessible through corresponding nvs_get() call for the key given. Use this option only when your application relies on such NVS API behaviour.

Default value:

- No (disabled)

NVS Security Provider

Contains:

- *CONFIG_NVS_SEC_HMAC_EFUSE_KEY_ID*
- *CONFIG_NVS_SEC_KEY_PROTECTION_SCHEME*

CONFIG_NVS_SEC_KEY_PROTECTION_SCHEME

NVS Encryption: Key Protection Scheme

Found in: Component config > NVS Security Provider

This choice defines the default NVS encryption keys protection scheme; which will be used for the default NVS partition. Users can use the corresponding scheme registration APIs to register other schemes for the default as well as other NVS partitions.

Available options:

- Using Flash Encryption (CONFIG_NVS_SEC_KEY_PROTECT_USING_FLASH_ENC) Protect the NVS Encryption Keys using Flash Encryption Requires a separate 'nvs_keys' partition (which will be encrypted by flash encryption) for storing the NVS encryption keys
- Using HMAC peripheral (CONFIG_NVS_SEC_KEY_PROTECT_USING_HMAC) Derive and protect the NVS Encryption Keys using the HMAC peripheral Requires the specified eFuse block (NVS_SEC_HMAC_EFUSE_KEY_ID or the v2 API argument) to be empty or pre-written with a key with the purpose ESP_EFUSE_KEY_PURPOSE_HMAC_UP

CONFIG_NVS_SEC_HMAC_EFUSE_KEY_ID

eFuse key ID storing the HMAC key

Found in: Component config > NVS Security Provider

eFuse block key ID storing the HMAC key for deriving the NVS encryption keys

Note: The eFuse block key ID required by the HMAC scheme (CONFIG_NVS_SEC_KEY_PROTECT_USING_HMAC) is set using this config when the default NVS partition is initialized with nvs_flash_init(). The eFuse block key ID can also be set at runtime by passing the appropriate value to the NVS security scheme registration APIs.

Range:

- from 0 to 6 if *CONFIG_NVS_SEC_KEY_PROTECT_USING_HMAC*

Default value:

- 6 if *CONFIG_NVS_SEC_KEY_PROTECT_USING_HMAC*

OpenThread Contains:

- *CONFIG_OPENTHREAD_DEVICE_TYPE*
- *CONFIG_OPENTHREAD_RADIO_TYPE*
- *CONFIG_OPENTHREAD_BORDER_ROUTER*
- *CONFIG_OPENTHREAD_COMMISSIONER*
- *CONFIG_OPENTHREAD_CSL_DEBUG_ENABLE*
- *CONFIG_OPENTHREAD_CSL_ENABLE*
- *CONFIG_OPENTHREAD_DIAG*
- *CONFIG_OPENTHREAD_DNS_CLIENT*
- *CONFIG_OPENTHREAD_DUA_ENABLE*
- *CONFIG_OPENTHREAD_JOINER*
- *CONFIG_OPENTHREAD_LINK_METRICS*
- *CONFIG_OPENTHREAD_MACFILTER_ENABLE*
- *CONFIG_OPENTHREAD_CLI*
- *CONFIG_OPENTHREAD_SPINEL_ONLY*
- *CONFIG_OPENTHREAD_RX_ON_WHEN_IDLE*
- *CONFIG_OPENTHREAD_RADIO_STATS_ENABLE*
- *CONFIG_OPENTHREAD_SRP_CLIENT*
- *CONFIG_OPENTHREAD_TIME_SYNC*
- *CONFIG_OPENTHREAD_RADIO_TREL*
- *CONFIG_OPENTHREAD_NCP_VENDOR_HOOK*
- *CONFIG_OPENTHREAD_MAC_MAX_CSMA_BACKOFFS_DIRECT*
- *CONFIG_OPENTHREAD_ENABLED*
- *OpenThread version message*
- *CONFIG_OPENTHREAD_XTAL_ACCURACY*
- *CONFIG_OPENTHREAD_CSL_UNCERTAIN*
- *CONFIG_OPENTHREAD_CSL_ACCURACY*
- *CONFIG_OPENTHREAD_NUM_MESSAGE_BUFFERS*
- *CONFIG_OPENTHREAD_RCP_TRANSPORT*

- *CONFIG_OPENTHREAD_MLE_MAX_CHILDREN*
- *CONFIG_OPENTHREAD_TMF_ADDR_CACHE_ENTRIES*
- *CONFIG_OPENTHREAD_SPINEL_RX_FRAME_BUFFER_SIZE*
- *CONFIG_OPENTHREAD_UART_BUFFER_SIZE*
- *Thread Address Query Config*
- *Thread Memory Allocation Config*
- *Thread Operational Dataset*
- *CONFIG_OPENTHREAD_DNS64_CLIENT*

CONFIG_OPENTHREAD_ENABLED

OpenThread

Found in: Component config > OpenThread

Select this option to enable OpenThread and show the submenu with OpenThread configuration choices.

Default value:

- No (disabled)

CONFIG_OPENTHREAD_LOG_LEVEL_DYNAMIC

Enable dynamic log level control

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED

Select this option to enable dynamic log level control for OpenThread

Default value:

- Yes (enabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_CONSOLE_TYPE

OpenThread console type

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED

Select OpenThread console type

Available options:

- OpenThread console type UART (CONFIG_OPENTHREAD_CONSOLE_TYPE_UART)
- OpenThread console type USB Serial/JTAG Controller (CONFIG_OPENTHREAD_CONSOLE_TYPE_USB_SERIAL_JTAG)

CONFIG_OPENTHREAD_LOG_LEVEL

OpenThread log verbosity

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED

Select OpenThread log level.

Available options:

- No logs (CONFIG_OPENTHREAD_LOG_LEVEL_NONE)
- Error logs (CONFIG_OPENTHREAD_LOG_LEVEL_CRIT)
- Warning logs (CONFIG_OPENTHREAD_LOG_LEVEL_WARN)
- Notice logs (CONFIG_OPENTHREAD_LOG_LEVEL_NOTE)
- Info logs (CONFIG_OPENTHREAD_LOG_LEVEL_INFO)
- Debug logs (CONFIG_OPENTHREAD_LOG_LEVEL_DEBG)

Thread Operational Dataset Contains:

- *CONFIG_OPENTHREAD_NETWORK_EXTPANID*
- *CONFIG_OPENTHREAD_MESH_LOCAL_PREFIX*
- *CONFIG_OPENTHREAD_NETWORK_CHANNEL*
- *CONFIG_OPENTHREAD_NETWORK_MASTERKEY*
- *CONFIG_OPENTHREAD_NETWORK_NAME*
- *CONFIG_OPENTHREAD_NETWORK_PANID*
- *CONFIG_OPENTHREAD_NETWORK_PSKC*

CONFIG_OPENTHREAD_NETWORK_NAME

OpenThread network name

Found in: Component config > OpenThread > Thread Operational Dataset

Default value:
- ”OpenThread-ESP”

CONFIG_OPENTHREAD_MESH_LOCAL_PREFIX

OpenThread mesh local prefix, format <address>/<plen>

Found in: Component config > OpenThread > Thread Operational Dataset

A string in the format ”<address>/<plen>”, where <*address*> is an IPv6 address and <*plen*> is a prefix length. For example ”fd00:db8:a0:0::/64”

Default value:
- ”fd00:db8:a0:0::/64”

CONFIG_OPENTHREAD_NETWORK_CHANNEL

OpenThread network channel

Found in: Component config > OpenThread > Thread Operational Dataset

Range:
- from 11 to 26

Default value:
- 15

CONFIG_OPENTHREAD_NETWORK_PANID

OpenThread network pan id

Found in: Component config > OpenThread > Thread Operational Dataset

Range:
- from 0 to 0xFFFE

Default value:
- ”0x1234”

CONFIG_OPENTHREAD_NETWORK_EXTPANID

OpenThread extended pan id

Found in: Component config > OpenThread > Thread Operational Dataset

The OpenThread network extended pan id in hex string format

Default value:
- dead00beef00cafe

CONFIG_OPENTHREAD_NETWORK_MASTERKEY

OpenThread network key

Found in: Component config > OpenThread > Thread Operational Dataset

The OpenThread network network key in hex string format

Default value:

- 00112233445566778899aabbccddeeff

CONFIG_OPENTHREAD_NETWORK_PSKC

OpenThread pre-shared commissioner key

Found in: Component config > OpenThread > Thread Operational Dataset

The OpenThread pre-shared commissioner key in hex string format

Default value:

- 104810e2315100afd6bc9215a6bfac53

CONFIG_OPENTHREAD_RADIO_TYPE

Config the Thread radio type with 15.4 link

Found in: Component config > OpenThread

Configure how OpenThread connects to the 15.4 radio

Available options:

- Native 15.4 radio (CONFIG_OPENTHREAD_RADIO_NATIVE)
  Select this to use the native 15.4 radio.
- Connect via UART (CONFIG_OPENTHREAD_RADIO_SPINEL_UART)
  Select this to connect to a Radio Co-Processor via UART.
- Connect via SPI (CONFIG_OPENTHREAD_RADIO_SPINEL_SPI)
  Select this to connect to a Radio Co-Processor via SPI.
- Disable the Thread radio based on 15.4 link (CONFIG_OPENTHREAD_RADIO_154_NONE)
  Select this to disable the Thread radio based on 15.4 link.

CONFIG_OPENTHREAD_RADIO_TREL

Enable Thread Radio Encapsulation Link (TREL)

Found in: Component config > OpenThread

Select this option to enable Thread Radio Encapsulation Link.

Default value:

- No (disabled) if EXAMPLE_CONNECT_WIFI || EXAMPLE_CONNECT_ETHERNET

CONFIG_OPENTHREAD_TREL_PORT

The port of openthread trel service

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_RADIO_TREL

Configure the port number of TREL service.

Default value:

- 12390 if *CONFIG_OPENTHREAD_RADIO_TREL*

CONFIG_OPENTHREAD_DEVICE_TYPE

Config the Thread device type

Found in: Component config > OpenThread

OpenThread can be configured to different device types (FTD, MTD, Radio)

Available options:

- Full Thread Device (CONFIG_OPENTHREAD_FTD)
  Select this to enable Full Thread Device which can act as router and leader in a Thread network.
- Minimal Thread Device (CONFIG_OPENTHREAD_MTD)
  Select this to enable Minimal Thread Device which can only act as end device in a Thread network. This will reduce the code size of the OpenThread stack.
- Radio Only Device (CONFIG_OPENTHREAD_RADIO)
  Select this to enable Radio Only Device which can only forward 15.4 packets to the host. The OpenThread stack will be run on the host and OpenThread will have minimal footprint on the radio only device.

CONFIG_OPENTHREAD_RCP_TRANSPORT

The RCP transport type

Found in: Component config > OpenThread

Available options:

- UART RCP (CONFIG_OPENTHREAD_RCP_UART)
  Select this to enable UART connection to host.
- SPI RCP (CONFIG_OPENTHREAD_RCP_SPI)
  Select this to enable SPI connection to host.

OpenThread version message

Contains:

- *CONFIG_OPENTHREAD_PACKAGE_NAME*
- *CONFIG_OPENTHREAD_PLATFORM_INFO*

CONFIG_OPENTHREAD_PACKAGE_NAME

OpenThread package name

Found in: Component config > OpenThread > OpenThread version message

The OpenThread package name.

Default value:

- ”openthread-esp32” if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_PLATFORM_INFO

platform information

Found in: Component config > OpenThread > OpenThread version message

The OpenThread platform information.

Default value:

- ”esp32p4” if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_NCP_VENDOR_HOOK

Enable vendor command for RCP

Found in: Component config > OpenThread

Select this to enable OpenThread NCP vendor commands.

Default value:

- Yes (enabled) if *CONFIG_OPENTHREAD_RADIO*

CONFIG_OPENTHREAD_CLI

Enable Openthread Command-Line Interface

Found in: Component config > OpenThread

Select this option to enable Command-Line Interface in OpenThread.

Default value:

- Yes (enabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_DIAG

Enable diag

Found in: Component config > OpenThread

Select this option to enable Diag in OpenThread. This will enable diag mode and a series of diag commands in the OpenThread command line. These commands allow users to manipulate low-level features of the storage and 15.4 radio.

Default value:

- Yes (enabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_COMMISSIONER

Enable Commissioner

Found in: Component config > OpenThread

Select this option to enable commissioner in OpenThread. This will enable the device to act as a commissioner in the Thread network. A commissioner checks the pre-shared key from a joining device with the Thread commissioning protocol and shares the network parameter with the joining device upon success.

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_COMM_MAX_JOINER_ENTRIES

The size of max commissioning joiner entries

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_COMMISSIONER

Default value:

- 2 if *CONFIG_OPENTHREAD_COMMISSIONER*

CONFIG_OPENTHREAD_JOINER

Enable Joiner

Found in: Component config > OpenThread

Select this option to enable Joiner in OpenThread. This allows a device to join the Thread network with a pre-shared key using the Thread commissioning protocol.

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_SRP_CLIENT

Enable SRP Client

Found in: Component config > OpenThread

Select this option to enable SRP Client in OpenThread. This allows a device to register SRP services to SRP Server.

Default value:

- Yes (enabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_SRP_CLIENT_MAX_SERVICES

Specifies number of service entries in the SRP client service pool

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_SRP_CLIENT

Set the max buffer size of service entries in the SRP client service pool.

Default value:

- 5 if *CONFIG_OPENTHREAD_SRP_CLIENT*

CONFIG_OPENTHREAD_DNS_CLIENT

Enable DNS Client

Found in: Component config > OpenThread

Select this option to enable DNS Client in OpenThread.

Default value:

- Yes (enabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_BORDER_ROUTER

Enable Border Router

Found in: Component config > OpenThread

Select this option to enable border router features in OpenThread.

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED*

Thread Memory Allocation Config

Contains:

- *CONFIG_OPENTHREAD_MEM_ALLOC_EXTERNAL*
- *CONFIG_OPENTHREAD_PLATFORM_MSGPOOL_MANAGEMENT*

CONFIG_OPENTHREAD_MEM_ALLOC_EXTERNAL

Allocate memory from PSRAM

Found in: Component config > OpenThread > Thread Memory Allocation Config

Select this option to allocate buffer from PSRAM for Thread

Default value:

- Yes (enabled) if *CONFIG_OPENTHREAD_ENABLED* && (*CONFIG_SPIRAM_USE_CAPS_ALLOC* || *CONFIG_SPIRAM_USE_MALLOC*)

CONFIG_OPENTHREAD_PLATFORM_MSGPOOL_MANAGEMENT

Allocate message pool buffer from PSRAM

Found in: Component config > OpenThread > Thread Memory Allocation Config

If enabled, the message pool is managed by platform defined logic.

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED* && (*CONFIG_SPIRAM_USE_CAPS_ALLOC* || *CONFIG_SPIRAM_USE_MALLOC*)

CONFIG_OPENTHREAD_NUM_MESSAGE_BUFFERS

The number of openthread message buffers

Found in: Component config > OpenThread

Default value:

- 65 if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_SPINEL_RX_FRAME_BUFFER_SIZE

The size of openthread spinel rx frame buffer

Found in: Component config > OpenThread

Default value:

- 1024 if (*CONFIG_OPENTHREAD_MTD* || *CONFIG_OPENTHREAD_RADIO*) && (*CONFIG_OPENTHREAD_ENABLED* || *CONFIG_OPENTHREAD_SPINEL_ONLY*)
- 2048 if (*CONFIG_OPENTHREAD_FTD* || *CONFIG_OPENTHREAD_SPINEL_ONLY*) && (*CONFIG_OPENTHREAD_ENABLED* || *CONFIG_OPENTHREAD_SPINEL_ONLY*)

CONFIG_OPENTHREAD_MAC_MAX_CSMA_BACKOFFS_DIRECT

Maximum backoffs times before declaring a channel access failure.

Found in: Component config > OpenThread

The maximum number of backoffs the CSMA-CA algorithm will attempt before declaring a channel access failure.

Default value:

- 4 if *CONFIG_OPENTHREAD_ENABLED* || *CONFIG_OPENTHREAD_SPINEL_ONLY*

CONFIG_OPENTHREAD_MLE_MAX_CHILDREN

The size of max MLE children entries

Found in: Component config > OpenThread

Default value:

- 10 if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_TMF_ADDR_CACHE_ENTRIES

The size of max TMF address cache entries

Found in: Component config > OpenThread

Default value:

- 20 if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_DNS64_CLIENT

Use dns64 client

Found in: Component config > OpenThread

Select this option to acquire NAT64 address from dns servers.

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED* && *CONFIG_LWIP_IPV4*

CONFIG_OPENTHREAD_DNS_SERVER_ADDR

DNS server address (IPv4)

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_DNS64_CLIENT

Set the DNS server IPv4 address.

Default value:

- "8.8.8.8" if *CONFIG_OPENTHREAD_DNS64_CLIENT*

CONFIG_OPENTHREAD_UART_BUFFER_SIZE

The uart received buffer size of openthread

Found in: Component config > OpenThread

Set the OpenThread UART buffer size.

Default value:

- 2048 if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_LINK_METRICS

Enable link metrics feature

Found in: Component config > OpenThread

Select this option to enable link metrics feature

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_MACFILTER_ENABLE

Enable mac filter feature

Found in: Component config > OpenThread

Select this option to enable mac filter feature

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_CSL_ENABLE

Enable CSL feature

Found in: Component config > OpenThread

Select this option to enable CSL feature

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_XTAL_ACCURACY

The accuracy of the XTAL

Found in: Component config > OpenThread

The device's XTAL accuracy, in ppm.

Default value:

- 130

CONFIG_OPENTHREAD_CSL_ACCURACY

The current CSL rx/tx scheduling drift, in units of ± ppm

Found in: Component config > OpenThread

The current accuracy of the clock used for scheduling CSL operations

Default value:

- 1 if *CONFIG_OPENTHREAD_CSL_ENABLE*

CONFIG_OPENTHREAD_CSL_UNCERTAIN

The CSL Uncertainty in units of 10 us.

Found in: Component config > OpenThread

The fixed uncertainty of the Device for scheduling CSL Transmissions in units of 10 microseconds.

Default value:

- 1 if *CONFIG_OPENTHREAD_CSL_ENABLE*

CONFIG_OPENTHREAD_CSL_DEBUG_ENABLE

Enable CSL debug

Found in: Component config > OpenThread

Select this option to set rx on when sleep in CSL feature, only for debug

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_CSL_ENABLE*

CONFIG_OPENTHREAD_DUA_ENABLE

Enable Domain Unicast Address feature

Found in: Component config > OpenThread

Only used for Thread1.2 certification

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_TIME_SYNC

Enable the time synchronization service feature

Found in: Component config > OpenThread

Select this option to enable time synchronization feature, the devices in the same Thread network could sync to the same network time.

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_ENABLED*

CONFIG_OPENTHREAD_RADIO_STATS_ENABLE

Enable Radio Statistics feature

Found in: Component config > OpenThread

Select this option to enable the radio statistics feature, you can use radio command to print some radio Statistics information.

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_FTD* || *CONFIG_OPENTHREAD_MTD*

CONFIG_OPENTHREAD_SPINEL_ONLY

Enable OpenThread External Radio Spinel feature

Found in: Component config > OpenThread

Select this option to enable the OpenThread Radio Spinel for external protocol stack, such as Zigbee.

Default value:

- No (disabled)

CONFIG_OPENTHREAD_RX_ON_WHEN_IDLE

Enable OpenThread radio capability rx on when idle

Found in: Component config > OpenThread

Select this option to enable OpenThread radio capability rx on when idle. Do not support this feature when SW coexistence is enabled.

Default value:

- No (disabled) if *CONFIG_ESP_COEX_SW_COEXIST_ENABLE*

Thread Address Query Config Contains:

- *CONFIG_OPENTHREAD_ADDRESS_QUERY_RETRY_DELAY*
- *CONFIG_OPENTHREAD_ADDRESS_QUERY_MAX_RETRY_DELAY*
- *CONFIG_OPENTHREAD_ADDRESS_QUERY_TIMEOUT*

CONFIG_OPENTHREAD_ADDRESS_QUERY_TIMEOUT

Timeout value (in seconds) for a address notification response after sending an address query.

Found in: Component config > OpenThread > Thread Address Query Config

Default value:

- 3 if *CONFIG_OPENTHREAD_FTD* || *CONFIG_OPENTHREAD_MTD*

CONFIG_OPENTHREAD_ADDRESS_QUERY_RETRY_DELAY

Initial retry delay for address query (in seconds).

Found in: Component config > OpenThread > Thread Address Query Config

Default value:

- 15 if *CONFIG_OPENTHREAD_FTD* || *CONFIG_OPENTHREAD_MTD*

CONFIG_OPENTHREAD_ADDRESS_QUERY_MAX_RETRY_DELAY

Maximum retry delay for address query (in seconds).

Found in: Component config > OpenThread > Thread Address Query Config

Default value:

- 120 if *CONFIG_OPENTHREAD_FTD* || *CONFIG_OPENTHREAD_MTD*

Protocomm Contains:

- *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0*
- *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1*
- *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2*

CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0

Support protocomm security version 0 (no security)

Found in: Component config > Protocomm

Enable support of security version 0. Disabling this option saves some code size. Consult the Enabling protocomm security version section of the Protocomm documentation in ESP-IDF Programming guide for more details.

Default value:

- Yes (enabled)

CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1

Support protocomm security version 1 (Curve25519 key exchange + AES-CTR encryption/decryption)

Found in: Component config > Protocomm

Enable support of security version 1. Disabling this option saves some code size. Consult the Enabling protocomm security version section of the Protocomm documentation in ESP-IDF Programming guide for more details.

Default value:

- Yes (enabled)

CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2

Support protocomm security version 2 (SRP6a-based key exchange + AES-GCM encryption/decryption)

Found in: Component config > Protocomm

Enable support of security version 2. Disabling this option saves some code size. Consult the Enabling protocomm security version section of the Protocomm documentation in ESP-IDF Programming guide for more details.

Default value:

- Yes (enabled)

PThreads Contains:

- *CONFIG_PTHREAD_TASK_NAME_DEFAULT*
- *CONFIG_PTHREAD_TASK_CORE_DEFAULT*
- *CONFIG_PTHREAD_TASK_PRIO_DEFAULT*
- *CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT*
- *CONFIG_PTHREAD_STACK_MIN*

CONFIG_PTHREAD_TASK_PRIO_DEFAULT

Default task priority

Found in: Component config > PThreads

Priority used to create new tasks with default pthread parameters.

Range:
- from 0 to 255

Default value:
- 5

CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT

Default task stack size

Found in: Component config > PThreads

Stack size used to create new tasks with default pthread parameters.

Default value:
- 3072

CONFIG_PTHREAD_STACK_MIN

Minimum allowed pthread stack size

Found in: Component config > PThreads

Minimum allowed pthread stack size set in attributes passed to pthread_create

Default value:
- 768

CONFIG_PTHREAD_TASK_CORE_DEFAULT

Default pthread core affinity

Found in: Component config > PThreads

The default core to which pthreads are pinned.

Available options:

- No affinity (CONFIG_PTHREAD_DEFAULT_CORE_NO_AFFINITY)
- Core 0 (CONFIG_PTHREAD_DEFAULT_CORE_0)
- Core 1 (CONFIG_PTHREAD_DEFAULT_CORE_1)

CONFIG_PTHREAD_TASK_NAME_DEFAULT

Default name of pthreads

Found in: Component config > PThreads

The default name of pthreads.

Default value:
- ”pthread”

SoC Settings Contains:

- *MMU Config*

MMU Config

Main Flash configuration Contains:

- *Optional and Experimental Features (READ DOCS FIRST)*
- *SPI Flash behavior when brownout*

SPI Flash behavior when brownout Contains:

- *CONFIG_SPI_FLASH_BROWNOUT_RESET_XMC*

CONFIG_SPI_FLASH_BROWNOUT_RESET_XMC

Enable sending reset when brownout for XMC flash chips

Found in: Component config > Main Flash configuration > SPI Flash behavior when brownout

When this option is selected, the patch will be enabled for XMC. Follow the recommended flow by XMC for better stability.

DO NOT DISABLE UNLESS YOU KNOW WHAT YOU ARE DOING.

Optional and Experimental Features (READ DOCS FIRST) Contains:

- *CONFIG_SPI_FLASH_AUTO_SUSPEND*
- *CONFIG_SPI_FLASH_FORCE_ENABLE_XMC_C_SUSPEND*
- *CONFIG_SPI_FLASH_SUSPEND_TSUS_VAL_US*
- *CONFIG_SPI_FLASH_HPM_DC*

CONFIG_SPI_FLASH_HPM_DC

Support HPM using DC (READ DOCS FIRST)

Found in: Component config > Main Flash configuration > Optional and Experimental Features (READ DOCS FIRST)

This feature needs your bootloader to be compiled DC-aware (BOOTLOADER_FLASH_DC_AWARE=y). Otherwise the chip will not be able to boot after a reset.

Available options:

- Auto (Enable when bootloader support enabled (BOOTLOADER_FLASH_DC_AWARE)) (CONFIG_SPI_FLASH_HPM_DC_AUTO)
- Disable (READ DOCS FIRST) (CONFIG_SPI_FLASH_HPM_DC_DISABLE)

CONFIG_SPI_FLASH_AUTO_SUSPEND

Auto suspend long erase/write operations (READ DOCS FIRST)

Found in: Component config > Main Flash configuration > Optional and Experimental Features (READ DOCS FIRST)

This option is disabled by default because it is supported only for specific flash chips and for specific Espressif chips. To evaluate if you can use this feature refer to *Optional Features for Flash > Auto Suspend & Resume* of the *ESP-IDF Programming Guide*.

CAUTION: If you want to OTA to an app with this feature turned on, please make sure the bootloader has the support for it. (later than IDF v4.3)

If you are using an official Espressif module, please contact Espressif Business support to check if the module has the flash that support this feature installed. Also refer to *Concurrency Constraints for Flash on SPI1 > Flash Auto Suspend Feature* before enabling this option.

CONFIG_SPI_FLASH_SUSPEND_TSUS_VAL_US

SPI flash tSUS value (refer to chapter AC CHARACTERISTICS)

Found in: Component config > Main Flash configuration > Optional and Experimental Features (READ DOCS FIRST)

This config is used for setting Tsus parameter. Tsus means CS# high to next command after suspend. You can refer to the chapter of AC CHARACTERISTICS of flash datasheet.

CONFIG_SPI_FLASH_FORCE_ENABLE_XMC_C_SUSPEND

Enable XMC-C series flash chip suspend feature anyway

Found in: Component config > Main Flash configuration > Optional and Experimental Features (READ DOCS FIRST)

XMC-C series is regarded as not qualified for the Suspend feature, since its specification has a tRS >= 1ms restriction. We strongly do not suggest using it for the Suspend feature. However, if your product in field has enabled this feature, you may still enable this config option to keep the legacy behavior.

For new users, DO NOT enable this config.

SPI Flash driver

Contains:

- *Auto-detect flash chips*
- *CONFIG_SPI_FLASH_BYPASS_BLOCK_ERASE*
- *CONFIG_SPI_FLASH_ENABLE_ENCRYPTED_READ_WRITE*
- *CONFIG_SPI_FLASH_ENABLE_COUNTERS*
- *CONFIG_SPI_FLASH_ROM_DRIVER_PATCH*
- *CONFIG_SPI_FLASH_YIELD_DURING_ERASE*
- *CONFIG_SPI_FLASH_CHECK_ERASE_TIMEOUT_DISABLED*
- *CONFIG_SPI_FLASH_WRITE_CHUNK_SIZE*
- *CONFIG_SPI_FLASH_OVERRIDE_CHIP_DRIVER_LIST*
- *CONFIG_SPI_FLASH_SIZE_OVERRIDE*
- *CONFIG_SPI_FLASH_ROM_IMPL*
- *CONFIG_SPI_FLASH_VERIFY_WRITE*
- *CONFIG_SPI_FLASH_DANGEROUS_WRITE*

CONFIG_SPI_FLASH_VERIFY_WRITE

Verify SPI flash writes

Found in: Component config > SPI Flash driver

If this option is enabled, any time SPI flash is written then the data will be read back and verified. This can catch hardware problems with SPI flash, or flash which was not erased before verification.

CONFIG_SPI_FLASH_LOG_FAILED_WRITE

Log errors if verification fails

Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_VERIFY_WRITE

If this option is enabled, if SPI flash write verification fails then a log error line will be written with the address, expected & actual values. This can be useful when debugging hardware SPI flash problems.

CONFIG_SPI_FLASH_WARN_SETTING_ZERO_TO_ONE

Log warning if writing zero bits to ones

Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_VERIFY_WRITE

If this option is enabled, any SPI flash write which tries to set zero bits in the flash to ones will log a warning. Such writes will not result in the requested data appearing identically in flash once written, as SPI NOR flash can only set bits to one when an entire sector is erased. After erasing, individual bits can only be written from one to zero.

Note that some software (such as SPIFFS) which is aware of SPI NOR flash may write one bits as an optimisation, relying on the data in flash becoming a bitwise AND of the new data and any existing data. Such software will log spurious warnings if this option is enabled.

CONFIG_SPI_FLASH_ENABLE_COUNTERS

Enable operation counters

Found in: Component config > SPI Flash driver

This option enables the following APIs:

- esp_flash_reset_counters
- esp_flash_dump_counters
- esp_flash_get_counters

These APIs may be used to collect performance data for spi_flash APIs and to help understand behaviour of libraries which use SPI flash.

CONFIG_SPI_FLASH_ROM_DRIVER_PATCH

Enable SPI flash ROM driver patched functions

Found in: Component config > SPI Flash driver

Enable this flag to use patched versions of SPI flash ROM driver functions. This option should be enabled, if any one of the following is true: (1) need to write to flash on ESP32-D2WD; (2) main SPI flash is connected to non-default pins; (3) main SPI flash chip is manufactured by ISSI.

CONFIG_SPI_FLASH_ROM_IMPL

Use esp_flash implementation in ROM

Found in: Component config > SPI Flash driver

Enable this flag to use new SPI flash driver functions from ROM instead of ESP-IDF.

If keeping this as ”n” in your project, you will have less free IRAM. But you can use all of our flash features.

If making this as ”y” in your project, you will increase free IRAM. But you may miss out on some flash features and support for new flash chips.

Currently the ROM cannot support the following features:

- SPI_FLASH_AUTO_SUSPEND (C3, S3)

CONFIG_SPI_FLASH_DANGEROUS_WRITE

Writing to dangerous flash regions

Found in: Component config > SPI Flash driver

SPI flash APIs can optionally abort or return a failure code if erasing or writing addresses that fall at the beginning of flash (covering the bootloader and partition table) or that overlap the app partition that contains the running app.

It is not recommended to ever write to these regions from an IDF app, and this check prevents logic errors or corrupted firmware memory from damaging these regions.

Note that this feature *does not* check calls to the esp_rom_xxx SPI flash ROM functions. These functions should not be called directly from IDF applications.

Available options:

- Aborts (CONFIG_SPI_FLASH_DANGEROUS_WRITE_ABORTS)
- Fails (CONFIG_SPI_FLASH_DANGEROUS_WRITE_FAILS)
- Allowed (CONFIG_SPI_FLASH_DANGEROUS_WRITE_ALLOWED)

CONFIG_SPI_FLASH_BYPASS_BLOCK_ERASE

Bypass a block erase and always do sector erase

Found in: Component config > SPI Flash driver

Some flash chips can have very high ”max” erase times, especially for block erase (32KB or 64KB). This option allows to bypass ”block erase” and always do sector erase commands. This will be much slower overall in most cases, but improves latency for other code to run.

CONFIG_SPI_FLASH_YIELD_DURING_ERASE

Enables yield operation during flash erase

Found in: Component config > SPI Flash driver

This allows to yield the CPUs between erase commands. Prevents starvation of other tasks. Please use this configuration together with `SPI\_FLASH\_ERASE\_YIELD\_DURATION\_MS` and `SPI\_FLASH\_ERASE\_YIELD\_TICKS` after carefully checking flash datasheet to avoid a watchdog timeout. For more information, please check *SPI Flash API* reference documentation under section *OS Function*.

CONFIG_SPI_FLASH_ERASE_YIELD_DURATION_MS

Duration of erasing to yield CPUs (ms)

Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_YIELD_DURING_ERASE

If a duration of one erase command is large then it will yield CPUs after finishing a current command.

CONFIG_SPI_FLASH_ERASE_YIELD_TICKS

CPU release time (tick) for an erase operation

Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_YIELD_DURING_ERASE

Defines how many ticks will be before returning to continue a erasing.

CONFIG_SPI_FLASH_WRITE_CHUNK_SIZE

Flash write chunk size

Found in: Component config > SPI Flash driver

Flash write is broken down in terms of multiple (smaller) write operations. This configuration options helps to set individual write chunk size, smaller value here ensures that cache (and non-IRAM resident interrupts) remains disabled for shorter duration.

CONFIG_SPI_FLASH_SIZE_OVERRIDE

Override flash size in bootloader header by ESPTOOLPY_FLASHSIZE

Found in: Component config > SPI Flash driver

SPI Flash driver uses the flash size configured in bootloader header by default. Enable this option to override flash size with latest ESPTOOLPY_FLASHSIZE value from the app header if the size in the bootloader header is incorrect.

CONFIG_SPI_FLASH_CHECK_ERASE_TIMEOUT_DISABLED

Flash timeout checkout disabled

Found in: Component config > SPI Flash driver

This option is helpful if you are using a flash chip whose timeout is quite large or unpredictable.

CONFIG_SPI_FLASH_OVERRIDE_CHIP_DRIVER_LIST

Override default chip driver list

Found in: Component config > SPI Flash driver

This option allows the chip driver list to be customized, instead of using the default list provided by ESP-IDF.

When this option is enabled, the default list is no longer compiled or linked. Instead, the *default_registered_chips* structure must be provided by the user.

See example: custom_chip_driver under examples/storage for more details.

Auto-detect flash chips Contains:

- *CONFIG_SPI_FLASH_SUPPORT_BOYA_CHIP*
- *CONFIG_SPI_FLASH_SUPPORT_GD_CHIP*
- *CONFIG_SPI_FLASH_SUPPORT_ISSI_CHIP*
- *CONFIG_SPI_FLASH_SUPPORT_MXIC_CHIP*
- *CONFIG_SPI_FLASH_SUPPORT_TH_CHIP*
- *CONFIG_SPI_FLASH_SUPPORT_WINBOND_CHIP*

CONFIG_SPI_FLASH_SUPPORT_ISSI_CHIP

ISSI

Found in: Component config > SPI Flash driver > Auto-detect flash chips

Enable this to support auto detection of ISSI chips if chip vendor not directly given by `chip\_drv` member of the chip struct. This adds support for variant chips, however will extend detecting time.

CONFIG_SPI_FLASH_SUPPORT_MXIC_CHIP

MXIC

Found in: Component config > SPI Flash driver > Auto-detect flash chips

Enable this to support auto detection of MXIC chips if chip vendor not directly given by `chip\_drv` member of the chip struct. This adds support for variant chips, however will extend detecting time.

CONFIG_SPI_FLASH_SUPPORT_GD_CHIP

GigaDevice

Found in: Component config > SPI Flash driver > Auto-detect flash chips

Enable this to support auto detection of GD (GigaDevice) chips if chip vendor not directly given by `chip\_drv` member of the chip struct. If you are using Wrover modules, please don't disable this, otherwise your flash may not work in 4-bit mode.

This adds support for variant chips, however will extend detecting time and image size. Note that the default chip driver supports the GD chips with product ID 60H.

CONFIG_SPI_FLASH_SUPPORT_WINBOND_CHIP

Winbond

Found in: Component config > SPI Flash driver > Auto-detect flash chips

Enable this to support auto detection of Winbond chips if chip vendor not directly given by `chip\_drv` member of the chip struct. This adds support for variant chips, however will extend detecting time.

CONFIG_SPI_FLASH_SUPPORT_BOYA_CHIP

BOYA

Found in: Component config > SPI Flash driver > Auto-detect flash chips

Enable this to support auto detection of BOYA chips if chip vendor not directly given by `chip\_drv` member of the chip struct. This adds support for variant chips, however will extend detecting time.

CONFIG_SPI_FLASH_SUPPORT_TH_CHIP

TH

Found in: Component config > SPI Flash driver > Auto-detect flash chips

Enable this to support auto detection of TH chips if chip vendor not directly given by `chip\_drv` member of the chip struct. This adds support for variant chips, however will extend detecting time.

CONFIG_SPI_FLASH_ENABLE_ENCRYPTED_READ_WRITE

Enable encrypted partition read/write operations

Found in: Component config > SPI Flash driver

This option enables flash read/write operations to encrypted partition/s. This option is kept enabled irrespective of state of flash encryption feature. However, in case application is not using flash encryption feature and is in need of some additional memory from IRAM region (~1KB) then this config can be disabled.

SPIFFS Configuration

Contains:

- *Debug Configuration*
- *CONFIG_SPIFFS_USE_MAGIC*
- *CONFIG_SPIFFS_GC_STATS*
- *CONFIG_SPIFFS_PAGE_CHECK*
- *CONFIG_SPIFFS_FOLLOW_SYMLINKS*
- *CONFIG_SPIFFS_MAX_PARTITIONS*
- *CONFIG_SPIFFS_USE_MTIME*
- *CONFIG_SPIFFS_GC_MAX_RUNS*
- *CONFIG_SPIFFS_OBJ_NAME_LEN*
- *CONFIG_SPIFFS_META_LENGTH*

- *SPIFFS Cache Configuration*
- *CONFIG_SPIFFS_PAGE_SIZE*
- *CONFIG_SPIFFS_MTIME_WIDE_64_BITS*

CONFIG_SPIFFS_MAX_PARTITIONS

Maximum Number of Partitions

Found in: Component config > SPIFFS Configuration

Define maximum number of partitions that can be mounted.

Range:
- from 1 to 10

Default value:
- 3

SPIFFS Cache Configuration Contains:

- *CONFIG_SPIFFS_CACHE*

CONFIG_SPIFFS_CACHE

Enable SPIFFS Cache

Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration

Enables/disable memory read caching of nucleus file system operations.

Default value:
- Yes (enabled)

CONFIG_SPIFFS_CACHE_WR

Enable SPIFFS Write Caching

Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration > CONFIG_SPIFFS_CACHE

Enables memory write caching for file descriptors in hydrogen.

Default value:
- Yes (enabled)

CONFIG_SPIFFS_CACHE_STATS

Enable SPIFFS Cache Statistics

Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration > CONFIG_SPIFFS_CACHE

Enable/disable statistics on caching. Debug/test purpose only.

Default value:
- No (disabled)

CONFIG_SPIFFS_PAGE_CHECK

Enable SPIFFS Page Check

Found in: Component config > SPIFFS Configuration

Always check header of each accessed page to ensure consistent state. If enabled it will increase number of reads from flash, especially if cache is disabled.

Default value:
- Yes (enabled)

CONFIG_SPIFFS_GC_MAX_RUNS

Set Maximum GC Runs

Found in: Component config > SPIFFS Configuration

Define maximum number of GC runs to perform to reach desired free pages.

Range:
- from 1 to 10000

Default value:
- 10

CONFIG_SPIFFS_GC_STATS

Enable SPIFFS GC Statistics

Found in: Component config > SPIFFS Configuration

Enable/disable statistics on gc. Debug/test purpose only.

Default value:
- No (disabled)

CONFIG_SPIFFS_PAGE_SIZE

SPIFFS logical page size

Found in: Component config > SPIFFS Configuration

Logical page size of SPIFFS partition, in bytes. Must be multiple of flash page size (which is usually 256 bytes). Larger page sizes reduce overhead when storing large files, and improve filesystem performance when reading large files. Smaller page sizes reduce overhead when storing small (< page size) files.

Range:
- from 256 to 1024

Default value:
- 256

CONFIG_SPIFFS_OBJ_NAME_LEN

Set SPIFFS Maximum Name Length

Found in: Component config > SPIFFS Configuration

Object name maximum length. Note that this length include the zero-termination character, meaning maximum string of characters can at most be SPIFFS_OBJ_NAME_LEN - 1.

SPIFFS_OBJ_NAME_LEN + SPIFFS_META_LENGTH should not exceed SPIFFS_PAGE_SIZE - 64.

Range:
- from 1 to 256

Default value:
- 32

CONFIG_SPIFFS_FOLLOW_SYMLINKS

Enable symbolic links for image creation

Found in: Component config > SPIFFS Configuration

If this option is enabled, symbolic links are taken into account during partition image creation.

Default value:

- No (disabled)

CONFIG_SPIFFS_USE_MAGIC

Enable SPIFFS Filesystem Magic

Found in: Component config > SPIFFS Configuration

Enable this to have an identifiable spiffs filesystem. This will look for a magic in all sectors to determine if this is a valid spiffs system or not at mount time.

Default value:

- Yes (enabled)

CONFIG_SPIFFS_USE_MAGIC_LENGTH

Enable SPIFFS Filesystem Length Magic

Found in: Component config > SPIFFS Configuration > CONFIG_SPIFFS_USE_MAGIC

If this option is enabled, the magic will also be dependent on the length of the filesystem. For example, a filesystem configured and formatted for 4 megabytes will not be accepted for mounting with a configuration defining the filesystem as 2 megabytes.

Default value:

- Yes (enabled)

CONFIG_SPIFFS_META_LENGTH

Size of per-file metadata field

Found in: Component config > SPIFFS Configuration

This option sets the number of extra bytes stored in the file header. These bytes can be used in an application-specific manner. Set this to at least 4 bytes to enable support for saving file modification time.

SPIFFS_OBJ_NAME_LEN + SPIFFS_META_LENGTH should not exceed SPIFFS_PAGE_SIZE - 64.

Default value:

- 4

CONFIG_SPIFFS_USE_MTIME

Save file modification time

Found in: Component config > SPIFFS Configuration

If enabled, then the first 4 bytes of per-file metadata will be used to store file modification time (mtime), accessible through stat/fstat functions. Modification time is updated when the file is opened.

Default value:

- Yes (enabled)

CONFIG_SPIFFS_MTIME_WIDE_64_BITS

The time field occupies 64 bits in the image instead of 32 bits

Found in: Component config > SPIFFS Configuration

If this option is not set, the time field is 32 bits (up to 2106 year), otherwise it is 64 bits and make sure it matches SPIFFS_META_LENGTH. If the chip already has the spiffs image with the time field = 32 bits then this option cannot be applied in this case. Erase it first before using this option. To resolve the Y2K38 problem for the spiffs, use a toolchain with 64-bit time_t support.

Default value:

- No (disabled) if *CONFIG_SPIFFS_META_LENGTH* >= 8

Debug Configuration Contains:

- *CONFIG_SPIFFS_DBG*
- *CONFIG_SPIFFS_API_DBG*
- *CONFIG_SPIFFS_CACHE_DBG*
- *CONFIG_SPIFFS_CHECK_DBG*
- *CONFIG_SPIFFS_TEST_VISUALISATION*
- *CONFIG_SPIFFS_GC_DBG*

CONFIG_SPIFFS_DBG

Enable general SPIFFS debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print general debug messages to the console.

Default value:

- No (disabled)

CONFIG_SPIFFS_API_DBG

Enable SPIFFS API debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print API debug messages to the console.

Default value:

- No (disabled)

CONFIG_SPIFFS_GC_DBG

Enable SPIFFS Garbage Cleaner debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print GC debug messages to the console.

Default value:

- No (disabled)

CONFIG_SPIFFS_CACHE_DBG

Enable SPIFFS Cache debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print cache debug messages to the console.

Default value:

- No (disabled)

CONFIG_SPIFFS_CHECK_DBG

Enable SPIFFS Filesystem Check debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print Filesystem Check debug messages to the console.

Default value:

- No (disabled)

CONFIG_SPIFFS_TEST_VISUALISATION

Enable SPIFFS Filesystem Visualization

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enable this option to enable SPIFFS_vis function in the API.

Default value:

- No (disabled)

TCP Transport

Contains:

- *Websocket*

Websocket

Contains:

- *CONFIG_WS_TRANSPORT*

CONFIG_WS_TRANSPORT

Enable Websocket Transport

Found in: Component config > TCP Transport > Websocket

Enable support for creating websocket transport.

Default value:

- Yes (enabled)

CONFIG_WS_BUFFER_SIZE

Websocket transport buffer size

Found in: Component config > TCP Transport > Websocket > CONFIG_WS_TRANSPORT

Size of the buffer used for constructing the HTTP Upgrade request during connect

Default value:

- 1024

CONFIG_WS_DYNAMIC_BUFFER

Using dynamic websocket transport buffer

Found in: Component config > TCP Transport > Websocket > CONFIG_WS_TRANSPORT

If enable this option, websocket transport buffer will be freed after connection succeed to save more heap.

Default value:

- No (disabled)

Ultra Low Power (ULP) Co-processor Contains:

- *CONFIG_ULP_ROM_PRINT_ENABLE*
- *CONFIG_ULP_COPROC_ENABLED*
- *ULP Debugging Options*
- *ULP RISC-V Settings*

CONFIG_ULP_COPROC_ENABLED

Enable Ultra Low Power (ULP) Co-processor

Found in: Component config > Ultra Low Power (ULP) Co-processor

Enable this feature if you plan to use the ULP Co-processor. Once this option is enabled, further ULP co-processor configuration will appear in the menu.

Default value:

- No (disabled)

CONFIG_ULP_COPROC_TYPE

ULP Co-processor type

Found in: Component config > Ultra Low Power (ULP) Co-processor > CONFIG_ULP_COPROC_ENABLED

Choose the ULP Coprocessor type: ULP FSM (Finite State Machine) or ULP RISC-V.

Available options:

- ULP FSM (Finite State Machine) (CONFIG_ULP_COPROC_TYPE_FSM)
- ULP RISC-V (CONFIG_ULP_COPROC_TYPE_RISCV)
- LP core RISC-V (CONFIG_ULP_COPROC_TYPE_LP_CORE)

CONFIG_ULP_COPROC_RESERVE_MEM

RTC slow memory reserved for coprocessor

Found in: Component config > Ultra Low Power (ULP) Co-processor > CONFIG_ULP_COPROC_ENABLED

Bytes of memory to reserve for ULP Co-processor firmware & data. Data is reserved at the beginning of RTC slow memory.

Range:

- from 32 to 31088 if *CONFIG_ULP_COPROC_ENABLED*

Default value:

- 4096 if *CONFIG_ULP_COPROC_ENABLED*

ULP RISC-V Settings Contains:

- *CONFIG_ULP_RISCV_UART_BAUDRATE*
- *CONFIG_ULP_RISCV_INTERRUPT_ENABLE*
- *CONFIG_ULP_RISCV_I2C_RW_TIMEOUT*

CONFIG_ULP_RISCV_INTERRUPT_ENABLE

Enable ULP RISC-V interrupts

Found in: Component config > Ultra Low Power (ULP) Co-processor > ULP RISC-V Settings

Turn on this setting to enabled interrupts on the ULP RISC-V core.

Default value:

- No (disabled) if *CONFIG_ULP_COPROC_TYPE_RISCV*

CONFIG_ULP_RISCV_UART_BAUDRATE

Baudrate used by the bitbanged ULP RISC-V UART driver

Found in: Component config > Ultra Low Power (ULP) Co-processor > ULP RISC-V Settings

The accuracy of the bitbanged UART driver is limited, it is not recommend to increase the value above 19200.

Default value:

- 9600 if *CONFIG_ULP_COPROC_TYPE_RISCV*

CONFIG_ULP_RISCV_I2C_RW_TIMEOUT

Set timeout for ULP RISC-V I2C transaction timeout in ticks.

Found in: Component config > Ultra Low Power (ULP) Co-processor > ULP RISC-V Settings

Set the ULP RISC-V I2C read/write timeout. Set this value to -1 if the ULP RISC-V I2C read and write APIs should wait forever. Please note that the tick rate of the ULP co-processor would be different than the OS tick rate of the main core and therefore can have different timeout value depending on which core the API is invoked on.

Range:

- from -1 to 4294967295 if *CONFIG_ULP_COPROC_TYPE_RISCV*

Default value:

- 500 if *CONFIG_ULP_COPROC_TYPE_RISCV*

CONFIG_ULP_ROM_PRINT_ENABLE

Enable print utilities from LP ROM

Found in: Component config > Ultra Low Power (ULP) Co-processor

Set this option to enable printf functionality from LP ROM. This option can help reduce the LP core binary size by not linking printf functionality from RAM code. Note: For LP ROM prints to work properly, make sure that the LP core boots from the LP ROM.

Default value:

- Yes (enabled) if *CONFIG_ULP_COPROC_TYPE_LP_CORE*

ULP Debugging Options

Contains:

- *CONFIG_ULP_PANIC_OUTPUT_ENABLE*
- *CONFIG_ULP_HP_UART_CONSOLE_PRINT*

CONFIG_ULP_PANIC_OUTPUT_ENABLE

Enable panic handler which outputs over LP UART

Found in: Component config > Ultra Low Power (ULP) Co-processor > ULP Debugging Options

Set this option to enable panic handler functionality. If this option is enabled then the LP Core will output a panic dump over LP UART, similar to what the main core does. Output depends on LP UART

already being initialized and configured. Disabling this option will reduce the LP core binary size by not linking in panic handler functionality.

Default value:

- Yes (enabled) if *CONFIG_ULP_COPROC_TYPE_LP_CORE*

CONFIG_ULP_HP_UART_CONSOLE_PRINT

Route lp_core_printf to the console HP-UART

Found in: Component config > Ultra Low Power (ULP) Co-processor > ULP Debugging Options

Set this option to route lp_core_printf to the console HP-UART. This allows you to easily view print outputs from the LP core, without having to connect to the LP-UART. This option comes with the following limitations:

1. There is no mutual exclusion between the HP-Core and the LP-Core accessing the HP-UART, which means that if both cores are logging heavily the output strings might get mangled together. 2. The HP-UART can only work while the HP-Core is running, which means that if the HP-Core is in deep sleep, the LP-Core will not be able to print to the console HP-UART.

Due to these limitations it is only recommended to use this option for easy debugging. For more serious use-cases you should use the LP-UART.

Unity unit testing library

Contains:

- *CONFIG_UNITY_ENABLE_COLOR*
- *CONFIG_UNITY_ENABLE_IDF_TEST_RUNNER*
- *CONFIG_UNITY_ENABLE_FIXTURE*
- *CONFIG_UNITY_ENABLE_BACKTRACE_ON_FAIL*
- *CONFIG_UNITY_ENABLE_64BIT*
- *CONFIG_UNITY_ENABLE_DOUBLE*
- *CONFIG_UNITY_ENABLE_FLOAT*

CONFIG_UNITY_ENABLE_FLOAT

Support for float type

Found in: Component config > Unity unit testing library

If not set, assertions on float arguments will not be available.

Default value:

- Yes (enabled)

CONFIG_UNITY_ENABLE_DOUBLE

Support for double type

Found in: Component config > Unity unit testing library

If not set, assertions on double arguments will not be available.

Default value:

- Yes (enabled)

CONFIG_UNITY_ENABLE_64BIT

Support for 64-bit integer types

Found in: Component config > Unity unit testing library

If not set, assertions on 64-bit integer types will always fail. If this feature is enabled, take care not to pass pointers (which are 32 bit) to UNITY_ASSERT_EQUAL, as that will cause pointer-to-int-cast warnings.

Default value:

- No (disabled)

CONFIG_UNITY_ENABLE_COLOR

Colorize test output

Found in: Component config > Unity unit testing library

If set, Unity will colorize test results using console escape sequences.

Default value:

- No (disabled)

CONFIG_UNITY_ENABLE_IDF_TEST_RUNNER

Include ESP-IDF test registration/running helpers

Found in: Component config > Unity unit testing library

If set, then the following features will be available:

- TEST_CASE macro which performs automatic registration of test functions
- Functions to run registered test functions: unity_run_all_tests, unity_run_tests_with_filter, unity_run_single_test_by_name.
- Interactive menu which lists test cases and allows choosing the tests to be run, available via unity_run_menu function.

Disable if a different test registration mechanism is used.

Default value:

- Yes (enabled)

CONFIG_UNITY_ENABLE_FIXTURE

Include Unity test fixture

Found in: Component config > Unity unit testing library

If set, unity_fixture.h header file and associated source files are part of the build. These provide an optional set of macros and functions to implement test groups.

Default value:

- No (disabled)

CONFIG_UNITY_ENABLE_BACKTRACE_ON_FAIL

Print a backtrace when a unit test fails

Found in: Component config > Unity unit testing library

If set, the unity framework will print the backtrace information before jumping back to the test menu. The jumping is usually occurs in assert functions such as TEST_ASSERT, TEST_FAIL etc.

Default value:

- No (disabled)

USB-OTG Contains:

- *CONFIG_USB_HOST_ENABLE_ENUM_FILTER_CALLBACK*
- *CONFIG_USB_HOST_HW_BUFFER_BIAS*
- *Hub Driver Configuration*
- *CONFIG_USB_HOST_CONTROL_TRANSFER_MAX_SIZE*

CONFIG_USB_HOST_CONTROL_TRANSFER_MAX_SIZE

Largest size (in bytes) of transfers to/from default endpoints

Found in: Component config > USB-OTG

Each USB device attached is allocated a dedicated buffer for its OUT/IN transfers to/from the device's control endpoint. The maximum size of that buffer is determined by this option. The limited size of the transfer buffer have the following implications: - The maximum length of control transfers is limited - Device's with configuration descriptors larger than this limit cannot be supported

Default value:

- 256

CONFIG_USB_HOST_HW_BUFFER_BIAS

Hardware FIFO size biasing

Found in: Component config > USB-OTG

The underlying hardware has size adjustable FIFOs to cache USB packets on reception (IN) or for transmission (OUT). The size of these FIFOs will affect the largest MPS (maximum packet size) and the maximum number of packets that can be cached at any one time. The hardware contains the following FIFOS: RX (for all IN packets), Non-periodic TX (for Bulk and Control OUT packets), and Periodic TX (for Interrupt and Isochronous OUT packets). This configuration option allows biasing the FIFO sizes towards a particular use case, which may be necessary for devices that have endpoints with large MPS. The MPS limits for each biasing are listed below:

Balanced: - IN (all transfer types), 408 bytes - OUT non-periodic (Bulk/Control), 192 bytes (i.e., 3 x 64 byte packets) - OUT periodic (Interrupt/Isochronous), 192 bytes

Bias IN: - IN (all transfer types), 600 bytes - OUT non-periodic (Bulk/Control), 64 bytes (i.e., 1 x 64 byte packets) - OUT periodic (Interrupt/Isochronous), 128 bytes

Bias Periodic OUT: - IN (all transfer types), 128 bytes - OUT non-periodic (Bulk/Control), 64 bytes (i.e., 1 x 64 byte packets) - OUT periodic (Interrupt/Isochronous), 600 bytes

Available options:

- Balanced (CONFIG_USB_HOST_HW_BUFFER_BIAS_BALANCED)
- Bias IN (CONFIG_USB_HOST_HW_BUFFER_BIAS_IN)
- Periodic OUT (CONFIG_USB_HOST_HW_BUFFER_BIAS_PERIODIC_OUT)

Hub Driver Configuration Contains:

- *Root Port configuration*
- *CONFIG_USB_HOST_HUBS_SUPPORTED*

Root Port configuration Contains:

- *CONFIG_USB_HOST_DEBOUNCE_DELAY_MS*
- *CONFIG_USB_HOST_RESET_HOLD_MS*
- *CONFIG_USB_HOST_RESET_RECOVERY_MS*
- *CONFIG_USB_HOST_SET_ADDR_RECOVERY_MS*

CONFIG_USB_HOST_DEBOUNCE_DELAY_MS

Debounce delay in ms

Found in: Component config > USB-OTG > Hub Driver Configuration > Root Port configuration

On connection of a USB device, the USB 2.0 specification requires a ”debounce interval with a minimum duration of 100ms” to allow the connection to stabilize (see USB 2.0 chapter 7.1.7.3 for more details). During the debounce interval, no new connection/disconnection events are registered.

The default value is set to 250 ms to be safe.

Default value:

- 250

CONFIG_USB_HOST_RESET_HOLD_MS

Reset hold in ms

Found in: Component config > USB-OTG > Hub Driver Configuration > Root Port configuration

The reset signaling can be generated on any Hub or Host Controller port by request from the USB System Software. The USB 2.0 specification requires that ”the reset signaling must be driven for a minimum of 10ms” (see USB 2.0 chapter 7.1.7.5 for more details). After the reset, the hub port will transition to the Enabled state (refer to Section 11.5).

The default value is set to 30 ms to be safe.

Default value:

- 30

CONFIG_USB_HOST_RESET_RECOVERY_MS

Reset recovery delay in ms

Found in: Component config > USB-OTG > Hub Driver Configuration > Root Port configuration

After a port stops driving the reset signal, the USB 2.0 specification requires that the ”USB System Software guarantees a minimum of 10 ms for reset recovery” before the attached device is expected to respond to data transfers (see USB 2.0 chapter 7.1.7.3 for more details). The device may ignore any data transfers during the recovery interval.

The default value is set to 30 ms to be safe.

Default value:

- 30

CONFIG_USB_HOST_SET_ADDR_RECOVERY_MS

SetAddress() recovery time in ms

Found in: Component config > USB-OTG > Hub Driver Configuration > Root Port configuration

”After successful completion of the Status stage, the device is allowed a SetAddress() recovery interval of 2 ms. At the end of this interval, the device must be able to accept Setup packets addressed to the new address. Also, at the end of the recovery interval, the device must not respond to tokens sent to the old address (unless, of course, the old and new address is the same).” See USB 2.0 chapter 9.2.6.3 for more details.

The default value is set to 10 ms to be safe.

Default value:

- 10

CONFIG_USB_HOST_HUBS_SUPPORTED

Support Hubs

Found in: Component config > USB-OTG > Hub Driver Configuration

Enables support of external Hubs.

Default value:

- No (disabled)

CONFIG_USB_HOST_HUB_MULTI_LEVEL

Support multiple Hubs

Found in: Component config > USB-OTG > Hub Driver Configuration > CONFIG_USB_HOST_HUBS_SUPPORTED

Enables support for connecting multiple Hubs simultaneously.

Default value:

- Yes (enabled) if *CONFIG_USB_HOST_HUBS_SUPPORTED*

Downstream Port configuration Contains:

- *CONFIG_USB_HOST_EXT_PORT_CUSTOM_POWER_ON_DELAY_ENABLE*
- *CONFIG_USB_HOST_EXT_PORT_RESET_RECOVERY_DELAY_MS*
- *CONFIG_USB_HOST_EXT_PORT_SUPPORT_LS*

CONFIG_USB_HOST_EXT_PORT_SUPPORT_LS

Support LS

Found in: Component config > USB-OTG > Hub Driver Configuration > CONFIG_USB_HOST_HUBS_SUPPORTED > Downstream Port configuration

Enables support of Low-speed devices, connected through the external Hub.

Default value:

- No (disabled) if *CONFIG_IDF_EXPERIMENTAL_FEATURES* && *CONFIG_USB_HOST_HUBS_SUPPORTED*

CONFIG_USB_HOST_EXT_PORT_RESET_RECOVERY_DELAY_MS

Reset recovery delay in ms

Found in: Component config > USB-OTG > Hub Driver Configuration > CONFIG_USB_HOST_HUBS_SUPPORTED > Downstream Port configuration

After a port stops driving the reset signal, the USB 2.0 specification requires that the ”USB System Software guarantees a minimum of 10 ms for reset recovery” before the attached device is expected to respond to data transfers (see USB 2.0 chapter 7.1.7.3 for more details). The device may ignore any data transfers during the recovery interval.

The default value is set to 30 ms to be safe.

Default value:

- 30 if *CONFIG_USB_HOST_HUBS_SUPPORTED*

CONFIG_USB_HOST_EXT_PORT_CUSTOM_POWER_ON_DELAY_ENABLE

Custom bPwrOn2PwrGood value

Found in: Component config > USB-OTG > Hub Driver Configuration > CONFIG_USB_HOST_HUBS_SUPPORTED > Downstream Port configuration

Enables the possibility to configure custom time for the power-on sequence on a port until power is good on that port.

When enabled, applies the custom PwrOn2PwrGood delay. When disabled, applies the PwrOn2PwrGood value from the Hub Descriptor.

Default value:

- No (disabled) if *CONFIG_USB_HOST_HUBS_SUPPORTED*

CONFIG_USB_HOST_EXT_PORT_CUSTOM_POWER_ON_DELAY_MS

PwrOn2PwrGood delay in ms

Found in: Component config > USB-OTG > Hub Driver Configuration > CONFIG_USB_HOST_HUBS_SUPPORTED > Downstream Port configuration > CONFIG_USB_HOST_EXT_PORT_CUSTOM_POWER_ON_DELAY_ENABLE

Custom value of delay from the time the power-on sequence begins on a port until power is good on that port. Value 0 is used for a hub with no power switches.

The default value is 100 ms.

Range:

- from 0 to 5000 if *CONFIG_USB_HOST_EXT_PORT_CUSTOM_POWER_ON_DELAY_ENABLE* && *CONFIG_USB_HOST_HUBS_SUPPORTED*

Default value:

- 100 if *CONFIG_USB_HOST_EXT_PORT_CUSTOM_POWER_ON_DELAY_ENABLE* && *CONFIG_USB_HOST_HUBS_SUPPORTED*

CONFIG_USB_HOST_ENABLE_ENUM_FILTER_CALLBACK

Enable enumeration filter callback

Found in: Component config > USB-OTG

The enumeration filter callback is called before enumeration of each newly attached device. This callback allows users to control whether a device should be enumerated, and what configuration number to use when enumerating a device.

If enabled, the enumeration filter callback can be set via 'usb_host_config_t' when calling 'usb_host_install()'.

Default value:

- No (disabled)

Virtual file system Contains:

- *CONFIG_VFS_SUPPORT_IO*

CONFIG_VFS_SUPPORT_IO

Provide basic I/O functions

Found in: Component config > Virtual file system

If enabled, the following functions are provided by the VFS component.

open, close, read, write, pread, pwrite, lseek, fstat, fsync, ioctl, fcntl

Filesystem drivers can then be registered to handle these functions for specific paths.

Disabling this option can save memory when the support for these functions is not required.

Note that the following functions can still be used with socket file descriptors when this option is disabled:

close, read, write, ioctl, fcntl.

Default value:

- Yes (enabled)

CONFIG_VFS_SUPPORT_DIR

Provide directory related functions

Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO

If enabled, the following functions are provided by the VFS component.

stat, link, unlink, rename, utime, access, truncate, rmdir, mkdir, opendir, closedir, readdir, readdir_r, seekdir, telldir, rewinddir

Filesystem drivers can then be registered to handle these functions for specific paths.

Disabling this option can save memory when the support for these functions is not required.

Default value:

- Yes (enabled)

CONFIG_VFS_SUPPORT_SELECT

Provide select function

Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO

If enabled, select function is provided by the VFS component, and can be used on peripheral file descriptors (such as UART) and sockets at the same time.

If disabled, the default select implementation will be provided by LWIP for sockets only.

Disabling this option can reduce code size if support for "select" on UART file descriptors is not required.

CONFIG_VFS_SUPPRESS_SELECT_DEBUG_OUTPUT

Suppress select() related debug outputs

Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO > CONFIG_VFS_SUPPORT_SELECT

Select() related functions might produce an inconveniently lot of debug outputs when one sets the default log level to DEBUG or higher. It is possible to suppress these debug outputs by enabling this option.

Default value:

- Yes (enabled)

CONFIG_VFS_SELECT_IN_RAM

Make VFS driver select() callbacks IRAM-safe

Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO > CONFIG_VFS_SUPPORT_SELECT

If enabled, VFS driver select() callback function will be placed in IRAM.

Default value:

- No (disabled)

CONFIG_VFS_SUPPORT_TERMIOS

Provide termios.h functions

Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO

Disabling this option can save memory when the support for termios.h is not required.

Default value:

- Yes (enabled)

CONFIG_VFS_MAX_COUNT

Maximum Number of Virtual Filesystems

Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO

Define maximum number of virtual filesystems that can be registered.

Range:

- from 1 to 20

Default value:

- 8

Host File System I/O (Semihosting) Contains:

- *CONFIG_VFS_SEMIHOSTFS_MAX_MOUNT_POINTS*

CONFIG_VFS_SEMIHOSTFS_MAX_MOUNT_POINTS

Host FS: Maximum number of the host filesystem mount points

Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO > Host File System I/O (Semihosting)

Define maximum number of host filesystem mount points.

Default value:

- 1

Wear Levelling Contains:

- *CONFIG_WL_SECTOR_MODE*
- *CONFIG_WL_SECTOR_SIZE*

CONFIG_WL_SECTOR_SIZE

Wear Levelling library sector size

Found in: Component config > Wear Levelling

Sector size used by wear levelling library. You can set default sector size or size that will fit to the flash device sector size.

With sector size set to 4096 bytes, wear levelling library is more efficient. However if FAT filesystem is used on top of wear levelling library, it will need more temporary storage: 4096 bytes for each mounted filesystem and 4096 bytes for each opened file.

With sector size set to 512 bytes, wear levelling library will perform more operations with flash memory, but less RAM will be used by FAT filesystem library (512 bytes for the filesystem and 512 bytes for each file opened).

Available options:

- 512 (CONFIG_WL_SECTOR_SIZE_512)
- 4096 (CONFIG_WL_SECTOR_SIZE_4096)

CONFIG_WL_SECTOR_MODE

Sector store mode

Found in: Component config > Wear Levelling

Specify the mode to store data into flash:

- In Performance mode a data will be stored to the RAM and then stored back to the flash. Compared to the Safety mode, this operation is faster, but if power will be lost when erase sector operation is in progress, then the data from complete flash device sector will be lost.
- In Safety mode data from complete flash device sector will be read from flash, modified, and then stored back to flash. Compared to the Performance mode, this operation is slower, but if power is lost during erase sector operation, then the data from full flash device sector will not be lost.

Available options:

- Performance (CONFIG_WL_SECTOR_MODE_PERF)
- Safety (CONFIG_WL_SECTOR_MODE_SAFE)

Wi-Fi Provisioning Manager Contains:

- *CONFIG_WIFI_PROV_BLE_NOTIFY*
- *CONFIG_WIFI_PROV_BLE_BONDING*
- *CONFIG_WIFI_PROV_BLE_SEC_CONN*
- *CONFIG_WIFI_PROV_BLE_FORCE_ENCRYPTION*
- *CONFIG_WIFI_PROV_KEEP_BLE_ON_AFTER_PROV*
- *CONFIG_WIFI_PROV_SCAN_MAX_ENTRIES*
- *CONFIG_WIFI_PROV_AUTOSTOP_TIMEOUT*
- *CONFIG_WIFI_PROV_STA_SCAN_METHOD*

CONFIG_WIFI_PROV_SCAN_MAX_ENTRIES

Max Wi-Fi Scan Result Entries

Found in: Component config > Wi-Fi Provisioning Manager

This sets the maximum number of entries of Wi-Fi scan results that will be kept by the provisioning manager

Range:
- from 1 to 255

Default value:
- 16

CONFIG_WIFI_PROV_AUTOSTOP_TIMEOUT

Provisioning auto-stop timeout

Found in: Component config > Wi-Fi Provisioning Manager

Time (in seconds) after which the Wi-Fi provisioning manager will auto-stop after connecting to a Wi-Fi network successfully.

Range:
- from 5 to 600

Default value:
- 30

CONFIG_WIFI_PROV_BLE_BONDING

Enable BLE bonding

Found in: Component config > Wi-Fi Provisioning Manager

This option is applicable only when provisioning transport is BLE.

CONFIG_WIFI_PROV_BLE_SEC_CONN

Enable BLE Secure connection flag

Found in: Component config > Wi-Fi Provisioning Manager

Used to enable Secure connection support when provisioning transport is BLE.

Default value:

- Yes (enabled) if *CONFIG_BT_NIMBLE_ENABLED*

CONFIG_WIFI_PROV_BLE_FORCE_ENCRYPTION

Force Link Encryption during characteristic Read / Write

Found in: Component config > Wi-Fi Provisioning Manager

Used to enforce link encryption when attempting to read / write characteristic

CONFIG_WIFI_PROV_BLE_NOTIFY

Add support for Notification for provisioning BLE descriptors

Found in: Component config > Wi-Fi Provisioning Manager

Used to enable support Notification in BLE descriptors of prov* characteristics

CONFIG_WIFI_PROV_KEEP_BLE_ON_AFTER_PROV

Keep BT on after provisioning is done

Found in: Component config > Wi-Fi Provisioning Manager

CONFIG_WIFI_PROV_DISCONNECT_AFTER_PROV

Terminate connection after provisioning is done

Found in: Component config > Wi-Fi Provisioning Manager > CONFIG_WIFI_PROV_KEEP_BLE_ON_AFTER_PROV

Default value:

- Yes (enabled) if *CONFIG_WIFI_PROV_KEEP_BLE_ON_AFTER_PROV*

CONFIG_WIFI_PROV_STA_SCAN_METHOD

Wifi Provisioning Scan Method

Found in: Component config > Wi-Fi Provisioning Manager

Available options:

- All Channel Scan (CONFIG_WIFI_PROV_STA_ALL_CHANNEL_SCAN)
  Scan will end after scanning the entire channel. This option is useful in Mesh WiFi Systems.

- Fast Scan (CONFIG_WIFI_PROV_STA_FAST_SCAN)
  Scan will end after an AP matching with the SSID has been detected.

CONFIG_IDF_EXPERIMENTAL_FEATURES

Make experimental features visible

Found in:

By enabling this option, ESP-IDF experimental feature options will be visible.

Note you should still enable a certain experimental feature option to use it, and you should read the corresponding risk warning and known issue list carefully.

Current experimental feature list:

- CONFIG_ESPTOOLPY_FLASHFREQ_120M && CONFIG_ESPTOOLPY_FLASH_SAMPLE_MODE_DTR
- CONFIG_SPIRAM_SPEED_120M && CONFIG_SPIRAM_MODE_OCT
- CONFIG_BOOTLOADER_CACHE_32BIT_ADDR_QUAD_FLASH
- CONFIG_ESP_WIFI_EAP_TLS1_3
- CONFIG_ESP_WIFI_ENABLE_ROAMING_APP
- CONFIG_USB_HOST_EXT_PORT_SUPPORT_LS
- CONFIG_USB_HOST_EXT_PORT_RESET_ATTEMPTS

Default value:

- No (disabled)

Deprecated options and their replacements

- CONFIG_A2DP_ENABLE (*CONFIG_BT_A2DP_ENABLE*)
- **CONFIG_A2D_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_A2D_TRACE_LEVEL*)**
  - CONFIG_A2D_TRACE_LEVEL_NONE
  - CONFIG_A2D_TRACE_LEVEL_ERROR
  - CONFIG_A2D_TRACE_LEVEL_WARNING
  - CONFIG_A2D_TRACE_LEVEL_API
  - CONFIG_A2D_TRACE_LEVEL_EVENT
  - CONFIG_A2D_TRACE_LEVEL_DEBUG
  - CONFIG_A2D_TRACE_LEVEL_VERBOSE
- CONFIG_ADC2_DISABLE_DAC (*CONFIG_ADC_DISABLE_DAC*)
- **CONFIG_APPL_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_APPL_TRACE_LEVEL*)**
  - CONFIG_APPL_TRACE_LEVEL_NONE
  - CONFIG_APPL_TRACE_LEVEL_ERROR
  - CONFIG_APPL_TRACE_LEVEL_WARNING
  - CONFIG_APPL_TRACE_LEVEL_API
  - CONFIG_APPL_TRACE_LEVEL_EVENT
  - CONFIG_APPL_TRACE_LEVEL_DEBUG
  - CONFIG_APPL_TRACE_LEVEL_VERBOSE
- CONFIG_APP_ANTI_ROLLBACK (*CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*)
- CONFIG_APP_ROLLBACK_ENABLE (*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*)
- CONFIG_APP_SECURE_VERSION (*CONFIG_BOOTLOADER_APP_SECURE_VERSION*)
- CONFIG_APP_SECURE_VERSION_SIZE_EFUSE_FIELD (*CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIEL*
- **CONFIG_AVCT_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_AVCT_TRACE_LEVEL*)**
  - CONFIG_AVCT_TRACE_LEVEL_NONE
  - CONFIG_AVCT_TRACE_LEVEL_ERROR
  - CONFIG_AVCT_TRACE_LEVEL_WARNING
  - CONFIG_AVCT_TRACE_LEVEL_API
  - CONFIG_AVCT_TRACE_LEVEL_EVENT
  - CONFIG_AVCT_TRACE_LEVEL_DEBUG
  - CONFIG_AVCT_TRACE_LEVEL_VERBOSE
- **CONFIG_AVDT_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_AVDT_TRACE_LEVEL*)**

  - CONFIG_AVDT_TRACE_LEVEL_NONE
  - CONFIG_AVDT_TRACE_LEVEL_ERROR
  - CONFIG_AVDT_TRACE_LEVEL_WARNING
  - CONFIG_AVDT_TRACE_LEVEL_API
  - CONFIG_AVDT_TRACE_LEVEL_EVENT
  - CONFIG_AVDT_TRACE_LEVEL_DEBUG
  - CONFIG_AVDT_TRACE_LEVEL_VERBOSE
- **CONFIG_AVRC_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_AVRC_TRACE_LEVEL***)
  - CONFIG_AVRC_TRACE_LEVEL_NONE
  - CONFIG_AVRC_TRACE_LEVEL_ERROR
  - CONFIG_AVRC_TRACE_LEVEL_WARNING
  - CONFIG_AVRC_TRACE_LEVEL_API
  - CONFIG_AVRC_TRACE_LEVEL_EVENT
  - CONFIG_AVRC_TRACE_LEVEL_DEBUG
  - CONFIG_AVRC_TRACE_LEVEL_VERBOSE
- CONFIG_BLE_ACTIVE_SCAN_REPORT_ADV_SCAN_RSP_INDIVIDUALLY (*CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN*)
- CONFIG_BLE_ESTABLISH_LINK_CONNECTION_TIMEOUT (*CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT*)
- CONFIG_BLE_HOST_QUEUE_CONGESTION_CHECK (*CONFIG_BT_BLE_HOST_QUEUE_CONG_CHECK*)
- CONFIG_BLE_MESH_GATT_PROXY (*CONFIG_BLE_MESH_GATT_PROXY_SERVER*)
- CONFIG_BLE_SMP_ENABLE (*CONFIG_BT_BLE_SMP_ENABLE*)
- CONFIG_BLUEDROID_MEM_DEBUG (*CONFIG_BT_BLUEDROID_MEM_DEBUG*)
- **CONFIG_BLUEDROID_PINNED_TO_CORE_CHOICE** (***CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE***)
  - CONFIG_BLUEDROID_PINNED_TO_CORE_0
  - CONFIG_BLUEDROID_PINNED_TO_CORE_1
- **CONFIG_BLUFI_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_BLUFI_TRACE_LEVEL***)
  - CONFIG_BLUFI_TRACE_LEVEL_NONE
  - CONFIG_BLUFI_TRACE_LEVEL_ERROR
  - CONFIG_BLUFI_TRACE_LEVEL_WARNING
  - CONFIG_BLUFI_TRACE_LEVEL_API
  - CONFIG_BLUFI_TRACE_LEVEL_EVENT
  - CONFIG_BLUFI_TRACE_LEVEL_DEBUG
  - CONFIG_BLUFI_TRACE_LEVEL_VERBOSE
- CONFIG_BNEP_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_BNEP_TRACE_LEVEL*)
- CONFIG_BROWNOUT_DET (*CONFIG_ESP_BROWNOUT_DET*)
- **CONFIG_BROWNOUT_DET_LVL_SEL** (***CONFIG_ESP_BROWNOUT_DET_LVL_SEL***)
  - CONFIG_BROWNOUT_DET_LVL_SEL_7
  - CONFIG_BROWNOUT_DET_LVL_SEL_6
  - CONFIG_BROWNOUT_DET_LVL_SEL_5
- **CONFIG_BTC_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_BTC_TRACE_LEVEL***)
  - CONFIG_BTC_TRACE_LEVEL_NONE
  - CONFIG_BTC_TRACE_LEVEL_ERROR
  - CONFIG_BTC_TRACE_LEVEL_WARNING
  - CONFIG_BTC_TRACE_LEVEL_API
  - CONFIG_BTC_TRACE_LEVEL_EVENT
  - CONFIG_BTC_TRACE_LEVEL_DEBUG
  - CONFIG_BTC_TRACE_LEVEL_VERBOSE
- CONFIG_BTC_TASK_STACK_SIZE (*CONFIG_BT_BTC_TASK_STACK_SIZE*)
- **CONFIG_BTH_LOG_SDP_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_SDP_TRACE_LEVEL***)
  - CONFIG_SDP_TRACE_LEVEL_NONE
  - CONFIG_SDP_TRACE_LEVEL_ERROR
  - CONFIG_SDP_TRACE_LEVEL_WARNING
  - CONFIG_SDP_TRACE_LEVEL_API
  - CONFIG_SDP_TRACE_LEVEL_EVENT
  - CONFIG_SDP_TRACE_LEVEL_DEBUG
  - CONFIG_SDP_TRACE_LEVEL_VERBOSE

- **CONFIG_BTIF_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_BTIF_TRACE_LEVEL*)**
  - CONFIG_BTIF_TRACE_LEVEL_NONE
  - CONFIG_BTIF_TRACE_LEVEL_ERROR
  - CONFIG_BTIF_TRACE_LEVEL_WARNING
  - CONFIG_BTIF_TRACE_LEVEL_API
  - CONFIG_BTIF_TRACE_LEVEL_EVENT
  - CONFIG_BTIF_TRACE_LEVEL_DEBUG
  - CONFIG_BTIF_TRACE_LEVEL_VERBOSE
- **CONFIG_BTM_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_BTM_TRACE_LEVEL*)**
  - CONFIG_BTM_TRACE_LEVEL_NONE
  - CONFIG_BTM_TRACE_LEVEL_ERROR
  - CONFIG_BTM_TRACE_LEVEL_WARNING
  - CONFIG_BTM_TRACE_LEVEL_API
  - CONFIG_BTM_TRACE_LEVEL_EVENT
  - CONFIG_BTM_TRACE_LEVEL_DEBUG
  - CONFIG_BTM_TRACE_LEVEL_VERBOSE
- CONFIG_BTU_TASK_STACK_SIZE (*CONFIG_BT_BTU_TASK_STACK_SIZE*)
- CONFIG_BT_NIMBLE_ACL_BUF_COUNT (*CONFIG_BT_NIMBLE_TRANSPORT_ACL_FROM_LL_COUNT*)
- CONFIG_BT_NIMBLE_ACL_BUF_SIZE (*CONFIG_BT_NIMBLE_TRANSPORT_ACL_SIZE*)
- CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE (*CONFIG_BT_NIMBLE_TRANSPORT_EVT_SIZE*)
- CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT (*CONFIG_BT_NIMBLE_TRANSPORT_EVT_COUNT*)
- CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT (*CONFIG_BT_NIMBLE_TRANSPORT_EVT_DISCARD_COUNT*)
- CONFIG_BT_NIMBLE_MSYS1_BLOCK_COUNT (*CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT*)
- CONFIG_BT_NIMBLE_SM_SC_LVL (*CONFIG_BT_NIMBLE_SM_LVL*)
- CONFIG_BT_NIMBLE_TASK_STACK_SIZE (*CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE*)
- CONFIG_CLASSIC_BT_ENABLED (*CONFIG_BT_CLASSIC_ENABLED*)
- **CONFIG_CONSOLE_UART (*CONFIG_ESP_CONSOLE_UART*)**
  - CONFIG_CONSOLE_UART_DEFAULT
  - CONFIG_CONSOLE_UART_CUSTOM
  - CONFIG_CONSOLE_UART_NONE, CONFIG_ESP_CONSOLE_UART_NONE
- CONFIG_CONSOLE_UART_BAUDRATE (*CONFIG_ESP_CONSOLE_UART_BAUDRATE*)
- **CONFIG_CONSOLE_UART_NUM (*CONFIG_ESP_CONSOLE_UART_NUM*)**
  - CONFIG_CONSOLE_UART_CUSTOM_NUM_0
  - CONFIG_CONSOLE_UART_CUSTOM_NUM_1
- CONFIG_CONSOLE_UART_RX_GPIO (*CONFIG_ESP_CONSOLE_UART_RX_GPIO*)
- CONFIG_CONSOLE_UART_TX_GPIO (*CONFIG_ESP_CONSOLE_UART_TX_GPIO*)
- CONFIG_CXX_EXCEPTIONS (*CONFIG_COMPILER_CXX_EXCEPTIONS*)
- CONFIG_CXX_EXCEPTIONS_EMG_POOL_SIZE (*CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE*)
- CONFIG_EFUSE_SECURE_VERSION_EMULATE (*CONFIG_BOOTLOADER_EFUSE_SECURE_VERSION_EMULATE*)
- CONFIG_ENABLE_STATIC_TASK_CLEAN_UP_HOOK (*CONFIG_FREERTOS_ENABLE_STATIC_TASK_CLEAN_UP*)
- CONFIG_ESP32_APPTRACE_ONPANIC_HOST_FLUSH_TMO (*CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO*)
- CONFIG_ESP32_APPTRACE_PENDING_DATA_SIZE_MAX (*CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX*)
- CONFIG_ESP32_APPTRACE_POSTMORTEM_FLUSH_TRAX_THRESH (*CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH*)
- **CONFIG_ESP32_CORE_DUMP_DECODE (*CONFIG_ESP_COREDUMP_DECODE*)**
  - CONFIG_ESP32_CORE_DUMP_DECODE_INFO
  - CONFIG_ESP32_CORE_DUMP_DECODE_DISABLE
- CONFIG_ESP32_CORE_DUMP_MAX_TASKS_NUM (*CONFIG_ESP_COREDUMP_MAX_TASKS_NUM*)
- CONFIG_ESP32_CORE_DUMP_STACK_SIZE (*CONFIG_ESP_COREDUMP_STACK_SIZE*)
- CONFIG_ESP32_CORE_DUMP_UART_DELAY (*CONFIG_ESP_COREDUMP_UART_DELAY*)
- CONFIG_ESP32_DEBUG_STUBS_ENABLE (*CONFIG_ESP_DEBUG_STUBS_ENABLE*)
- CONFIG_ESP32_GCOV_ENABLE (*CONFIG_APPTRACE_GCOV_ENABLE*)
- CONFIG_ESP32_PTHREAD_STACK_MIN (*CONFIG_PTHREAD_STACK_MIN*)
- **CONFIG_ESP32_PTHREAD_TASK_CORE_DEFAULT (*CONFIG_PTHREAD_TASK_CORE_DEFAULT*)**
  - CONFIG_ESP32_DEFAULT_PTHREAD_CORE_NO_AFFINITY
  - CONFIG_ESP32_DEFAULT_PTHREAD_CORE_0

    - CONFIG_ESP32_DEFAULT_PTHREAD_CORE_1
- CONFIG_ESP32_PTHREAD_TASK_NAME_DEFAULT (*CONFIG_PTHREAD_TASK_NAME_DEFAULT*)
- CONFIG_ESP32_PTHREAD_TASK_PRIO_DEFAULT (*CONFIG_PTHREAD_TASK_PRIO_DEFAULT*)
- CONFIG_ESP32_PTHREAD_TASK_STACK_SIZE_DEFAULT (*CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT*)
- CONFIG_ESP32_RTC_XTAL_BOOTSTRAP_CYCLES (*CONFIG_ESP_SYSTEM_RTC_EXT_XTAL_BOOTSTRAP_CYCLES*)
- CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED (*CONFIG_ESP_WIFI_AMPDU_RX_ENABLED*)
- CONFIG_ESP32_WIFI_AMPDU_TX_ENABLED (*CONFIG_ESP_WIFI_AMPDU_TX_ENABLED*)
- CONFIG_ESP32_WIFI_AMSDU_TX_ENABLED (*CONFIG_ESP_WIFI_AMSDU_TX_ENABLED*)
- CONFIG_ESP32_WIFI_CACHE_TX_BUFFER_NUM (*CONFIG_ESP_WIFI_CACHE_TX_BUFFER_NUM*)
- CONFIG_ESP32_WIFI_CSI_ENABLED (*CONFIG_ESP_WIFI_CSI_ENABLED*)
- CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM (*CONFIG_ESP_WIFI_DYNAMIC_RX_BUFFER_NUM*)
- CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER_NUM (*CONFIG_ESP_WIFI_DYNAMIC_TX_BUFFER_NUM*)
- CONFIG_ESP32_WIFI_ENABLE_WPA3_OWE_STA (*CONFIG_ESP_WIFI_ENABLE_WPA3_OWE_STA*)
- CONFIG_ESP32_WIFI_ENABLE_WPA3_SAE (*CONFIG_ESP_WIFI_ENABLE_WPA3_SAE*)
- CONFIG_ESP32_WIFI_IRAM_OPT (*CONFIG_ESP_WIFI_IRAM_OPT*)
- CONFIG_ESP32_WIFI_MGMT_SBUF_NUM (*CONFIG_ESP_WIFI_MGMT_SBUF_NUM*)
- CONFIG_ESP32_WIFI_NVS_ENABLED (*CONFIG_ESP_WIFI_NVS_ENABLED*)
- CONFIG_ESP32_WIFI_RX_BA_WIN (*CONFIG_ESP_WIFI_RX_BA_WIN*)
- CONFIG_ESP32_WIFI_RX_IRAM_OPT (*CONFIG_ESP_WIFI_RX_IRAM_OPT*)
- CONFIG_ESP32_WIFI_SOFTAP_BEACON_MAX_LEN (*CONFIG_ESP_WIFI_SOFTAP_BEACON_MAX_LEN*)
- CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM (*CONFIG_ESP_WIFI_STATIC_RX_BUFFER_NUM*)
- CONFIG_ESP32_WIFI_STATIC_TX_BUFFER_NUM (*CONFIG_ESP_WIFI_STATIC_TX_BUFFER_NUM*)
- CONFIG_ESP32_WIFI_SW_COEXIST_ENABLE (*CONFIG_ESP_COEX_SW_COEXIST_ENABLE*)
- **CONFIG_ESP32_WIFI_TASK_CORE_ID (*CONFIG_ESP_WIFI_TASK_CORE_ID*)**
    - CONFIG_ESP32_WIFI_TASK_PINNED_TO_CORE_0
    - CONFIG_ESP32_WIFI_TASK_PINNED_TO_CORE_1
- CONFIG_ESP32_WIFI_TX_BA_WIN (*CONFIG_ESP_WIFI_TX_BA_WIN*)
- **CONFIG_ESP32_WIFI_TX_BUFFER (*CONFIG_ESP_WIFI_TX_BUFFER*)**
    - CONFIG_ESP32_WIFI_STATIC_TX_BUFFER
    - CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER
- CONFIG_ESP_GRATUITOUS_ARP (*CONFIG_LWIP_ESP_GRATUITOUS_ARP*)
- CONFIG_ESP_SYSTEM_PD_FLASH (*CONFIG_ESP_SLEEP_POWER_DOWN_FLASH*)
- CONFIG_ESP_SYSTEM_PM_POWER_DOWN_CPU (*CONFIG_PM_POWER_DOWN_CPU_IN_LIGHT_SLEEP*)
- CONFIG_ESP_TASK_WDT (*CONFIG_ESP_TASK_WDT_INIT*)
- CONFIG_ESP_WIFI_EXTERNAL_COEXIST_ENABLE (*CONFIG_ESP_COEX_EXTERNAL_COEXIST_ENABLE*)
- CONFIG_ESP_WIFI_SW_COEXIST_ENABLE (*CONFIG_ESP_COEX_SW_COEXIST_ENABLE*)
- CONFIG_EVENT_LOOP_PROFILING (*CONFIG_ESP_EVENT_LOOP_PROFILING*)
- CONFIG_EXTERNAL_COEX_ENABLE (*CONFIG_ESP_COEX_EXTERNAL_COEXIST_ENABLE*)
- CONFIG_FLASH_ENCRYPTION_ENABLED (*CONFIG_SECURE_FLASH_ENC_ENABLED*)
- CONFIG_FLASH_ENCRYPTION_UART_BOOTLOADER_ALLOW_CACHE (*CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE*)
- CONFIG_FLASH_ENCRYPTION_UART_BOOTLOADER_ALLOW_ENCRYPT (*CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC*)
- **CONFIG_GAP_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_GAP_TRACE_LEVEL*)**
    - CONFIG_GAP_TRACE_LEVEL_NONE
    - CONFIG_GAP_TRACE_LEVEL_ERROR
    - CONFIG_GAP_TRACE_LEVEL_WARNING
    - CONFIG_GAP_TRACE_LEVEL_API
    - CONFIG_GAP_TRACE_LEVEL_EVENT
    - CONFIG_GAP_TRACE_LEVEL_DEBUG
    - CONFIG_GAP_TRACE_LEVEL_VERBOSE
- CONFIG_GARP_TMR_INTERVAL (*CONFIG_LWIP_GARP_TMR_INTERVAL*)
- CONFIG_GATTC_CACHE_NVS_FLASH (*CONFIG_BT_GATTC_CACHE_NVS_FLASH*)
- CONFIG_GATTC_ENABLE (*CONFIG_BT_GATTC_ENABLE*)
- CONFIG_GATTS_ENABLE (*CONFIG_BT_GATTS_ENABLE*)
- **CONFIG_GATTS_SEND_SERVICE_CHANGE_MODE (*CONFIG_BT_GATTS_SEND_SERVICE_CHANGE_MODE*)**

    - CONFIG_GATTS_SEND_SERVICE_CHANGE_MANUAL

  - CONFIG_GATTS_SEND_SERVICE_CHANGE_AUTO
- **CONFIG_GATT_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_GATT_TRACE_LEVEL***)
  - CONFIG_GATT_TRACE_LEVEL_NONE
  - CONFIG_GATT_TRACE_LEVEL_ERROR
  - CONFIG_GATT_TRACE_LEVEL_WARNING
  - CONFIG_GATT_TRACE_LEVEL_API
  - CONFIG_GATT_TRACE_LEVEL_EVENT
  - CONFIG_GATT_TRACE_LEVEL_DEBUG
  - CONFIG_GATT_TRACE_LEVEL_VERBOSE
- CONFIG_GDBSTUB_MAX_TASKS (*CONFIG_ESP_GDBSTUB_MAX_TASKS*)
- CONFIG_GDBSTUB_SUPPORT_TASKS (*CONFIG_ESP_GDBSTUB_SUPPORT_TASKS*)
- **CONFIG_HCI_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_HCI_TRACE_LEVEL***)
  - CONFIG_HCI_TRACE_LEVEL_NONE
  - CONFIG_HCI_TRACE_LEVEL_ERROR
  - CONFIG_HCI_TRACE_LEVEL_WARNING
  - CONFIG_HCI_TRACE_LEVEL_API
  - CONFIG_HCI_TRACE_LEVEL_EVENT
  - CONFIG_HCI_TRACE_LEVEL_DEBUG
  - CONFIG_HCI_TRACE_LEVEL_VERBOSE
- CONFIG_HFP_AG_ENABLE (*CONFIG_BT_HFP_AG_ENABLE*)
- **CONFIG_HFP_AUDIO_DATA_PATH** (***CONFIG_BT_HFP_AUDIO_DATA_PATH***)
  - CONFIG_HFP_AUDIO_DATA_PATH_PCM
  - CONFIG_HFP_AUDIO_DATA_PATH_HCI
- CONFIG_HFP_CLIENT_ENABLE (*CONFIG_BT_HFP_CLIENT_ENABLE*)
- CONFIG_HFP_ENABLE (*CONFIG_BT_HFP_ENABLE*)
- **CONFIG_HID_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_HID_TRACE_LEVEL***)
  - CONFIG_HID_TRACE_LEVEL_NONE
  - CONFIG_HID_TRACE_LEVEL_ERROR
  - CONFIG_HID_TRACE_LEVEL_WARNING
  - CONFIG_HID_TRACE_LEVEL_API
  - CONFIG_HID_TRACE_LEVEL_EVENT
  - CONFIG_HID_TRACE_LEVEL_DEBUG
  - CONFIG_HID_TRACE_LEVEL_VERBOSE
- CONFIG_INT_WDT (*CONFIG_ESP_INT_WDT*)
- CONFIG_INT_WDT_CHECK_CPU1 (*CONFIG_ESP_INT_WDT_CHECK_CPU1*)
- CONFIG_INT_WDT_TIMEOUT_MS (*CONFIG_ESP_INT_WDT_TIMEOUT_MS*)
- CONFIG_IPC_TASK_STACK_SIZE (*CONFIG_ESP_IPC_TASK_STACK_SIZE*)
- **CONFIG_L2CAP_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_L2CAP_TRACE_LEVEL***)
  - CONFIG_L2CAP_TRACE_LEVEL_NONE
  - CONFIG_L2CAP_TRACE_LEVEL_ERROR
  - CONFIG_L2CAP_TRACE_LEVEL_WARNING
  - CONFIG_L2CAP_TRACE_LEVEL_API
  - CONFIG_L2CAP_TRACE_LEVEL_EVENT
  - CONFIG_L2CAP_TRACE_LEVEL_DEBUG
  - CONFIG_L2CAP_TRACE_LEVEL_VERBOSE
- CONFIG_L2_TO_L3_COPY (*CONFIG_LWIP_L2_TO_L3_COPY*)
- **CONFIG_LOG_BOOTLOADER_LEVEL** (***CONFIG_BOOTLOADER_LOG_LEVEL***)
  - CONFIG_LOG_BOOTLOADER_LEVEL_NONE
  - CONFIG_LOG_BOOTLOADER_LEVEL_ERROR
  - CONFIG_LOG_BOOTLOADER_LEVEL_WARN
  - CONFIG_LOG_BOOTLOADER_LEVEL_INFO
  - CONFIG_LOG_BOOTLOADER_LEVEL_DEBUG
  - CONFIG_LOG_BOOTLOADER_LEVEL_VERBOSE
- CONFIG_MAIN_TASK_STACK_SIZE (*CONFIG_ESP_MAIN_TASK_STACK_SIZE*)
- **CONFIG_MCA_INITIAL_TRACE_LEVEL** (***CONFIG_BT_LOG_MCA_TRACE_LEVEL***)
  - CONFIG_MCA_TRACE_LEVEL_NONE
  - CONFIG_MCA_TRACE_LEVEL_ERROR
  - CONFIG_MCA_TRACE_LEVEL_WARNING

  - CONFIG_MCA_TRACE_LEVEL_API
  - CONFIG_MCA_TRACE_LEVEL_EVENT
  - CONFIG_MCA_TRACE_LEVEL_DEBUG
  - CONFIG_MCA_TRACE_LEVEL_VERBOSE
- CONFIG_MCPWM_ISR_IN_IRAM (*CONFIG_MCPWM_ISR_IRAM_SAFE*)
- CONFIG_NIMBLE_ATT_PREFERRED_MTU (*CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU*)
- CONFIG_NIMBLE_CRYPTO_STACK_MBEDTLS (*CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS*)
- CONFIG_NIMBLE_DEBUG (*CONFIG_BT_NIMBLE_DEBUG*)
- CONFIG_NIMBLE_GAP_DEVICE_NAME_MAX_LEN (*CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MAX_LEN*)
- CONFIG_NIMBLE_HS_FLOW_CTRL (*CONFIG_BT_NIMBLE_HS_FLOW_CTRL*)
- CONFIG_NIMBLE_HS_FLOW_CTRL_ITVL (*CONFIG_BT_NIMBLE_HS_FLOW_CTRL_ITVL*)
- CONFIG_NIMBLE_HS_FLOW_CTRL_THRESH (*CONFIG_BT_NIMBLE_HS_FLOW_CTRL_THRESH*)
- CONFIG_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT (*CONFIG_BT_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT*)
- CONFIG_NIMBLE_L2CAP_COC_MAX_NUM (*CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM*)
- CONFIG_NIMBLE_MAX_BONDS (*CONFIG_BT_NIMBLE_MAX_BONDS*)
- CONFIG_NIMBLE_MAX_CCCDS (*CONFIG_BT_NIMBLE_MAX_CCCDS*)
- CONFIG_NIMBLE_MAX_CONNECTIONS (*CONFIG_BT_NIMBLE_MAX_CONNECTIONS*)
- **CONFIG_NIMBLE_MEM_ALLOC_MODE (*CONFIG_BT_NIMBLE_MEM_ALLOC_MODE*)**
  - CONFIG_NIMBLE_MEM_ALLOC_MODE_INTERNAL
  - CONFIG_NIMBLE_MEM_ALLOC_MODE_EXTERNAL
  - CONFIG_NIMBLE_MEM_ALLOC_MODE_DEFAULT
- CONFIG_NIMBLE_MESH (*CONFIG_BT_NIMBLE_MESH*)
- CONFIG_NIMBLE_MESH_DEVICE_NAME (*CONFIG_BT_NIMBLE_MESH_DEVICE_NAME*)
- CONFIG_NIMBLE_MESH_FRIEND (*CONFIG_BT_NIMBLE_MESH_FRIEND*)
- CONFIG_NIMBLE_MESH_GATT_PROXY (*CONFIG_BT_NIMBLE_MESH_GATT_PROXY*)
- CONFIG_NIMBLE_MESH_LOW_POWER (*CONFIG_BT_NIMBLE_MESH_LOW_POWER*)
- CONFIG_NIMBLE_MESH_PB_ADV (*CONFIG_BT_NIMBLE_MESH_PB_ADV*)
- CONFIG_NIMBLE_MESH_PB_GATT (*CONFIG_BT_NIMBLE_MESH_PB_GATT*)
- CONFIG_NIMBLE_MESH_PROV (*CONFIG_BT_NIMBLE_MESH_PROV*)
- CONFIG_NIMBLE_MESH_PROXY (*CONFIG_BT_NIMBLE_MESH_PROXY*)
- CONFIG_NIMBLE_MESH_RELAY (*CONFIG_BT_NIMBLE_MESH_RELAY*)
- CONFIG_NIMBLE_NVS_PERSIST (*CONFIG_BT_NIMBLE_NVS_PERSIST*)
- **CONFIG_NIMBLE_PINNED_TO_CORE_CHOICE (*CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE*)**
  - CONFIG_NIMBLE_PINNED_TO_CORE_0
  - CONFIG_NIMBLE_PINNED_TO_CORE_1
- CONFIG_NIMBLE_ROLE_BROADCASTER (*CONFIG_BT_NIMBLE_ROLE_BROADCASTER*)
- CONFIG_NIMBLE_ROLE_CENTRAL (*CONFIG_BT_NIMBLE_ROLE_CENTRAL*)
- CONFIG_NIMBLE_ROLE_OBSERVER (*CONFIG_BT_NIMBLE_ROLE_OBSERVER*)
- CONFIG_NIMBLE_ROLE_PERIPHERAL (*CONFIG_BT_NIMBLE_ROLE_PERIPHERAL*)
- CONFIG_NIMBLE_RPA_TIMEOUT (*CONFIG_BT_NIMBLE_RPA_TIMEOUT*)
- CONFIG_NIMBLE_SM_LEGACY (*CONFIG_BT_NIMBLE_SM_LEGACY*)
- CONFIG_NIMBLE_SM_SC (*CONFIG_BT_NIMBLE_SM_SC*)
- CONFIG_NIMBLE_SM_SC_DEBUG_KEYS (*CONFIG_BT_NIMBLE_SM_SC_DEBUG_KEYS*)
- CONFIG_NIMBLE_SVC_GAP_APPEARANCE (*CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE*)
- CONFIG_NIMBLE_SVC_GAP_DEVICE_NAME (*CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME*)
- CONFIG_NIMBLE_TASK_STACK_SIZE (*CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE*)
- CONFIG_NO_BLOBS (*CONFIG_APP_NO_BLOBS*)
- **CONFIG_OPTIMIZATION_ASSERTION_LEVEL (*CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL*)**
  - CONFIG_OPTIMIZATION_ASSERTIONS_ENABLED
  - CONFIG_OPTIMIZATION_ASSERTIONS_SILENT
  - CONFIG_OPTIMIZATION_ASSERTIONS_DISABLED
- **CONFIG_OPTIMIZATION_COMPILER (*CONFIG_COMPILER_OPTIMIZATION*)**
  - CONFIG_OPTIMIZATION_LEVEL_DEBUG, CONFIG_COMPILER_OPTIMIZATION_LEVEL_DEBUG, CONFIG_COMPILER_OPTIMIZATION_DEFAULT
  - CONFIG_OPTIMIZATION_LEVEL_RELEASE, CONFIG_COMPILER_OPTIMIZATION_LEVEL_RELEASE

- **CONFIG_OSI_INITIAL_TRACE_LEVEL** (*CONFIG_BT_LOG_OSI_TRACE_LEVEL*)
  - CONFIG_OSI_TRACE_LEVEL_NONE
  - CONFIG_OSI_TRACE_LEVEL_ERROR
  - CONFIG_OSI_TRACE_LEVEL_WARNING
  - CONFIG_OSI_TRACE_LEVEL_API
  - CONFIG_OSI_TRACE_LEVEL_EVENT
  - CONFIG_OSI_TRACE_LEVEL_DEBUG
  - CONFIG_OSI_TRACE_LEVEL_VERBOSE
- CONFIG_OTA_ALLOW_HTTP (*CONFIG_ESP_HTTPS_OTA_ALLOW_HTTP*)
- **CONFIG_PAN_INITIAL_TRACE_LEVEL** (*CONFIG_BT_LOG_PAN_TRACE_LEVEL*)
  - CONFIG_PAN_TRACE_LEVEL_NONE
  - CONFIG_PAN_TRACE_LEVEL_ERROR
  - CONFIG_PAN_TRACE_LEVEL_WARNING
  - CONFIG_PAN_TRACE_LEVEL_API
  - CONFIG_PAN_TRACE_LEVEL_EVENT
  - CONFIG_PAN_TRACE_LEVEL_DEBUG
  - CONFIG_PAN_TRACE_LEVEL_VERBOSE
- CONFIG_POST_EVENTS_FROM_IRAM_ISR (*CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR*)
- CONFIG_POST_EVENTS_FROM_ISR (*CONFIG_ESP_EVENT_POST_FROM_ISR*)
- CONFIG_PPP_CHAP_SUPPORT (*CONFIG_LWIP_PPP_CHAP_SUPPORT*)
- CONFIG_PPP_DEBUG_ON (*CONFIG_LWIP_PPP_DEBUG_ON*)
- CONFIG_PPP_MPPE_SUPPORT (*CONFIG_LWIP_PPP_MPPE_SUPPORT*)
- CONFIG_PPP_MSCHAP_SUPPORT (*CONFIG_LWIP_PPP_MSCHAP_SUPPORT*)
- CONFIG_PPP_NOTIFY_PHASE_SUPPORT (*CONFIG_LWIP_PPP_NOTIFY_PHASE_SUPPORT*)
- CONFIG_PPP_PAP_SUPPORT (*CONFIG_LWIP_PPP_PAP_SUPPORT*)
- CONFIG_PPP_SUPPORT (*CONFIG_LWIP_PPP_SUPPORT*)
- **CONFIG_RFCOMM_INITIAL_TRACE_LEVEL** (*CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL*)
  - CONFIG_RFCOMM_TRACE_LEVEL_NONE
  - CONFIG_RFCOMM_TRACE_LEVEL_ERROR
  - CONFIG_RFCOMM_TRACE_LEVEL_WARNING
  - CONFIG_RFCOMM_TRACE_LEVEL_API
  - CONFIG_RFCOMM_TRACE_LEVEL_EVENT
  - CONFIG_RFCOMM_TRACE_LEVEL_DEBUG
  - CONFIG_RFCOMM_TRACE_LEVEL_VERBOSE
- CONFIG_SEMIHOSTFS_MAX_MOUNT_POINTS (*CONFIG_VFS_SEMIHOSTFS_MAX_MOUNT_POINTS*)
- **CONFIG_SMP_INITIAL_TRACE_LEVEL** (*CONFIG_BT_LOG_SMP_TRACE_LEVEL*)
  - CONFIG_SMP_TRACE_LEVEL_NONE
  - CONFIG_SMP_TRACE_LEVEL_ERROR
  - CONFIG_SMP_TRACE_LEVEL_WARNING
  - CONFIG_SMP_TRACE_LEVEL_API
  - CONFIG_SMP_TRACE_LEVEL_EVENT
  - CONFIG_SMP_TRACE_LEVEL_DEBUG
  - CONFIG_SMP_TRACE_LEVEL_VERBOSE
- CONFIG_SMP_SLAVE_CON_PARAMS_UPD_ENABLE (*CONFIG_BT_SMP_SLAVE_CON_PARAMS_UPD_ENABLE*)
- CONFIG_SPI_FLASH_32BIT_ADDR_ENABLE (*CONFIG_BOOTLOADER_CACHE_32BIT_ADDR_QUAD_FLASH*)
- CONFIG_SPI_FLASH_QUAD_32BIT_ADDR_ENABLE (*CONFIG_BOOTLOADER_CACHE_32BIT_ADDR_QUAD_FLASH*)
- **CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS** (*CONFIG_SPI_FLASH_DANGEROUS_WRITE*)
  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_ABORTS
  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_FAILS
  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_ALLOWED
- **CONFIG_STACK_CHECK_MODE** (*CONFIG_COMPILER_STACK_CHECK_MODE*)
  - CONFIG_STACK_CHECK_NONE
  - CONFIG_STACK_CHECK_NORM
  - CONFIG_STACK_CHECK_STRONG
  - CONFIG_STACK_CHECK_ALL
- CONFIG_SUPPORT_TERMIOS (*CONFIG_VFS_SUPPORT_TERMIOS*)
- CONFIG_SUPPRESS_SELECT_DEBUG_OUTPUT (*CONFIG_VFS_SUPPRESS_SELECT_DEBUG_OUTPUT*)

- CONFIG_SW_COEXIST_ENABLE (*CONFIG_ESP_COEX_SW_COEXIST_ENABLE*)
- CONFIG_SYSTEM_EVENT_QUEUE_SIZE (*CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE*)
- CONFIG_SYSTEM_EVENT_TASK_STACK_SIZE (*CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE*)
- CONFIG_SYSVIEW_BUF_WAIT_TMO (*CONFIG_APPTRACE_SV_BUF_WAIT_TMO*)
- CONFIG_SYSVIEW_ENABLE (*CONFIG_APPTRACE_SV_ENABLE*)
- CONFIG_SYSVIEW_EVT_IDLE_ENABLE (*CONFIG_APPTRACE_SV_EVT_IDLE_ENABLE*)
- CONFIG_SYSVIEW_EVT_ISR_ENTER_ENABLE (*CONFIG_APPTRACE_SV_EVT_ISR_ENTER_ENABLE*)
- CONFIG_SYSVIEW_EVT_ISR_EXIT_ENABLE (*CONFIG_APPTRACE_SV_EVT_ISR_EXIT_ENABLE*)
- CONFIG_SYSVIEW_EVT_ISR_TO_SCHEDULER_ENABLE (*CONFIG_APPTRACE_SV_EVT_ISR_TO_SCHED_ENABLE*)
- CONFIG_SYSVIEW_EVT_OVERFLOW_ENABLE (*CONFIG_APPTRACE_SV_EVT_OVERFLOW_ENABLE*)
- CONFIG_SYSVIEW_EVT_TASK_CREATE_ENABLE (*CONFIG_APPTRACE_SV_EVT_TASK_CREATE_ENABLE*)
- CONFIG_SYSVIEW_EVT_TASK_START_EXEC_ENABLE (*CONFIG_APPTRACE_SV_EVT_TASK_START_EXEC_ENABL*
- CONFIG_SYSVIEW_EVT_TASK_START_READY_ENABLE (*CONFIG_APPTRACE_SV_EVT_TASK_START_READY_EN*
- CONFIG_SYSVIEW_EVT_TASK_STOP_EXEC_ENABLE (*CONFIG_APPTRACE_SV_EVT_TASK_STOP_EXEC_ENABLE*)
- CONFIG_SYSVIEW_EVT_TASK_STOP_READY_ENABLE (*CONFIG_APPTRACE_SV_EVT_TASK_STOP_READY_ENAB*
- CONFIG_SYSVIEW_EVT_TASK_TERMINATE_ENABLE (*CONFIG_APPTRACE_SV_EVT_TASK_TERMINATE_ENABLE*)
- CONFIG_SYSVIEW_EVT_TIMER_ENTER_ENABLE (*CONFIG_APPTRACE_SV_EVT_TIMER_ENTER_ENABLE*)
- CONFIG_SYSVIEW_EVT_TIMER_EXIT_ENABLE (*CONFIG_APPTRACE_SV_EVT_TIMER_EXIT_ENABLE*)
- CONFIG_SYSVIEW_MAX_TASKS (*CONFIG_APPTRACE_SV_MAX_TASKS*)
- **CONFIG_SYSVIEW_TS_SOURCE (*CONFIG_APPTRACE_SV_TS_SOURCE*)**
  - CONFIG_SYSVIEW_TS_SOURCE_CCOUNT
  - CONFIG_SYSVIEW_TS_SOURCE_ESP_TIMER
- CONFIG_TASK_WDT (*CONFIG_ESP_TASK_WDT_INIT*)
- CONFIG_TASK_WDT_CHECK_IDLE_TASK_CPU0 (*CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0*)
- CONFIG_TASK_WDT_CHECK_IDLE_TASK_CPU1 (*CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1*)
- CONFIG_TASK_WDT_PANIC (*CONFIG_ESP_TASK_WDT_PANIC*)
- CONFIG_TASK_WDT_TIMEOUT_S (*CONFIG_ESP_TASK_WDT_TIMEOUT_S*)
- CONFIG_TCPIP_RECVMBOX_SIZE (*CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*)
- **CONFIG_TCPIP_TASK_AFFINITY (*CONFIG_LWIP_TCPIP_TASK_AFFINITY*)**
  - CONFIG_TCPIP_TASK_AFFINITY_NO_AFFINITY
  - CONFIG_TCPIP_TASK_AFFINITY_CPU0
  - CONFIG_TCPIP_TASK_AFFINITY_CPU1
- CONFIG_TCPIP_TASK_STACK_SIZE (*CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*)
- CONFIG_TCP_MAXRTX (*CONFIG_LWIP_TCP_MAXRTX*)
- CONFIG_TCP_MSL (*CONFIG_LWIP_TCP_MSL*)
- CONFIG_TCP_MSS (*CONFIG_LWIP_TCP_MSS*)
- **CONFIG_TCP_OVERSIZE (*CONFIG_LWIP_TCP_OVERSIZE*)**
  - CONFIG_TCP_OVERSIZE_MSS
  - CONFIG_TCP_OVERSIZE_QUARTER_MSS
  - CONFIG_TCP_OVERSIZE_DISABLE
- CONFIG_TCP_QUEUE_OOSEQ (*CONFIG_LWIP_TCP_QUEUE_OOSEQ*)
- CONFIG_TCP_RECVMBOX_SIZE (*CONFIG_LWIP_TCP_RECVMBOX_SIZE*)
- CONFIG_TCP_SND_BUF_DEFAULT (*CONFIG_LWIP_TCP_SND_BUF_DEFAULT*)
- CONFIG_TCP_SYNMAXRTX (*CONFIG_LWIP_TCP_SYNMAXRTX*)
- CONFIG_TCP_WND_DEFAULT (*CONFIG_LWIP_TCP_WND_DEFAULT*)
- CONFIG_TIMER_QUEUE_LENGTH (*CONFIG_FREERTOS_TIMER_QUEUE_LENGTH*)
- CONFIG_TIMER_TASK_PRIORITY (*CONFIG_FREERTOS_TIMER_TASK_PRIORITY*)
- CONFIG_TIMER_TASK_STACK_DEPTH (*CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH*)
- CONFIG_TIMER_TASK_STACK_SIZE (*CONFIG_ESP_TIMER_TASK_STACK_SIZE*)
- CONFIG_UDP_RECVMBOX_SIZE (*CONFIG_LWIP_UDP_RECVMBOX_SIZE*)
- CONFIG_WARN_WRITE_STRINGS (*CONFIG_COMPILER_WARN_WRITE_STRINGS*)
- CONFIG_WPA_11KV_SUPPORT (*CONFIG_ESP_WIFI_11KV_SUPPORT*)
- CONFIG_WPA_11R_SUPPORT (*CONFIG_ESP_WIFI_11R_SUPPORT*)
- CONFIG_WPA_DEBUG_PRINT (*CONFIG_ESP_WIFI_DEBUG_PRINT*)
- CONFIG_WPA_DPP_SUPPORT (*CONFIG_ESP_WIFI_DPP_SUPPORT*)
- CONFIG_WPA_MBEDTLS_CRYPTO (*CONFIG_ESP_WIFI_MBEDTLS_CRYPTO*)
- CONFIG_WPA_MBEDTLS_TLS_CLIENT (*CONFIG_ESP_WIFI_MBEDTLS_TLS_CLIENT*)
- CONFIG_WPA_MBO_SUPPORT (*CONFIG_ESP_WIFI_MBO_SUPPORT*)

- CONFIG_WPA_SCAN_CACHE (*CONFIG_ESP_WIFI_SCAN_CACHE*)
- CONFIG_WPA_SUITE_B_192 (*CONFIG_ESP_WIFI_SUITE_B_192*)
- CONFIG_WPA_TESTING_OPTIONS (*CONFIG_ESP_WIFI_TESTING_OPTIONS*)
- CONFIG_WPA_WAPI_PSK (*CONFIG_ESP_WIFI_WAPI_PSK*)
- CONFIG_WPA_WPS_SOFTAP_REGISTRAR (*CONFIG_ESP_WIFI_WPS_SOFTAP_REGISTRAR*)
- CONFIG_WPA_WPS_STRICT (*CONFIG_ESP_WIFI_WPS_STRICT*)

2.7 配网 API

2.7.1 协议通信

概述

协议通信 (protocomm) 组件用于管理安全会话并为多种传输提供框架。应用程序还可以直接使用 protocomm 层来增加特定扩展，用于配网或非配网使用场景。

以下功能可用于配网：

- 应用程序层面的通信安全
  - `protocomm_security0` （无安全功能）
  - `protocomm_security1` （Curve25519 密钥交换 + AES-CTR 加密/解密）
  - `protocomm_security2` （基于 SRP6a 的密钥交换 + AES-GCM 加密/解密）
- 所有权验证 (Proof-of-possession)（仅 `protocomm_security1` 支持该功能）
- 盐值和验证器 (Salt and Verifier)（仅 `protocomm_security2` 支持该功能）

在 protocomm 内部，protobuf（协议缓冲区）用于建立安全会话。用户可以自行选择（即使在不使用 Protobuf 的情况下）实现安全性，也可以在没有任何安全层的情况下使用协议。

Protocomm 为以下各种传输提供框架：

- 控制台：使用该传输方案时，设备端会自动调用处理程序。相关代码片段，请参见下文传输示例。

请注意，对于 `protocomm_security1` 和 `protocomm_security2`，客户端仍需要执行双向握手来建立会话。

启用 protocomm 安全版本

关于启用/禁用相应的安全版本，请参阅 protocomm 组件的项目配置菜单。相应配置选项如下：

- 支持 `protocomm_security0`，该版本无安全功能：*CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0*，该选项默认启用。
- 支持 `protocomm_security1`，使用 Curve25519 密钥交换和 AES-CTR 加密/解密：*CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1*，该选项默认启用。
- 支持 `protocomm_security2`，使用基于 SRP6a 的密钥交换和 AES-GCM 加密/解密：*CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2*。

备注： 启用多个安全版本后可以动态控制安全版本，但也会增加固件大小。

API 参考

Header File

- components/protocomm/include/common/protocomm.h
- This header file can be included with:

```
#include "protocomm.h"
```

- This header file is a part of the API provided by the `protocomm` component. To declare that your component depends on `protocomm`, add the following to your CMakeLists.txt:

```
REQUIRES protocomm
```

or

```
PRIV_REQUIRES protocomm
```

Functions

protocomm_t ***protocomm_new**(void)

Create a new protocomm instance.

This API will return a new dynamically allocated protocomm instance with all elements of the protocomm_t structure initialized to NULL.

返回

- protocomm_t* : On success
- NULL : No memory for allocating new instance

void **protocomm_delete**(*protocomm_t* *pc)

Delete a protocomm instance.

This API will deallocate a protocomm instance that was created using `protocomm_new()`.

参数 **pc** -- **[in]** Pointer to the protocomm instance to be deleted

esp_err_t **protocomm_add_endpoint**(*protocomm_t* *pc, const char *ep_name, *protocomm_req_handler_t* h, void *priv_data)

Add endpoint request handler for a protocomm instance.

This API will bind an endpoint handler function to the specified endpoint name, along with any private data that needs to be pass to the handler at the time of call.

备注:

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
- This function internally calls the registered `add_endpoint()` function of the selected transport which is a member of the protocomm_t instance structure.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **ep_name** -- **[in]** Endpoint identifier(name) string
- **h** -- **[in]** Endpoint handler function
- **priv_data** -- **[in]** Pointer to private data to be passed as a parameter to the handler function on call. Pass NULL if not needed.

返回

- ESP_OK : Success
- ESP_FAIL : Error adding endpoint / Endpoint with this name already exists
- ESP_ERR_NO_MEM : Error allocating endpoint resource
- ESP_ERR_INVALID_ARG : Null instance/name/handler arguments

esp_err_t **protocomm_remove_endpoint** (*protocomm_t* *pc, const char *ep_name)

Remove endpoint request handler for a protocomm instance.

This API will remove a registered endpoint handler identified by an endpoint name.

备注:

- This function internally calls the registered `remove_endpoint()` function which is a member of the protocomm_t instance structure.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **ep_name** -- **[in]** Endpoint identifier(name) string

返回

- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Endpoint with specified name doesn't exist
- ESP_ERR_INVALID_ARG : Null instance/name arguments

esp_err_t **protocomm_open_session** (*protocomm_t* *pc, uint32_t session_id)

Allocates internal resources for new transport session.

备注:

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **session_id** -- **[in]** Unique ID for a communication session

返回

- ESP_OK : Request handled successfully
- ESP_ERR_NO_MEM : Error allocating internal resource
- ESP_ERR_INVALID_ARG : Null instance/name arguments

esp_err_t **protocomm_close_session** (*protocomm_t* *pc, uint32_t session_id)

Frees internal resources used by a transport session.

备注:

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **session_id** -- **[in]** Unique ID for a communication session

返回

- ESP_OK : Request handled successfully
- ESP_ERR_INVALID_ARG : Null instance/name arguments

esp_err_t **protocomm_req_handle** (*protocomm_t* *pc, const char *ep_name, uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen)

Calls the registered handler of an endpoint session for processing incoming data and generating the response.

备注:

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
- Resulting output buffer must be deallocated by the caller.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **ep_name** -- **[in]** Endpoint identifier(name) string
- **session_id** -- **[in]** Unique ID for a communication session
- **inbuf** -- **[in]** Input buffer contains input request data which is to be processed by the registered handler
- **inlen** -- **[in]** Length of the input buffer
- **outbuf** -- **[out]** Pointer to internally allocated output buffer, where the resulting response data output from the registered handler is to be stored
- **outlen** -- **[out]** Buffer length of the allocated output buffer

返回

- ESP_OK : Request handled successfully
- ESP_FAIL : Internal error in execution of registered handler
- ESP_ERR_NO_MEM : Error allocating internal resource
- ESP_ERR_NOT_FOUND : Endpoint with specified name doesn't exist
- ESP_ERR_INVALID_ARG : Null instance/name arguments

esp_err_t **protocomm_set_security** (*protocomm_t* *pc, const char *ep_name, const *protocomm_security_t* *sec, const void *sec_params)

Add endpoint security for a protocomm instance.

This API will bind a security session establisher to the specified endpoint name, along with any proof of possession that may be required for authenticating a session client.

备注:

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
- The choice of security can be any `protocomm_security_t` instance. Choices `protocomm_security0` and `protocomm_security1` and `protocomm_security2` are readily available.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **ep_name** -- **[in]** Endpoint identifier(name) string
- **sec** -- **[in]** Pointer to endpoint security instance
- **sec_params** -- **[in]** Pointer to security params (NULL if not needed) The pointer should contain the security params struct of appropriate security version. For protocomm security version 1 and 2 sec_params should contain pointer to struct of type protocomm_security1_params_t and protocmm_security2_params_t respectively. The contents of this pointer must be valid till the security session has been running and is not closed.

返回

- ESP_OK : Success
- ESP_FAIL : Error adding endpoint / Endpoint with this name already exists
- ESP_ERR_INVALID_STATE : Security endpoint already set
- ESP_ERR_NO_MEM : Error allocating endpoint resource
- ESP_ERR_INVALID_ARG : Null instance/name/handler arguments

esp_err_t **protocomm_unset_security** (*protocomm_t* *pc, const char *ep_name)

Remove endpoint security for a protocomm instance.

This API will remove a registered security endpoint identified by an endpoint name.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **ep_name** -- **[in]** Endpoint identifier(name) string

返回

- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Endpoint with specified name doesn't exist
- ESP_ERR_INVALID_ARG : Null instance/name arguments

esp_err_t **protocomm_set_version**(*protocomm_t* *pc, const char *ep_name, const char *version)

Set endpoint for version verification.

This API can be used for setting an application specific protocol version which can be verified by clients through the endpoint.

备注:

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **ep_name** -- **[in]** Endpoint identifier(name) string
- **version** -- **[in]** Version identifier(name) string

返回

- ESP_OK : Success
- ESP_FAIL : Error adding endpoint / Endpoint with this name already exists
- ESP_ERR_INVALID_STATE : Version endpoint already set
- ESP_ERR_NO_MEM : Error allocating endpoint resource
- ESP_ERR_INVALID_ARG : Null instance/name/handler arguments

esp_err_t **protocomm_unset_version**(*protocomm_t* *pc, const char *ep_name)

Remove version verification endpoint from a protocomm instance.

This API will remove a registered version endpoint identified by an endpoint name.

参数

- **pc** -- **[in]** Pointer to the protocomm instance
- **ep_name** -- **[in]** Endpoint identifier(name) string

返回

- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Endpoint with specified name doesn't exist
- ESP_ERR_INVALID_ARG : Null instance/name arguments

Type Definitions

typedef *esp_err_t* (***protocomm_req_handler_t**)(uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen, void *priv_data)

Function prototype for protocomm endpoint handler.

typedef struct protocomm **protocomm_t**

This structure corresponds to a unique instance of protocomm returned when the API `protocomm_new()` is called. The remaining Protocomm APIs require this object as the first parameter.

备注: Structure of the protocomm object is kept private

Header File

- components/protocomm/include/security/protocomm_security.h
- This header file can be included with:

```
#include "protocomm_security.h"
```

- This header file is a part of the API provided by the `protocomm` component. To declare that your component depends on `protocomm`, add the following to your CMakeLists.txt:

```
REQUIRES protocomm
```

or

```
PRIV_REQUIRES protocomm
```

Structures

struct **protocomm_security1_params**

Protocomm Security 1 parameters: Proof Of Possession.

Public Members

const uint8_t ***data**

Pointer to buffer containing the proof of possession data

uint16_t **len**

Length (in bytes) of the proof of possession data

struct **protocomm_security2_params**

Protocomm Security 2 parameters: Salt and Verifier.

Public Members

const char ***salt**

Pointer to the buffer containing the salt

uint16_t **salt_len**

Length (in bytes) of the salt

const char ***verifier**

Pointer to the buffer containing the verifier

uint16_t **verifier_len**

Length (in bytes) of the verifier

struct **protocomm_security**

Protocomm security object structure.

The member functions are used for implementing secure protocomm sessions.

备注： This structure should not have any dynamic members to allow re-entrancy

Public Members

int **ver**

Unique version number of security implementation

esp_err_t (***init**)(*protocomm_security_handle_t* *handle)

Function for initializing/allocating security infrastructure

esp_err_t (***cleanup**)(*protocomm_security_handle_t* handle)

Function for deallocating security infrastructure

esp_err_t (***new_transport_session**)(*protocomm_security_handle_t* handle, uint32_t session_id)

Starts new secure transport session with specified ID

esp_err_t (***close_transport_session**)(*protocomm_security_handle_t* handle, uint32_t session_id)

Closes a secure transport session with specified ID

esp_err_t (***security_req_handler**)(*protocomm_security_handle_t* handle, const void *sec_params, uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen, void *priv_data)

Handler function for authenticating connection request and establishing secure session

esp_err_t (***encrypt**)(*protocomm_security_handle_t* handle, uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen)

Function which implements the encryption algorithm

esp_err_t (***decrypt**)(*protocomm_security_handle_t* handle, uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen)

Function which implements the decryption algorithm

Type Definitions

typedef struct *protocomm_security1_params* **protocomm_security1_params_t**

Protocomm Security 1 parameters: Proof Of Possession.

typedef *protocomm_security1_params_t* **protocomm_security_pop_t**

typedef struct *protocomm_security2_params* **protocomm_security2_params_t**

Protocomm Security 2 parameters: Salt and Verifier.

typedef void ***protocomm_security_handle_t**

typedef struct *protocomm_security* **protocomm_security_t**

Protocomm security object structure.

The member functions are used for implementing secure protocomm sessions.

备注： This structure should not have any dynamic members to allow re-entrancy

Enumerations

enum **protocomm_security_session_event_t**

Events generated by the protocomm security layer.

These events are generated while establishing secured session.

Values:

enumerator **PROTOCOMM_SECURITY_SESSION_SETUP_OK**

Secured session established successfully

enumerator **PROTOCOMM_SECURITY_SESSION_INVALID_SECURITY_PARAMS**

Received invalid (NULL) security parameters (username / client public-key)

enumerator **PROTOCOMM_SECURITY_SESSION_CREDENTIALS_MISMATCH**

Received incorrect credentials (username / PoP)

Header File

- components/protocomm/include/security/protocomm_security0.h
- This header file can be included with:

```
#include "protocomm_security0.h"
```

- This header file is a part of the API provided by the `protocomm` component. To declare that your component depends on `protocomm`, add the following to your CMakeLists.txt:

```
REQUIRES protocomm
```

or

```
PRIV_REQUIRES protocomm
```

Header File

- components/protocomm/include/security/protocomm_security1.h
- This header file can be included with:

```
#include "protocomm_security1.h"
```

- This header file is a part of the API provided by the `protocomm` component. To declare that your component depends on `protocomm`, add the following to your CMakeLists.txt:

```
REQUIRES protocomm
```

or

```
PRIV_REQUIRES protocomm
```

Header File

- components/protocomm/include/security/protocomm_security2.h
- This header file can be included with:

```
#include "protocomm_security2.h"
```

- This header file is a part of the API provided by the `protocomm` component. To declare that your component depends on `protocomm`, add the following to your CMakeLists.txt:

```
REQUIRES protocomm
```

or

```
PRIV_REQUIRES protocomm
```

Header File

- components/protocomm/include/crypto/srp6a/esp_srp.h
- This header file can be included with:

```
#include "esp_srp.h"
```

- This header file is a part of the API provided by the `protocomm` component. To declare that your component depends on `protocomm`, add the following to your CMakeLists.txt:

```
REQUIRES protocomm
```

or

```
PRIV_REQUIRES protocomm
```

Functions

esp_srp_handle_t ***esp_srp_init** (*esp_ng_type_t* ng)

Initialize srp context for given NG type.

备注： the handle gets freed with `esp_srp_free`

参数 **ng** -- NG type given by `esp_ng_type_t`
返回 esp_srp_handle_t* srp handle

void **esp_srp_free** (*esp_srp_handle_t* *hd)

free esp_srp_context

参数 **hd** -- handle to be free

esp_err_t **esp_srp_srv_pubkey** (*esp_srp_handle_t* *hd, const char *username, int username_len, const char *pass, int pass_len, int salt_len, char **bytes_B, int *len_B, char **bytes_salt)

Returns B (pub key) and salt. [Step2.b].

备注： *bytes_B MUST NOT BE FREED BY THE CALLER

备注： *bytes_salt MUST NOT BE FREE BY THE CALLER

参数

- **hd** -- esp_srp handle
- **username** -- Username not expected NULL terminated
- **username_len** -- Username length
- **pass** -- Password not expected to be NULL terminated
- **pass_len** -- Pasword length
- **salt_len** -- Salt length
- **bytes_B** -- Public Key returned
- **len_B** -- Length of the public key
- **bytes_salt** -- Salt bytes generated

返回 esp_err_t ESP_OK on success, appropriate error otherwise

esp_err_t **esp_srp_gen_salt_verifier**(const char *username, int username_len, const char *pass, int pass_len, char **bytes_salt, int salt_len, char **verifier, int *verifier_len)

Generate salt-verifier pair, given username, password and salt length.

备注: if API has returned ESP_OK, salt and verifier generated need to be freed by caller

备注: Usually, username and password are not saved on the device. Rather salt and verifier are generated outside the device and are embedded. this covenience API can be used to generate salt and verifier on the fly for development use case. OR for devices which intentionally want to generate different password each time and can send it to the client securely. e.g., a device has a display and it shows the pin

参数

- **username** -- **[in]** username
- **username_len** -- **[in]** length of the username
- **pass** -- **[in]** password
- **pass_len** -- **[in]** length of the password
- **bytes_salt** -- **[out]** generated salt on successful generation, or NULL
- **salt_len** -- **[in]** salt length
- **verifier** -- **[out]** generated verifier on successful generation, or NULL
- **verifier_len** -- **[out]** length of the generated verifier

返回 esp_err_t ESP_OK on success, appropriate error otherwise

esp_err_t **esp_srp_set_salt_verifier**(*esp_srp_handle_t* *hd, const char *salt, int salt_len, const char *verifier, int verifier_len)

Set the Salt and Verifier pre-generated for a given password. This should be used only if the actual password is not available. The public key can then be generated using *esp_srp_srv_pubkey_from_salt_verifier()* and not *esp_srp_srv_pubkey()*

参数

- **hd** -- esp_srp_handle
- **salt** -- pre-generated salt bytes
- **salt_len** -- length of the salt bytes
- **verifier** -- pre-generated verifier
- **verifier_len** -- length of the verifier bytes

返回 esp_err_t ESP_OK on success, appropriate error otherwise

esp_err_t **esp_srp_srv_pubkey_from_salt_verifier**(*esp_srp_handle_t* *hd, char **bytes_B, int *len_B)

Returns B (pub key)[Step2.b] when the salt and verifier are set using *esp_srp_set_salt_verifier()*

备注: *bytes_B MUST NOT BE FREED BY THE CALLER

参数

- **hd** -- esp_srp handle
- **bytes_B** -- Key returned to the called
- **len_B** -- Length of the key returned

返回 esp_err_t ESP_OK on success, appropriate error otherwise

esp_err_t **esp_srp_get_session_key**(*esp_srp_handle_t* *hd, char *bytes_A, int len_A, char **bytes_key, uint16_t *len_key)

Get session key in `*bytes_key` given by len in `*len_key`. [Step2.c].

This calculated session key is used for further communication given the proofs are exchanged/authenticated with `esp_srp_exchange_proofs`

备注： *bytes_key MUST NOT BE FREED BY THE CALLER

参数

- **hd** -- esp_srp handle
- **bytes_A** -- Private Key
- **len_A** -- Private Key length
- **bytes_key** -- Key returned to the caller
- **len_key** -- length of the key in *bytes_key

返回 esp_err_t ESP_OK on success, appropriate error otherwise

esp_err_t **esp_srp_exchange_proofs** (*esp_srp_handle_t* *hd, char *username, uint16_t username_len, char *bytes_user_proof, char *bytes_host_proof)

Complete the authentication. If this step fails, the session_key exchanged should not be used.

This is the final authentication step in SRP algorithm [Step4.1, Step4.b, Step4.c]

参数

- **hd** -- esp_srp handle
- **username** -- Username not expected NULL terminated
- **username_len** -- Username length
- **bytes_user_proof** -- param in
- **bytes_host_proof** -- parameter out (should be SHA512_DIGEST_LENGTH) bytes in size

返回 esp_err_t ESP_OK if user's proof is ok and subsequently bytes_host_proof is populated with our own proof.

Type Definitions

typedef struct esp_srp_handle **esp_srp_handle_t**

esp_srp handle as the result of `esp_srp_init`

The handle is returned by `esp_srp_init` on successful init. It is then passed for subsequent API calls as an argument. `esp_srp_free` can be used to clean up the handle. After `esp_srp_free` the handle becomes invalid.

Enumerations

enum **esp_ng_type_t**

Large prime+generator to be used for the algorithm.

Values:

enumerator **ESP_NG_3072**

Header File

- components/protocomm/include/transports/protocomm_httpd.h
- This header file can be included with:

```
#include "protocomm_httpd.h"
```

- This header file is a part of the API provided by the `protocomm` component. To declare that your component depends on `protocomm`, add the following to your CMakeLists.txt:

```
REQUIRES protocomm
```

or

```
PRIV_REQUIRES protocomm
```

Functions

esp_err_t **protocomm_httpd_start** (*protocomm_t* *pc, const *protocomm_httpd_config_t* *config)

Start HTTPD protocomm transport.

This API internally creates a framework to allow endpoint registration and security configuration for the protocomm.

备注: This is a singleton. ie. Protocomm can have multiple instances, but only one instance can be bound to an HTTP transport layer.

参数

- **pc** -- **[in]** Protocomm instance pointer obtained from protocomm_new()
- **config** -- **[in]** Pointer to config structure for initializing HTTP server

返回

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_NOT_SUPPORTED : Transport layer bound to another protocomm instance
- ESP_ERR_INVALID_STATE : Transport layer already bound to this protocomm instance
- ESP_ERR_NO_MEM : Memory allocation for server resource failed
- ESP_ERR_HTTPD_* : HTTP server error on start

esp_err_t **protocomm_httpd_stop** (*protocomm_t* *pc)

Stop HTTPD protocomm transport.

This API cleans up the HTTPD transport protocomm and frees all the handlers registered with the protocomm.

参数 **pc** -- **[in]** Same protocomm instance that was passed to protocomm_httpd_start()

返回

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : Null / incorrect protocomm instance pointer

Unions

union **protocomm_httpd_config_data_t**

#include <protocomm_httpd.h> Protocomm HTTPD Configuration Data

Public Members

void ***handle**

HTTP Server Handle, if ext_handle_provided is set to true

protocomm_http_server_config_t **config**

HTTP Server Configuration, if a server is not already active

Structures

struct **protocomm_http_server_config_t**

Config parameters for protocomm HTTP server.

Public Members

uint16_t **port**

Port on which the HTTP server will listen

size_t **stack_size**

Stack size of server task, adjusted depending upon stack usage of endpoint handler

unsigned **task_priority**

Priority of server task

struct **protocomm_httpd_config_t**

Config parameters for protocomm HTTP server.

Public Members

bool **ext_handle_provided**

Flag to indicate of an external HTTP Server Handle has been provided. In such as case, protocomm will use the same HTTP Server and not start a new one internally.

protocomm_httpd_config_data_t **data**

Protocomm HTTPD Configuration Data

Macros

PROTOCOMM_HTTPD_DEFAULT_CONFIG()

2.8 存储 API

本节提供高层次的存储 API 的参考文档。这些 API 基于如 SPI flash、SD/MMC 等低层次驱动。

- *分区表 API* 基于*分区表*，允许以块为单位访问 SPI flash。
- *非易失性存储库 (NVS)* 在 SPI NOR flash 上实现了一个有容错性，和磨损均衡功能的键值对存储。
- *虚拟文件系统 (VFS)* 库提供了一个用于注册文件系统驱动的接口。SPIFFS、FAT 以及多种其他的文件系统库都基于 VFS。
- *SPIFFS* 是一个专为 SPI NOR flash 优化的磨损均衡的文件系统，非常适用于小分区和低吞吐率的应用。
- *FAT* 是一个可用于 SPI flash 或者 SD/MMC 存储卡的标准文件系统。
- *磨损均衡* 库实现了一个适用于 SPI NOR flash 的 flash 翻译层 (FTL)，用于 flash 中 FAT 分区的容器。

备注： 建议使用高层次的 API（`esp_partition` 或者文件系统）而非低层次驱动 API 去访问 SPI NOR flash。

由于 NOR flash 和乐鑫硬件的一些限制，访问主 flash 会影响各个系统的性能。关于这些限制的更多信息，参见*SPI flash API*。

2.8.1 FAT 文件系统

ESP-IDF 使用 FatFs 库来实现 FAT 文件系统。FatFs 库位于 `fatfs` 组件中，支持直接使用，也可以借助 C 标准库和 POSIX API 通过 VFS（虚拟文件系统）使用 FatFs 库的大多数功能。

此外，我们对 FatFs 库进行了扩展，新增了支持可插拔磁盘 I/O 调度层，从而允许在运行时将 FatFs 驱动映射到物理磁盘。

FatFs 与 VFS 配合使用

头文件 fatfs/vfs/esp_vfs_fat.h 定义了连接 FatFs 和 VFS 的函数。

函数 `esp_vfs_fat_register()` 分配一个 `FATFS` 结构，并在 VFS 中注册特定路径前缀。如果文件路径以此前缀开头，则对此文件的后续操作将转至 FatFs API。

函数*`esp_vfs_fat_unregister_path()`* 删除在 VFS 中的注册，并释放 `FATFS` 结构。

多数应用程序在使用 `esp_vfs_fat_` 函数时，采用如下步骤：

1. **调用 `esp_vfs_fat_register()`，指定：**
   - 挂载文件系统的路径前缀（例如，`"/sdcard"` 或 `"/spiflash"`）
   - FatFs 驱动编号
   - 一个用于接收指向 `FATFS` 结构指针的变量
2. 调用*`ff_diskio_register()`*，为步骤 1 中的驱动编号注册磁盘 I/O 驱动；
3. 如需使用与传递到 `esp_vfs_fat_register()` 相同的驱动编号挂载文件系统，可调用 FatFs 函数 `f_mount()`。如果目标逻辑驱动上不存在该文件系统，`f_mount()` 将调用失败并报告 `FR_NO_FILESYSTEM` 错误。此时，应首先调用 `f_mkfs()`，在驱动上创建新的 FatFS 结构体，然后重新调用 `f_mount()`。注意，应在上述步骤之前调用 `f_fdisk()` 对 SD 卡进行分区。请参考 FatFs 文档，查看更多信息；
4. 调用 C 标准库和 POSIX API 对路径中带有步骤 1 中所述前缀的文件（例如，`"/sdcard/hello.txt"`）执行打开、读取、写入、擦除、复制等操作。文件系统默认使用 8.3 文件名 格式 (SFN)。如需使用长文件名 (LFN)，启用*CONFIG_FATFS_LONG_FILENAMES* 选项。请参考 FatFs 文件系统，查看更多信息；
5. 可以直接调用 FatFs 库函数，但需要使用没有 VFS 前缀的路径，如 `"/hello.txt"`；
6. 关闭所有打开的文件；
7. 调用 FatFs 函数 `f_mount()` 并使用 NULL `FATFS*` 参数，为与上述编号相同的驱动卸载文件系统；
8. 调用 FatFs 函数*`ff_diskio_register()`* 并使用 NULL `ff_diskio_impl_t*` 参数和相同的驱动编号，来释放注册的磁盘 I/O 驱动；
9. 调用*`esp_vfs_fat_unregister_path()`* 并使用文件系统挂载的路径将 FatFs 从 VFS 中移除，并释放步骤 1 中分配的 `FATFS` 结构。

便捷函数*`esp_vfs_fat_sdmmc_mount()`*、*`esp_vfs_fat_sdspi_mount()`* 和*`esp_vfs_fat_sdcard_unmount()`* 对上述步骤进行了封装，并加入了对 SD 卡初始化的处理。我们将在下一章节详细介绍以上函数。

备注： FAT 文件系统不支持硬链接，因此调用 `link()` 后会复制文件内容（仅适用于 FatFs 卷上的文件）。

FatFs 与 VFS 和 SD 卡配合使用

头文件 fatfs/vfs/esp_vfs_fat.h 定义了便捷函数*`esp_vfs_fat_sdmmc_mount()`*、*`esp_vfs_fat_sdspi_mount()`* 和*`esp_vfs_fat_sdcard_unmount()`*。这些函数分别执行上一章节的步骤 1-3 和步骤 7-9，并初始化 SD 卡，但仅提供有限的错误处理功能。我们鼓励开发人员查看源代码，将更多高级功能集成到产品应用中。

便捷函数*`esp_vfs_fat_sdmmc_unmount()`* 用于卸载文件系统并释放从*`esp_vfs_fat_sdmmc_mount()`* 函数获取的资源。

FatFs 与 VFS 配合使用（只读模式下）

头文件 fatfs/vfs/esp_vfs_fat.h 也定义了两个便捷函数*esp_vfs_fat_spiflash_mount_ro()*和*esp_vfs_fat_spiflash_unmount_ro()*。上述两个函数分别对 FAT 只读分区执行步骤 1-3 和步骤 7-9。有些数据分区仅在工厂配置时写入一次，之后在整个硬件生命周期内都不会再有任何改动。利用上述两个函数处理这种数据分区非常方便。

配置选项

FatFs 组件有以下配置选项：

- *CONFIG_FATFS_USE_FASTSEEK* - 如果启用该选项，POSIX `lseek()` 函数将以更快的速度执行。快速查找不适用于编辑模式下的文件，所以，使用快速查找时，应在只读模式下打开（或者关闭然后重新打开）文件。
- *CONFIG_FATFS_IMMEDIATE_FSYNC* - 如果启用该选项，FatFs 将在每次调用 `write()`、`pwrite()`、`link()`、`truncate()` 和 `ftruncate()` 函数后，自动调用 `f_sync()` 以同步最近的文件改动。该功能可提高文件系统中文件的一致性和文件大小报告的准确性，但由于需要频繁进行磁盘操作，性能将会受到影响。
- *CONFIG_FATFS_LINK_LOCK* - 如果启用该选项，可保证 API 的线程安全，但如果应用程序需要快速频繁地进行小文件操作（例如将日志记录到文件），则可能有必要禁用该选项。请注意，如果禁用该选项，调用 `link()` 后的复制操作将是非原子的，此时如果在不同任务中对同一卷上的大文件调用 `link()`，则无法确保线程安全。

FatFs 磁盘 I/O 层

我们对 FatFs API 函数进行了扩展，实现了运行期间注册磁盘 I/O 驱动。

上述 API 为 SD/MMC 卡提供了磁盘 I/O 函数实现方式，可使用*ff_diskio_register_sdmmc()* 函数注册指定的 FatFs 驱动编号。

void **ff_diskio_register**(BYTE pdrv, const *ff_diskio_impl_t* *discio_impl)

Register or unregister diskio driver for given drive number.

When FATFS library calls one of disk_xxx functions for driver number pdrv, corresponding function in discio_impl for given pdrv will be called.

参数

- **pdrv** -- drive number
- **discio_impl** -- pointer to *ff_diskio_impl_t* structure with diskio functions or NULL to unregister and free previously registered drive

struct **ff_diskio_impl_t**

Structure of pointers to disk IO driver functions.

See FatFs documentation for details about these functions

Public Members

DSTATUS (***init**)(unsigned char pdrv)

disk initialization function

DSTATUS (***status**)(unsigned char pdrv)

disk status check function

DRESULT (***read**)(unsigned char pdrv, unsigned char *buff, uint32_t sector, unsigned count)

sector read function

DRESULT (***write**)(unsigned char pdrv, const unsigned char *buff, uint32_t sector, unsigned count)

sector write function

DRESULT (***ioctl**)(unsigned char pdrv, unsigned char cmd, void *buff)

function to get info about disk and do some misc operations

void **ff_diskio_register_sdmmc**(unsigned char pdrv, sdmmc_card_t *card)

Register SD/MMC diskio driver

参数

- **pdrv** -- drive number
- **card** -- pointer to sdmmc_card_t structure describing a card; card should be initialized before calling f_mount.

esp_err_t **ff_diskio_register_wl_partition**(unsigned char pdrv, *wl_handle_t* flash_handle)

Register spi flash partition

参数

- **pdrv** -- drive number
- **flash_handle** -- handle of the wear levelling partition.

esp_err_t **ff_diskio_register_raw_partition**(unsigned char pdrv, const *esp_partition_t* *part_handle)

Register spi flash partition

参数

- **pdrv** -- drive number
- **part_handle** -- pointer to raw flash partition.

FatFs 分区生成器

我们为 FatFs (wl_fatfsgen.py) 提供了分区生成器，该生成器集成在构建系统中，方便用户在自己的项目中使用。

该生成器可以在主机上创建文件系统镜像，并用指定的主机文件夹内容对其进行填充。

该脚本是建立在分区生成器的基础上 (fatfsgen.py)，目前除了可以生成分区外，也可以初始化损耗均衡。

目前的最新版本支持短文件名、长文件名、FAT12 和 FAT16。长文件名的上限是 255 个字符，文件名中可以包含多个 `.` 字符以及其他字符，如 `+`、`,`、`;`、`=`、`[` and `]` 等。

如需进一步了解 FatFs 分区生成器或分区分析器，请查看 Generating and parsing FAT partition on host。

构建系统中使用 FatFs 分区生成器 通过调用 `fatfs_create_partition_image` 可以直接从 CMake 构建系统中调用 FatFs 分区生成器:

```
fatfs_create_spiflash_image(<partition> <base_dir> [FLASH_IN_PROJECT])
```

如果不希望在生成分区时使用损耗均衡，可以使用 `fatfs_create_rawflash_image`:

```
fatfs_create_rawflash_image(<partition> <base_dir> [FLASH_IN_PROJECT])
```

`fatfs_create_spiflash_image` 以及 `fatfs_create_rawflash_image` 必须从项目的 CMakeLists.txt 中调用。

如果决定使用 `fatfs_create_rawflash_image` （不支持损耗均衡），请注意它仅支持在设备中以只读模式安装。

该函数的参数如下：

1. partition - 分区的名称，需要在分区表中定义（如 storage/fatfsgen/partitions_example.csv）。

2. base_dir - 目录名称，该目录会被编码为 FatFs 分区，也可以选择将其被烧录进设备。但注意必须在分区表中指定合适的分区大小。
3. FLASH_IN_PROJECT 标志 - 可选参数，用户可以通过指定 FLASH_IN_PROJECT，选择在执行 idf.py flash -p <PORT> 时让分区镜像自动与应用程序二进制文件、分区表等一同烧录进设备。
4. PRESERVE_TIME 标志 - 可选参数，用户可强制让目标镜像保留源文件夹的时间戳。如果不保留，每个目标镜像的时间戳都将设置为 FATFS 默认初始时间（1980 年 1 月 1 日）。
5. ONE_FAT 标志 - 可选参数，支持生成仅包含单个 FAT（文件分配表）的 FATFS 卷。与包含两个 FAT 的 FATFS 卷相比，这样做可以拥有相对较大的可用空间（通过 FAT 使用的扇区数 * 扇区大小计算），但会增加 FATFS 卷损坏的风险。

例如:

```
fatfs_create_partition_image(my_fatfs_partition my_folder FLASH_IN_PROJECT)
```

没有指定 FLASH_IN_PROJECT 时也可以生成分区镜像，但是用户需要使用 esptool.py 或自定义的构建系统目标对其手动烧录。

相关示例请查看 storage/fatfsgen。

FatFs 分区分析器

我们为 FatFs 提供分区分析器 (fatfsparse.py)。

该分析器为 FatFs 分区生成器 (fatfsgen.py) 的逆向工具，可以根据 FatFs 镜像在主机上生成文件夹结构。

可以使用:

```
./fatfsparse.py [-h] [--wl-layer {detect,enabled,disabled}] [--verbose] fatfs_
↪image.img
```

生成文件夹结构之前，参数 --verbose 将根据 FatFs 镜像的引导扇区在终端打印详细信息。

FATFS 最小分区大小及限制

FATFS 组件支持 FAT12、FAT16 和 FAT32 文件系统类型。文件系统类型取决于卷上簇的数量（簇数通过数据扇区数量除以每簇包含的扇区数计算得出）。最小分区大小由分配给 FAT 表、根目录和数据簇的扇区数量决定。

- 对于 4096 字节的扇区，启用损耗均衡的 FAT 分区大小最小支持 32 KB。对于 512 字节的扇区，最小分区大小取决于损耗均衡的配置：性能模式下，最小支持 20 KB，安全模式下最小支持 28 KB（需要额外的 2 个扇区）。
- 启用了损耗均衡的分区会预留 4 个扇区用于损耗均衡操作。此外，FATFS 本身也会使用 4 个扇区，分别为 1 个保留扇区、1 个 FAT 扇区、1 个根目录扇区和 1 个数据扇区。
- 增加分区大小将分配更多的数据扇区，提供更大的存储空间。
- 对小于 528 KB 的分区，将分配 1 个根目录扇区；对于更大的分区，将分配 4 个根目录扇区。
- 默认会创建两个 FAT 扇区，因此分区大小会增加一个扇区来容纳这个额外的 FAT 扇区。如要启用单个 FAT 扇区，可以在 *struct esp_vfs_fat_mount_config_t* 中（参见 fatfs/vfs/esp_vfs_fat.h）设置 *use_one_fat* 选项。启用此选项后，最小分区大小可减少至 32 KB。
- 计算损耗均衡分区大小的一般公式为:

  ```
  partition_size = 损耗均衡扇区数 * FLASH_SEC_SIZE + FATFS 分区扇区数量 * FAT_
  ↪SEC_SIZE
  ```

  其中:
  - 损耗均衡扇区数固定为 4 个
  - FLASH_SEC_SIZE 为 4096 字节
  - FATFS 分区扇区包括：1 个保留扇区 + FAT 扇区 + 根目录扇区 + 数据扇区
  - FAT_SEC_SIZE 根据不同的配置，可以是 512 字节或 4096 字节
- 对于未启用损耗均衡、扇区大小为 512 字节的只读分区，最小分区大小可减少至 2 KB。

高级 API 参考

Header File

- components/fatfs/vfs/esp_vfs_fat.h
- This header file can be included with:

```
#include "esp_vfs_fat.h"
```

- This header file is a part of the API provided by the `fatfs` component. To declare that your component depends on `fatfs`, add the following to your CMakeLists.txt:

```
REQUIRES fatfs
```

or

```
PRIV_REQUIRES fatfs
```

Functions

esp_err_t **esp_vfs_fat_register_cfg**(const *esp_vfs_fat_conf_t* *conf, FATFS **out_fs)

Register FATFS with VFS component.

This function registers given FAT drive in VFS, at the specified base path. Input arguments are held in *esp_vfs_fat_conf_t* structure. If only one drive is used, fat_drive argument can be an empty string. Refer to FATFS library documentation on how to specify FAT drive. This function also allocates FATFS structure which should be used for f_mount call.

备注: This function doesn't mount the drive into FATFS, it just connects POSIX and C standard library IO function with FATFS. You need to mount desired drive into FATFS separately.

参数

- **conf** -- pointer to *esp_vfs_fat_conf_t* configuration structure
- **out_fs** -- **[out]** pointer to FATFS structure which can be used for FATFS f_mount call is returned via this argument.

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_register was already called
- ESP_ERR_NO_MEM if not enough memory or too many VFSes already registered

esp_err_t **esp_vfs_fat_unregister_path**(const char *base_path)

Un-register FATFS from VFS.

备注: FATFS structure returned by esp_vfs_fat_register is destroyed after this call. Make sure to call f_mount function to unmount it before calling esp_vfs_fat_unregister_ctx. Difference between this function and the one above is that this one will release the correct drive, while the one above will release the last registered one

---

参数 **base_path** -- path prefix where FATFS is registered. This is the same used when esp_vfs_fat_register was called

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if FATFS is not registered in VFS

esp_err_t **esp_vfs_fat_sdmmc_mount**(const char *base_path, const sdmmc_host_t *host_config, const void *slot_config, const *esp_vfs_fat_mount_config_t* *mount_config, sdmmc_card_t **out_card)

Convenience function to get FAT filesystem on SD card registered in VFS.

This is an all-in-one function which does the following:

- initializes SDMMC driver or SPI driver with configuration in host_config
- initializes SD card with configuration in slot_config
- mounts FAT partition on SD card using FATFS library, with configuration in mount_config
- registers FATFS library with VFS, with prefix given by base_prefix variable

This function is intended to make example code more compact. For real world applications, developers should implement the logic of probing SD card, locating and mounting partition, and registering FATFS in VFS, with proper error checking and handling of exceptional conditions.

备 注: Use this API to mount a card through SDSPI is deprecated. Please call `esp_vfs_fat_sdspi_mount()` instead for that case.

参数

- **`base_path`** -- path where partition should be registered (e.g. "/sdcard")
- **`host_config`** -- Pointer to structure describing SDMMC host. When using SDMMC peripheral, this structure can be initialized using SDMMC_HOST_DEFAULT() macro. When using SPI peripheral, this structure can be initialized using SDSPI_HOST_DEFAULT() macro.
- **`slot_config`** -- Pointer to structure with slot configuration. For SDMMC peripheral, pass a pointer to *sdmmc_slot_config_t* structure initialized using SDMMC_SLOT_CONFIG_DEFAULT.
- **`mount_config`** -- pointer to structure with extra parameters for mounting FATFS
- **`out_card`** -- **[out]** if not NULL, pointer to the card information structure will be returned via this argument

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount was already called
- ESP_ERR_NO_MEM if memory can not be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from SDMMC or SPI drivers, SDMMC protocol, or FATFS drivers

esp_err_t **`esp_vfs_fat_sdspi_mount`**(const char *base_path, const sdmmc_host_t *host_config_input, const *sdspi_device_config_t* *slot_config, const *esp_vfs_fat_mount_config_t* *mount_config, sdmmc_card_t **out_card)

Convenience function to get FAT filesystem on SD card registered in VFS.

This is an all-in-one function which does the following:

- initializes an SPI Master device based on the SPI Master driver with configuration in slot_config, and attach it to an initialized SPI bus.
- initializes SD card with configuration in host_config_input
- mounts FAT partition on SD card using FATFS library, with configuration in mount_config
- registers FATFS library with VFS, with prefix given by base_prefix variable

This function is intended to make example code more compact. For real world applications, developers should implement the logic of probing SD card, locating and mounting partition, and registering FATFS in VFS, with proper error checking and handling of exceptional conditions.

备注: This function try to attach the new SD SPI device to the bus specified in host_config. Make sure the SPI bus specified in `host_config->slot` have been initialized by `spi_bus_initialize()` before.

参数

- **`base_path`** -- path where partition should be registered (e.g. "/sdcard")

- **host_config_input** -- Pointer to structure describing SDMMC host. This structure can be initialized using SDSPI_HOST_DEFAULT() macro.
- **slot_config** -- Pointer to structure with slot configuration. For SPI peripheral, pass a pointer to *sdspi_device_config_t* structure initialized using SDSPI_DEVICE_CONFIG_DEFAULT().
- **mount_config** -- pointer to structure with extra parameters for mounting FATFS
- **out_card** -- **[out]** If not NULL, pointer to the card information structure will be returned via this argument. It is suggested to hold this handle and use it to unmount the card later if needed. Otherwise it's not suggested to use more than one card at the same time and unmount one of them in your application.

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount was already called
- ESP_ERR_NO_MEM if memory can not be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from SDMMC or SPI drivers, SDMMC protocol, or FATFS drivers

esp_err_t **esp_vfs_fat_sdmmc_unmount**(void)

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_sdmmc_mount.

Deprecated:

Use `esp_vfs_fat_sdcard_unmount()` instead.

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount hasn't been called

esp_err_t **esp_vfs_fat_sdcard_unmount**(const char *base_path, sdmmc_card_t *card)

Unmount an SD card from the FAT filesystem and release resources acquired using `esp_vfs_fat_sdmmc_mount()` or `esp_vfs_fat_sdspi_mount()`

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the card argument is unregistered
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount hasn't been called

esp_err_t **esp_vfs_fat_sdcard_format_cfg**(const char *base_path, sdmmc_card_t *card, *esp_vfs_fat_mount_config_t* *cfg)

Format FAT filesystem with given configuration.

备注: This API should be only called when the FAT is already mounted.

参数

- **base_path** -- Path where partition should be registered (e.g. "/sdcard")
- **card** -- Pointer to the card handle, which should be initialised by calling `esp_vfs_fat_sdspi_mount` first
- **cfg** -- Pointer to structure with extra parameters for formatting FATFS (only relevant fields are used). If NULL, the previous configuration will be used.

返回

- ESP_OK
- ESP_ERR_INVALID_STATE: FAT partition isn't mounted, call esp_vfs_fat_sdmmc_mount or esp_vfs_fat_sdspi_mount first
- ESP_ERR_NO_MEM: if memory can not be allocated
- ESP_FAIL: fail to format it, or fail to mount back

esp_err_t **esp_vfs_fat_sdcard_format**(const char *base_path, sdmmc_card_t *card)

Format FAT filesystem.

备注: This API should be only called when the FAT is already mounted.

参数

- **base_path** -- Path where partition should be registered (e.g. "/sdcard")
- **card** -- Pointer to the card handle, which should be initialised by calling `esp_vfs_fat_sdspi_mount` first

返回

- ESP_OK
- ESP_ERR_INVALID_STATE: FAT partition isn't mounted, call esp_vfs_fat_sdmmc_mount or esp_vfs_fat_sdspi_mount first
- ESP_ERR_NO_MEM: if memory can not be allocated
- ESP_FAIL: fail to format it, or fail to mount back

esp_err_t **esp_vfs_fat_spiflash_mount_rw_wl**(const char *base_path, const char *partition_label, const *esp_vfs_fat_mount_config_t* *mount_config, *wl_handle_t* *wl_handle)

Convenience function to initialize FAT filesystem in SPI flash and register it in VFS.

This is an all-in-one function which does the following:

- finds the partition with defined partition_label. Partition label should be configured in the partition table.
- initializes flash wear levelling library on top of the given partition
- mounts FAT partition using FATFS library on top of flash wear levelling library
- registers FATFS library with VFS, with prefix given by base_prefix variable

This function is intended to make example code more compact.

参数

- **base_path** -- path where FATFS partition should be mounted (e.g. "/spiflash")
- **partition_label** -- label of the partition which should be used
- **mount_config** -- pointer to structure with extra parameters for mounting FATFS
- **wl_handle** -- **[out]** wear levelling driver handle

返回

- ESP_OK on success
- ESP_ERR_NOT_FOUND if the partition table does not contain FATFS partition with given label
- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount_rw_wl was already called
- ESP_ERR_NO_MEM if memory can not be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from wear levelling library, SPI flash driver, or FATFS drivers

esp_err_t **esp_vfs_fat_spiflash_unmount_rw_wl**(const char *base_path, *wl_handle_t* wl_handle)

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_spiflash_mount_rw_wl.

参数

- **base_path** -- path where partition should be registered (e.g. "/spiflash")
- **wl_handle** -- wear levelling driver handle returned by esp_vfs_fat_spiflash_mount_rw_wl

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount_rw_wl hasn't been called

esp_err_t **esp_vfs_fat_spiflash_format_cfg_rw_wl**(const char *base_path, const char *partition_label, *esp_vfs_fat_mount_config_t* *cfg)

Format FAT filesystem with given configuration.

备注： This API can be called when the FAT is mounted / not mounted. If this API is called when the FAT isn't mounted (by calling esp_vfs_fat_spiflash_mount_rw_wl), this API will first mount the FAT then format it, then restore back to the original state.

参数

- **base_path** -- Path where partition should be registered (e.g. "/spiflash")
- **partition_label** -- Label of the partition which should be used
- **cfg** -- Pointer to structure with extra parameters for formatting FATFS (only relevant fields are used). If NULL and mounted the previous configuration will be used. If NULL and unmounted the default configuration will be used.

返回

- ESP_OK
- ESP_ERR_NO_MEM: if memory can not be allocated
- Other errors from esp_vfs_fat_spiflash_mount_rw_wl

esp_err_t **esp_vfs_fat_spiflash_format_rw_wl**(const char *base_path, const char *partition_label)

Format FAT filesystem.

备注： This API can be called when the FAT is mounted / not mounted. If this API is called when the FAT isn't mounted (by calling esp_vfs_fat_spiflash_mount_rw_wl), this API will first mount the FAT then format it, then restore back to the original state.

参数

- **base_path** -- Path where partition should be registered (e.g. "/spiflash")
- **partition_label** -- Label of the partition which should be used

返回

- ESP_OK
- ESP_ERR_NO_MEM: if memory can not be allocated
- Other errors from esp_vfs_fat_spiflash_mount_rw_wl

esp_err_t **esp_vfs_fat_spiflash_mount_ro**(const char *base_path, const char *partition_label, const *esp_vfs_fat_mount_config_t* *mount_config)

Convenience function to initialize read-only FAT filesystem and register it in VFS.

This is an all-in-one function which does the following:

- finds the partition with defined partition_label. Partition label should be configured in the partition table.
- mounts FAT partition using FATFS library
- registers FATFS library with VFS, with prefix given by base_prefix variable

---

备注： Wear levelling is not used when FAT is mounted in read-only mode using this function.

参数

- **base_path** -- path where FATFS partition should be mounted (e.g. "/spiflash")
- **partition_label** -- label of the partition which should be used
- **mount_config** -- pointer to structure with extra parameters for mounting FATFS

返回

- ESP_OK on success
- ESP_ERR_NOT_FOUND if the partition table does not contain FATFS partition with given label

- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount_ro was already called for the same partition
- ESP_ERR_NO_MEM if memory can not be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from SPI flash driver, or FATFS drivers

esp_err_t **esp_vfs_fat_spiflash_unmount_ro**(const char *base_path, const char *partition_label)

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_spiflash_mount_ro.

参数

- **base_path** -- path where partition should be registered (e.g. "/spiflash")
- **partition_label** -- label of partition to be unmounted

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount_ro hasn't been called

esp_err_t **esp_vfs_fat_info**(const char *base_path, uint64_t *out_total_bytes, uint64_t *out_free_bytes)

Get information for FATFS partition.

参数

- **base_path** -- Base path of the partition examined (e.g. "/spiflash")
- **out_total_bytes** -- **[out]** Size of the file system
- **out_free_bytes** -- **[out]** Free bytes available in the file system

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if partition not found
- ESP_FAIL if another FRESULT error (saved in errno)

esp_err_t **esp_vfs_fat_create_contiguous_file**(const char *base_path, const char *full_path, uint64_t size, bool alloc_now)

Create a file with contiguous space at given path.

备注: The file cannot exist before calling this function (or the file size has to be 0) For more information see documentation for `f_expand` from FATFS library

---

参数

- **base_path** -- Base path of the partition examined (e.g. "/spiflash")
- **full_path** -- Full path of the file (e.g. "/spiflash/ABC.TXT")
- **size** -- File size expanded to, number of bytes in size to prepare or allocate for the file
- **alloc_now** -- True == allocate space now, false == prepare to allocate –see `f_expand` from FATFS

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if invalid arguments (e.g. any of arguments are NULL or size lower or equal to 0)
- ESP_ERR_INVALID_STATE if partition not found
- ESP_FAIL if another FRESULT error (saved in errno)

esp_err_t **esp_vfs_fat_test_contiguous_file**(const char *base_path, const char *full_path, bool *is_contiguous)

Test if a file is contiguous in the FAT filesystem.

参数

- **base_path** -- Base path of the partition examined (e.g. "/spiflash")
- **full_path** -- Full path of the file (e.g. "/spiflash/ABC.TXT")
- **is_contiguous** -- **[out]** True == allocate space now, false == prepare to allocate –see `f_expand` from FATFS

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if invalid arguments (e.g. any of arguments are NULL)
- ESP_ERR_INVALID_STATE if partition not found
- ESP_FAIL if another FRESULT error (saved in errno)

Structures

struct **esp_vfs_fat_conf_t**

Configuration structure for esp_vfs_fat_register.

Public Members

const char ***base_path**

Path prefix where FATFS should be registered,

const char ***fat_drive**

FATFS drive specification; if only one drive is used, can be an empty string.

size_t **max_files**

Maximum number of files which can be open at the same time.

struct **esp_vfs_fat_mount_config_t**

Configuration arguments for esp_vfs_fat_sdmmc_mount and esp_vfs_fat_spiflash_mount_rw_wl functions.

Public Members

bool **format_if_mount_failed**

If FAT partition can not be mounted, and this parameter is true, create partition table and format the filesystem.

int **max_files**

Max number of open files.

size_t **allocation_unit_size**

If format_if_mount_failed is set, and mount fails, format the card with given allocation unit size. Must be a power of 2, between sector size and 128 * sector size. For SD cards, sector size is always 512 bytes. For wear_levelling, sector size is determined by CONFIG_WL_SECTOR_SIZE option.

Using larger allocation unit size will result in higher read/write performance and higher overhead when storing small files.

Setting this field to 0 will result in allocation unit set to the sector size.

bool **disk_status_check_enable**

Enables real ff_disk_status function implementation for SD cards (ff_sdmmc_status). Possibly slows down IO performance.

Try to enable if you need to handle situations when SD cards are not unmounted properly before physical removal or you are experiencing issues with SD cards.

Doesn't do anything for other memory storage media.

bool **use_one_fat**

Use 1 FAT (File Allocation Tables) instead of 2. This decreases reliability, but makes more space available (usually only one sector). Note that this option has effect only when the filesystem is formatted. When mounting an already-formatted partition, the actual number of FATs may be different.

Macros

VFS_FAT_MOUNT_DEFAULT_CONFIG()

Type Definitions

typedef *esp_vfs_fat_mount_config_t* **esp_vfs_fat_sdmmc_mount_config_t**

2.8.2 量产程序

介绍

这一程序主要用于量产时为每一设备创建工厂 NVS（非易失性存储器）分区镜像。NVS 分区镜像由 CSV（逗号分隔值）文件生成，文件中包含了用户提供的配置项及配置值。

注意，该程序仅创建用于量产的二进制镜像，您需要使用以下工具将镜像烧录到设备上：

- esptool.py
- Flash 下载工具 （仅适用于 Windows）。下载后解压，然后按照 doc 文件夹中的说明操作。
- 使用定制的生产工具直接烧录程序

准备工作

该程序依赖于 esp-idf 的 NVS 分区程序

- **操作系统要求：**
  - Linux、MacOS 或 Windows（标准版）
- **安装依赖包：**
  - Python

备注：

使用该程序之前，请确保：

- Python 路径已添加到 PATH 环境变量中；
- 已经安装 *requirement.txt* 中的软件包，*requirement.txt* 在 esp-idf 根目录下。

具体流程

CSV 配置文件

CSV 配置文件中包含设备待烧录的配置信息，定义了待烧录的配置项。

配置文件中数据格式如下（*REPEAT* 标签可选）：

```
name1,namespace,     <-- 第一个条目应该为 "namespace" 类型
key1,type1,encoding1
key2,type2,encoding2,REPEAT
name2,namespace,
key3,type3,encoding3
key4,type4,encoding4
```

备注： 文件第一行应始终为 namespace 条目。

每行应包含三个参数：key、type 和 encoding，并以逗号分隔。如果有 REPEAT 标签，则主 CSV 文件中所有设备此键值均相同。

有关各个参数的详细说明，请参阅 *NVS* 分区生成程序的 *README* 文件。

CSV 配置文件示例如下:

```
app,namespace,
firmware_key,data,hex2bin
serial_no,data,string,REPEAT
device_no,data,i32
```

备注:

请确保：

- 逗号',' 前后无空格；
- CSV 文件每行末尾无空格。

主 CSV 文件

主 CSV 文件中包含设备待烧录的详细信息，文件中每行均对应一个设备实体。

主 CSV 文件的数据格式如下:

```
key1,key2,key3,.....
value1,value2,value3,....
```

备注： 文件中键 (key) 名应始终置于文件首行。从配置文件中获取的键，在此文件中的排列顺序应与其在配置文件中的排列顺序相同。主 CSV 文件同时可以包含其它列（键），这些列将被视为元数据，而不会编译进最终二进制文件。

每行应包含相应键的键值 (value)，并用逗号隔开。如果某键带有 REPEAT 标签，则仅需在第二行（即第一个条目）输入对应的值，后面其他行为空。

参数描述如下：

value Data value

value 是与键对应的键值。

主 CSV 文件示例如下:

```
id,firmware_key,serial_no,device_no
1,1a2b3c4d5e6faabb,A1,101
2,1a2b3c4d5e6fccdd,,102
3,1a2b3c4d5e6feeff,,103
```

备注： 如果出现 *REPEAT* 标签，则会在相同目录下生成一个新的主 CSV 文件用作主输入文件，并在每行为带有 *REPEAT* 标签的键插入键值。

量产程序还会创建中间 CSV 文件，NVS 分区程序将使用此 CSV 文件作为输入，然后生成二进制文件。

中间 CSV 文件的格式如下:

```
key,type,encoding,value
key,namespace, ,
key1,type1,encoding1,value1
key2,type2,encoding2,value2
```

此步骤将为每一设备生成一个中间 CSV 文件。

运行量产程序

使用方法:

```
python mfg_gen.py [-h] {generate,generate-key} ...
```

可选参数:

序号	参数	描述
1	-h, --help	显示帮助信息并退出

命令:

运行 mfg_gen.py {command} -h 查看更多帮助信息

序号	参数	描述
1	generate	生成 NVS 分区
2	generate-key	生成加密密钥

为每个设备生成工厂镜像（默认）

使用方法:

```
python mfg_gen.py generate [-h] [--fileid FILEID] [--version {1,2}] [--keygen]
                           [--keyfile KEYFILE] [--inputkey INPUTKEY]
                           [--outdir OUTDIR]
                           conf values prefix size
```

位置参数:

参数	描述
conf	待解析的 CSV 配置文件路径
values	待解析的主 CSV 文件路径
prefix	每个输出文件名前缀的唯一名称
size	NVS 分区大小（以字节为单位，且为 4096 的整数倍）

可选参数:

参数	描述
-h, --help	显示帮助信息并退出
--fileid FILEID	每个文件名后缀的唯一文件标识符（主 CSV 文件中的任意键），默认为数值 1、2、3...
--version {1,2}	• 设置多页 Blob 版本。 • 版本 1 - 禁用多页 Blob； • 版本 2 - 启用多页 Blob； • 默认版本：版本 2
--keygen	生成 NVS 分区加密密钥
--inputkey INPUTKEY	内含 NVS 分区加密密钥的文件
--outdir OUTDIR	输出目录，用于存储创建的文件（默认当前目录）

请运行以下命令为每个设备生成工厂镜像，量产程序同时提供了一个 CSV 示例文件:

```
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_
↪singlepage_blob.csv Sample 0x3000
```

主 CSV 文件应在 `file` 类型下设置一个相对路径，相对于运行该程序的当前目录。

为每个设备生成工厂加密镜像

运行以下命令为每一设备生成工厂加密镜像，量产程序同时提供了一个 CSV 示例文件。

- 通过量产程序生成加密密钥来进行加密:

```
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_
↪singlepage_blob.csv Sample 0x3000 --keygen
```

备注: 创建的加密密钥格式为 `<outdir>/keys/keys-<prefix>-<fileid>.bin`。加密密钥存储于新建文件的 `keys/` 目录下，与 NVS 密钥分区结构兼容。更多信息请参考*NVS 密钥分区*。

- 提供加密密钥用作二进制输入文件来进行加密:

```
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_
↪singlepage_blob.csv Sample 0x3000 --inputkey keys/sample_keys.bin
```

仅生成加密密钥

使用方法:

```
python mfg_gen.py generate-key [-h] [--keyfile KEYFILE] [--outdir OUTDIR]
```

可选参数: +------------------+---+ | 参数 | 描述 | +------------------+---+ | -h, --help | 显示帮助信息并退出 | +------------------+---+ | --keyfile KEYFILE | 加密密钥文件的输出路径 | +------------------+---+ | --outdir OUTDIR | 输出目录，用于存储创建的文件（默认当前目录） | +------------------+---+

运行以下命令仅生成加密密钥:

```
python mfg_gen.py generate-key
```

备注: 创建的加密密钥格式为 `<outdir>/keys/keys-<timestamp>.bin`。时间戳格式为: `%m-%d_%H-%M`。如需自定义目标文件名，请使用 --keyfile 参数。

生成的加密密钥二进制文件还可以用于为每个设备的工厂镜像加密。

`fileid` 参数的默认值为 1、2、3...，与主 CSV 文件中的行一一对应，内含设备配置值。

运行量产程序时，将在指定的 `outdir` 目录下创建以下文件夹：

- `bin/` 存储生成的二进制文件
- `csv/` 存储生成的中间 CSV 文件
- `keys/` 存储加密密钥（创建工厂加密镜像时会用到）

2.8.3 非易失性存储库

简介

非易失性存储 (NVS) 库主要用于在 flash 中存储键值格式的数据。本文档将详细介绍 NVS 常用的一些概念。

底层存储 NVS 库通过调用*esp_partition* API 使用主 flash 的部分空间，即类型为 `data` 且子类型为 `nvs` 的所有分区。应用程序可调用*`nvs_open()`* API 选择使用带有 `nvs` 标签的分区，也可以通过调用*`nvs_open_from_partition()`* API 选择使用指定名称的任意分区。

NVS 库后续版本可能会增加其他存储器后端，来将数据保存至其他 flash 芯片（SPI 或 I2C 接口）、RTC 或 FRAM 中。

备注： 如果 NVS 分区被截断（例如，更改分区表布局时），则应擦除分区内容。可以使用 ESP-IDF 构建系统中的 `idf.py erase-flash` 命令擦除 flash 上的所有内容。

备注： NVS 最适合存储一些较小的数据，而非字符串或二进制大对象 (BLOB) 等较大的数据。如需存储较大的 BLOB 或者字符串，请考虑使用基于磨损均衡库的 FAT 文件系统。

键值对 NVS 的操作对象为键值对，其中键是 ASCII 字符串，当前支持的最大键长为 15 个字符。值可以为以下几种类型：

- 整数型：`uint8_t`、`int8_t`、`uint16_t`、`int16_t`、`uint32_t`、`int32_t`、`uint64_t` 和 `int64_t`；
- 以 0 结尾的字符串；
- 可变长度的二进制数据 (BLOB)

备注： 字符串值当前上限为 4000 字节，其中包括空终止符。BLOB 值上限为 508,000 字节或分区大小的 97.6% 减去 4000 字节，以较低值为准。

后续可能会增加对 `float` 和 `double` 等其他类型数据的支持。

键必须唯一。为现有的键写入新值时，会将旧的值及数据类型更新为写入操作指定的值和数据类型。

读取值时会执行数据类型检查。如果读取操作预期的数据类型与对应键的数据类型不匹配，则返回错误。

命名空间 为了减少不同组件之间键名的潜在冲突，NVS 将每个键值对分配给一个命名空间。命名空间的命名规则遵循键名的命名规则，例如，最多可占 15 个字符。此外，单个 NVS 分区最多只能容纳 254 个不同的命名空间。命名空间的名称在调用*`nvs_open()`* 或*`nvs_open_from_partition`* 中指定，调用后将返回一个不透明句柄，用于后续调用 `nvs_get_*`、`nvs_set_*` 和 `nvs_commit` 函数。这样，一个句柄关联一个命名空间，键名便不会与其他命名空间中相同键名冲突。请注意，不同 NVS 分区中具有相同名称的命名空间将被视为不同的命名空间。

NVS 迭代器 迭代器允许根据指定的分区名称、命名空间和数据类型轮询 NVS 中存储的键值对。

使用以下函数，可执行相关操作：

- `nvs_entry_find`：创建一个不透明句柄，用于后续调用 `nvs_entry_next` 和 `nvs_entry_info` 函数；
- `nvs_entry_next`：让迭代器指向下一个键值对；
- `nvs_entry_info`：返回每个键值对的信息。

总的来说，所有通过*`nvs_entry_find()`* 获得的迭代器（包括 `NULL` 迭代器）都必须使用*`nvs_release_iterator()`* 释放。

一般情况下，*`nvs_entry_find()`* 和*`nvs_entry_next()`* 会将给定的迭代器设置为 `NULL` 或为一个有效的迭代器。但如果出现参数错误（如返回 `ESP_ERR_NVS_NOT_FOUND`），给定的迭代器不会被修改。因此，在调用*`nvs_entry_find()`* 之前最好将迭代器初始化为 `NULL`，这样可以避免在释放迭代器之前进行复杂的错误检查。

安全性、篡改性及鲁棒性 NVS 与 ESP32-P4 flash 加密系统不直接兼容。然而，如果 NVS 加密与 ESP32-P4 flash 加密或 HMAC 外设一起使用，数据仍可以加密形式存储。详情请参考*NVS 加密*。

如果未启用 NVS 加密，任何对 flash 芯片有物理访问权限的用户都可以修改、擦除或添加键值对。NVS 加密启用后，如果不知道相应的 NVS 加密密钥，则无法修改或添加键值对并将其识别为有效键值对。但是，针对擦除操作没有相应的防篡改功能。

当 flash 处于不一致状态时，NVS 库会尝试恢复。在任何时间点关闭设备电源，然后重新打开电源，不会导致数据丢失；但如果关闭设备电源时正在写入新的键值对，这一键值对可能会丢失。该库还应该能够在 flash 中存在任何随机数据的情况下正常初始化。

NVS 加密

详情请参考*NVS 加密*。

NVS 分区生成程序

NVS 分区生成程序帮助生成 NVS 分区二进制文件，可使用烧录程序将二进制文件单独烧录至特定分区。烧录至分区上的键值对由 CSV 文件提供，详情请参考*NVS 分区生成程序*。

可以直接使用函数 `nvs_create_partition_image` 通过 CMake 创建分区二进制文件，无需手动调用 `nvs_partition_gen.py` 工具：

```
nvs_create_partition_image(<partition> <csv> [FLASH_IN_PROJECT] [DEPENDS  dep dep␣
↪dep ...])
```

位置参数:

参数	描述
`partition`	NVS 分区名
`csv`	解析的 CSV 文件路径

可选参数:

参数	描述
`FLASH_IN_PROJECT`	NVS 分区名
`DEPENDS`	指定命令依赖的文件

在没有指定 `FLASH_IN_PROJECT` 的情况下，也支持生成分区镜像，不过此时需要使用 `idf.py <partition>-flash` 手动进行烧录。举个例子，如果分区名为 `nvs`，则需使用的命令为 `idf.py nvs-flash`。

目前，仅支持从组件中的 `CMakeLists.txt` 文件调用 `nvs_create_partition_image`，且此选项仅适用于非加密分区。

应用示例

ESP-IDF storage 目录下提供了数个代码示例：

storage/nvs_rw_value

演示如何读取及写入 NVS 单个整数值。

此示例中的值表示 ESP32-P4 模组重启次数。NVS 中数据不会因为模组重启而丢失，因此只有将这一值存储于 NVS 中，才能起到重启次数计数器的作用。

该示例也演示了如何检测读取/写入操作是否成功，以及某个特定值是否在 NVS 中尚未初始化。诊断程序以纯文本形式提供，有助于追踪程序流程，及时发现问题。

storage/nvs_rw_blob

演示如何读取及写入 NVS 单个整数值和 BLOB（二进制大对象），并在 NVS 中存储这一数值，即便 ESP32-P4 模组重启也不会消失。

- value - 记录 ESP32-P4 模组软重启次数和硬重启次数。
- blob - 内含记录模组运行次数的表格。此表格将被从 NVS 读取至动态分配的 RAM 上。每次手动软重启后，表格内运行次数即增加一次，新加的运行次数被写入 NVS。下拉 GPIO0 即可手动软重启。

该示例也演示了如何执行诊断程序以检测读取/写入操作是否成功。

storage/nvs_rw_value_cxx

这个例子与 storage/nvs_rw_value 完全一样，只是使用了 C++ 的 NVS 句柄类。

内部实现

键值对日志 NVS 按顺序存储键值对，新的键值对添加在最后。因此，如需更新某一键值对，实际是在日志最后增加一对新的键值对，同时将旧的键值对标记为已擦除。

页面和条目 NVS 库在其操作中主要使用两个实体：页面和条目。页面是一个逻辑结构，用于存储部分的整体日志。逻辑页面对应 flash 的一个物理扇区，正在使用中的页面具有与之相关联的 序列号。序列号赋予了页面顺序，较高的序列号对应较晚创建的页面。页面有以下几种状态：

空或未初始化 页面对应的 flash 扇区为空白状态（所有字节均为 `0xff`）。此时，页面未存储任何数据且没有关联的序列号。

活跃状态 此时 flash 已完成初始化，页头部写入 flash，页面已具备有效序列号。页面中存在一些空条目，可写入数据。任意时刻，至多有一个页面处于活跃状态。

写满状态 flash 已写满键值对，状态不再改变。用户无法向写满状态下的页面写入新键值对，但仍可将一些键值对标记为已擦除。

擦除状态 未擦除的键值对将移至其他页面，以便擦除当前页面。这一状态仅为暂时性状态，即 API 调用返回时，页面应脱离这一状态。如果设备突然断电，下次开机时，设备将继续把未擦除的键值对移至其他页面，并继续擦除当前页面。

损坏状态 页头部包含无效数据，无法进一步解析该页面中的数据，因此之前写入该页面的所有条目均无法访问。相应的 flash 扇区并不会被立即擦除，而是与其他处于未初始化状态的扇区一起等待后续使用。这一状态可能对调试有用。

flash 扇区映射至逻辑页面并没有特定的顺序，NVS 库会检查存储在 flash 扇区的页面序列号，并根据序列号组织页面。

```
+--------+     +--------+     +--------+     +--------+
| Page 1 |     | Page 2 |     | Page 3 |     | Page 4 |
| Full   +---> | Full   +---> | Active |     | Empty  |   <- 状态
```

(下页继续)

(续上页)

```
| #11    |      | #12    |      | #14    |      |        |    <- 序列号
+---+----+      +----+---+      +----+---+      +---+----+
    |               |              |              |
    |               |              |              |
    |               |              |              |
+---v------+   +-----v----+   +------v---+   +------v---+
| Sector 3 |   | Sector 0 |   | Sector 2 |   | Sector 1 |    <- 物理扇区
+----------+   +----------+   +----------+   +----------+
```

页面结构 当前，我们假设 flash 扇区大小为 4096 字节，并且 ESP32-P4 flash 加密硬件在 32 字节块上运行。未来有可能引入一些编译时可配置项（可通过 menuconfig 进行配置），以适配具有不同扇区大小的 flash 芯片。但目前尚不清楚 SPI flash 驱动和 SPI flash cache 之类的系统组件是否支持其他扇区大小。

页面由头部、条目状态位图和条目三部分组成。为了实现与 ESP32-P4 flash 加密功能兼容，条目大小设置为 32 字节。如果键值为整数型，条目则保存一个键值对；如果键值为字符串或 BLOB 类型，则条目仅保存一个键值对的部分内容（更多信息详见条目结构描述）。

页面结构如下图所示，括号内数字表示该部分的大小（以字节为单位）。

```
+-----------+--------------+-------------+-------------------------+
| State (4) | Seq. no. (4) | version (1) | Unused (19) | CRC32 (4) |    页头部 (32)
+-----------+--------------+-------------+-------------------------+
|                 Entry state bitmap (32)                          |
+------------------------------------------------------------------+
|                         Entry 0 (32)                             |
+------------------------------------------------------------------+
|                         Entry 1 (32)                             |
+------------------------------------------------------------------+
/                                                                  /
/                                                                  /
+------------------------------------------------------------------+
|                         Entry 125 (32)                           |
+------------------------------------------------------------------+
```

头部和条目状态位图写入 flash 时不加密。如果启用了 ESP32-P4 flash 加密功能，则条目写入 flash 时将会加密。

通过将 0 写入某些位可以定义页面状态值，表示状态改变。因此，如果需要变更页面状态，并不一定要擦除页面，除非要将其变更为 擦除状态。

头部中的 `version` 字段反映了所用的 NVS 格式版本。为实现向后兼容，版本升级从 0xff 开始依次递减（例如，version-1 为 0xff，version-2 为 0xfe，以此类推）。

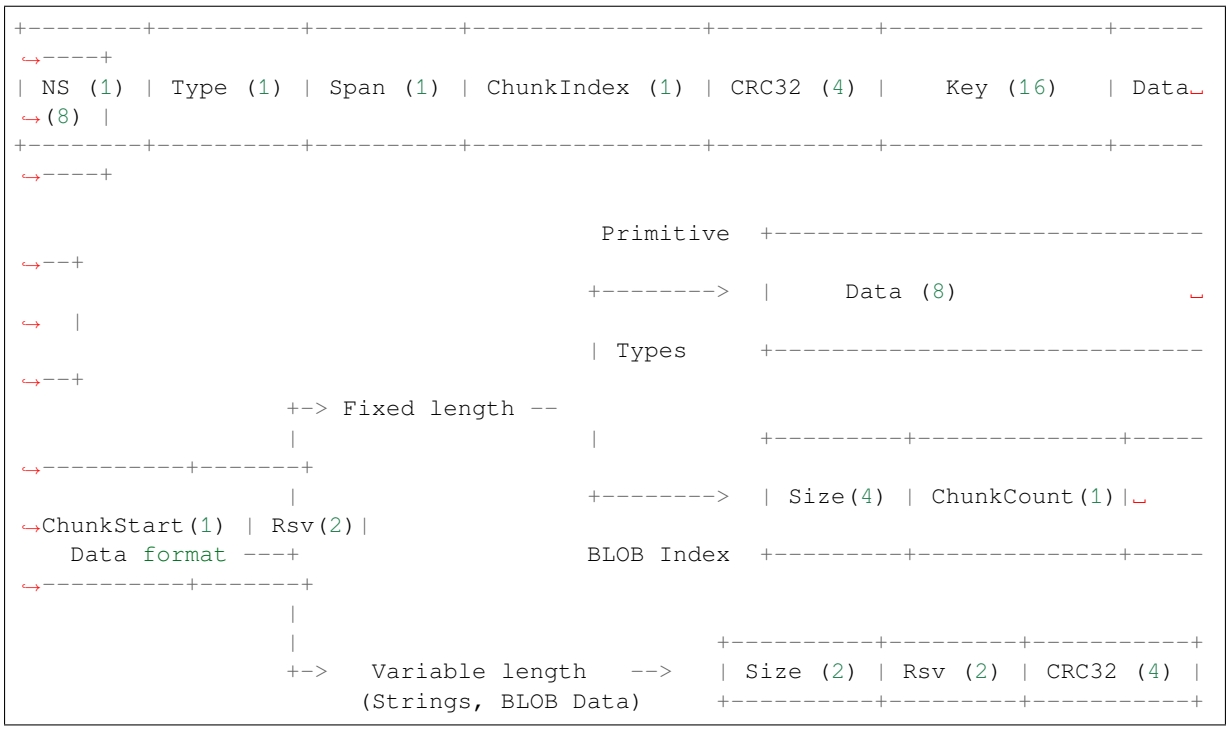
头部中 CRC32 值是由不包含状态值的条目计算所得（4 到 28 字节）。当前未使用的条目用 `0xff` 字节填充。

条目结构和条目状态位图的详细信息见下文描述。

条目和条目状态位图 每个条目可处于以下三种状态之一，每个状态在条目状态位图中用两位表示。位图中的最后四位 (256 - 2 * 126) 未使用。

空 (2'b11) 条目还未写入任何内容，处于未初始化状态（全部字节为 `0xff`）。
写入（2'b10） 一个键值对（或跨多个条目的键值对的部分内容）已写入条目中。
擦除（2'b00） 条目中的键值对已丢弃，条目内容不再解析。

条目结构 如果键值类型为基础类型，即 1 - 8 个字节长度的整数型，条目将保存一个键值对；如果键值类型为字符串或 BLOB 类型，条目将保存整个键值对的部分内容。另外，如果键值为字符串类型且跨多个条目，则键值所跨的所有条目均保存在同一页面。BLOB 则可以切分为多个块，实现跨多个页面。BLOB 索引是一个附加的固定长度元数据条目，用于追踪 BLOB 块。目前条目仍支持早期 BLOB 格式（可读取可修改），但这些 BLOB 一经修改，即以新格式储存至条目。



条目结构中各个字段含义如下：

命名空间 (NS, NameSpace) 该条目的命名空间索引，详细信息参见命名空间实现章节。

类型 (Type) 一个字节表示的值的数据类型，nvs_flash/include/nvs_handle.hpp 下的 `ItemType` 枚举了可能的类型。

跨度 (Span) 该键值对所用的条目数量。如果键值为整数型，条目数量即为 1。如果键值为字符串或 BLOB，则条目数量取决于值的长度。

块索引 (ChunkIndex) 用于存储 BLOB 类型数据块的索引。如果键值为其他数据类型，则此处索引应写入 `0xff`。

CRC32 对条目下所有字节进行校验后，所得的校验和（CRC32 字段不计算在内）。

键 (Key) 即以零结尾的 ASCII 字符串，字符串最长为 15 字节，不包含最后一个字节的零终止符。

数据 (Data) 如果键值类型为整数型，则数据字段仅包含键值。如果键值小于八个字节，使用 `0xff` 填充未使用的部分（右侧）。

如果键值类型为 BLOB 索引条目，则该字段的八个字节将保存以下数据块信息：

- **块大小** 整个 BLOB 数据的大小（以字节为单位）。该字段仅用于 BLOB 索引类型条目。
- **ChunkCount** 存储过程中 BLOB 分成的数据块总量。该字段仅用于 BLOB 索引类型条目。
- **ChunkStart** BLOB 第一个数据块的块索引，后续数据块索引依次递增，步长为 1。该字段仅用于 BLOB 索引类型条目。

如果键值类型为字符串或 BLOB 数据块，数据字段的这八个字节将保存该键值的一些附加信息，如下所示：

- **数据大小** 实际数据的大小（以字节为单位）。如果键值类型为字符串，此字段也应将零终止符包含在内。此字段仅用于字符串和 BLOB 类型条目。
- **CRC32** 数据所有字节的校验和，该字段仅用于字符串和 BLOB 类型条目。

可变长度值（字符串和 BLOB）写入后续条目，每个条目 32 字节。第一个条目的 `Span` 字段将指明使用了多少条目。

命名空间 如上所述，每个键值对属于一个命名空间。命名空间标识符（字符串）也作为键值对的键，存储在索引为 0 的命名空间中。与这些键对应的值就是这些命名空间的索引。

```
+-----------------------------------------+
| NS=0 Type=uint8_t Key="wifi" Value=1    |   Entry describing namespace "wifi"
+-----------------------------------------+
| NS=1 Type=uint32_t Key="channel" Value=6  |   Key "channel" in namespace "wifi"
+-----------------------------------------+
```

(下页继续)

(续上页)

```
| NS=0 Type=uint8_t Key="pwm" Value=2        |   Entry describing namespace "pwm"
+--------------------------------------------+
| NS=2 Type=uint16_t Key="channel" Value=20 |   Key "channel" in namespace "pwm"
+--------------------------------------------+
```

条目哈希列表 为了减少对 flash 执行的读操作次数，Page 类对象均设有一个列表，包含一对数据：条目索引和条目哈希值。该列表可大大提高检索速度，而无需迭代所有条目并逐个从 flash 中读取。`Page::findItem` 首先从哈希列表中检索条目哈希值，如果条目存在，则在页面内给出条目索引。由于哈希冲突，在哈希列表中检索条目哈希值可能会得到不同的条目，对 flash 中条目再次迭代可解决这一冲突。

哈希列表中每个节点均包含一个 24 位哈希值和 8 位条目索引。哈希值根据条目命名空间、键名和块索引由 CRC32 计算所得，计算结果保留 24 位。为减少将 32 位条目存储在链表中的开销，链表采用了数组的双向链表。每个数组占用 128 个字节，包含 29 个条目、两个链表指针和一个 32 位计数字段。因此，每页额外需要的 RAM 最少为 128 字节，最多为 640 字节。

API 参考

Header File

- components/nvs_flash/include/nvs_flash.h
- This header file can be included with:

```
#include "nvs_flash.h"
```

- This header file is a part of the API provided by the `nvs_flash` component. To declare that your component depends on `nvs_flash`, add the following to your CMakeLists.txt:

```
REQUIRES nvs_flash
```

or

```
PRIV_REQUIRES nvs_flash
```

Functions

esp_err_t **nvs_flash_init** (void)

Initialize the default NVS partition.

This API initialises the default NVS partition. The default NVS partition is the one that is labeled ”nvs” in the partition table.

When ”NVS_ENCRYPTION” is enabled in the menuconfig, this API enables the NVS encryption for the default NVS partition as follows

a. Read security configurations from the first NVS key partition listed in the partition table. (NVS key partition is any ”data” type partition which has the subtype value set to ”nvs_keys”)
b. If the NVS key partition obtained in the previous step is empty, generate and store new keys in that NVS key partition.
c. Internally call ”nvs_flash_secure_init()” with the security configurations obtained/generated in the previous steps.

Post initialization NVS read/write APIs remain the same irrespective of NVS encryption.

返回

- ESP_OK if storage was successfully initialized.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if no partition with label ”nvs” is found in the partition table
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures

- one of the error codes from the underlying flash storage driver
- error codes from nvs_flash_read_security_cfg API (when ”NVS_ENCRYPTION” is enabled).
- error codes from nvs_flash_generate_keys API (when ”NVS_ENCRYPTION” is enabled).
- error codes from nvs_flash_secure_init_partition API (when ”NVS_ENCRYPTION” is enabled) .

esp_err_t **nvs_flash_init_partition**(const char *partition_label)

Initialize NVS flash storage for the specified partition.

参数 **partition_label** -- **[in]** Label of the partition. Must be no longer than 16 characters.

返回

- ESP_OK if storage was successfully initialized.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if specified partition is not found in the partition table
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- one of the error codes from the underlying flash storage driver

esp_err_t **nvs_flash_init_partition_ptr**(const *esp_partition_t* *partition)

Initialize NVS flash storage for the partition specified by partition pointer.

参数 **partition** -- **[in]** pointer to a partition obtained by the ESP partition API.

返回

- ESP_OK if storage was successfully initialized
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_INVALID_ARG in case partition is NULL
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- one of the error codes from the underlying flash storage driver

esp_err_t **nvs_flash_deinit**(void)

Deinitialize NVS storage for the default NVS partition.

Default NVS partition is the partition with ”nvs” label in the partition table.

返回

- ESP_OK on success (storage was deinitialized)
- ESP_ERR_NVS_NOT_INITIALIZED if the storage was not initialized prior to this call

esp_err_t **nvs_flash_deinit_partition**(const char *partition_label)

Deinitialize NVS storage for the given NVS partition.

参数 **partition_label** -- **[in]** Label of the partition

返回

- ESP_OK on success
- ESP_ERR_NVS_NOT_INITIALIZED if the storage for given partition was not initialized prior to this call

esp_err_t **nvs_flash_erase**(void)

Erase the default NVS partition.

Erases all contents of the default NVS partition (one with label ”nvs”).

备注: If the partition is initialized, this function first de-initializes it. Afterwards, the partition has to be initialized again to be used.

返回

- ESP_OK on success
- ESP_ERR_NOT_FOUND if there is no NVS partition labeled ”nvs” in the partition table

- different error in case de-initialization fails (shouldn't happen)

esp_err_t **nvs_flash_erase_partition**(const char *part_name)

Erase specified NVS partition.

Erase all content of a specified NVS partition

备注: If the partition is initialized, this function first de-initializes it. Afterwards, the partition has to be initialized again to be used.

参数 **part_name** -- **[in]** Name (label) of the partition which should be erased

返回

- ESP_OK on success
- ESP_ERR_NOT_FOUND if there is no NVS partition with the specified name in the partition table
- different error in case de-initialization fails (shouldn't happen)

esp_err_t **nvs_flash_erase_partition_ptr**(const *esp_partition_t* *partition)

Erase custom partition.

Erase all content of specified custom partition.

备注: If the partition is initialized, this function first de-initializes it. Afterwards, the partition has to be initialized again to be used.

参数 **partition** -- **[in]** pointer to a partition obtained by the ESP partition API.

返回

- ESP_OK on success
- ESP_ERR_NOT_FOUND if there is no partition with the specified parameters in the partition table
- ESP_ERR_INVALID_ARG in case partition is NULL
- one of the error codes from the underlying flash storage driver

esp_err_t **nvs_flash_secure_init**(*nvs_sec_cfg_t* *cfg)

Initialize the default NVS partition.

This API initialises the default NVS partition. The default NVS partition is the one that is labeled "nvs" in the partition table.

参数 **cfg** -- **[in]** Security configuration (keys) to be used for NVS encryption/decryption. If cfg is NULL, no encryption is used.

返回

- ESP_OK if storage has been initialized successfully.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if no partition with label "nvs" is found in the partition table
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- one of the error codes from the underlying flash storage driver

esp_err_t **nvs_flash_secure_init_partition**(const char *partition_label, *nvs_sec_cfg_t* *cfg)

Initialize NVS flash storage for the specified partition.

参数

- **partition_label** -- **[in]** Label of the partition. Note that internally, a reference to passed value is kept and it should be accessible for future operations
- **cfg** -- **[in]** Security configuration (keys) to be used for NVS encryption/decryption. If cfg is null, no encryption/decryption is used.

返回

- ESP_OK if storage has been initialized successfully.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if specified partition is not found in the partition table
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- one of the error codes from the underlying flash storage driver

esp_err_t **nvs_flash_generate_keys**(const *esp_partition_t* *partition, *nvs_sec_cfg_t* *cfg)

Generate and store NVS keys in the provided esp partition.

参数

- **partition** -- **[in]** Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **cfg** -- **[out]** Pointer to nvs security configuration structure. Pointer must be non-NULL. Generated keys will be populated in this structure.

返回

- ESP_OK, if cfg was read successfully;
- ESP_ERR_INVALID_ARG, if partition or cfg is NULL;
- or error codes from esp_partition_write/erase APIs.

esp_err_t **nvs_flash_read_security_cfg**(const *esp_partition_t* *partition, *nvs_sec_cfg_t* *cfg)

Read NVS security configuration from a partition.

备注: Provided partition is assumed to be marked 'encrypted'.

参数

- **partition** -- **[in]** Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **cfg** -- **[out]** Pointer to nvs security configuration structure. Pointer must be non-NULL.

返回

- ESP_OK, if cfg was read successfully;
- ESP_ERR_INVALID_ARG, if partition or cfg is NULL
- ESP_ERR_NVS_KEYS_NOT_INITIALIZED, if the partition is not yet written with keys.
- ESP_ERR_NVS_CORRUPT_KEY_PART, if the partition containing keys is found to be corrupt
- or error codes from esp_partition_read API.

esp_err_t **nvs_flash_register_security_scheme**(*nvs_sec_scheme_t* *scheme_cfg)

Registers the given security scheme for NVS encryption The scheme registered with sec_scheme_id by this API be used as the default security scheme for the "nvs" partition. Users will have to call this API explicitly in their application.

参数 **scheme_cfg** -- **[in]** Pointer to the security scheme configuration structure that the user (or the nvs_key_provider) wants to register.

返回

- ESP_OK, if security scheme registration succeeds;
- ESP_ERR_INVALID_ARG, if scheme_cfg is NULL;
- ESP_FAIL, if security scheme registration fails

nvs_sec_scheme_t ***nvs_flash_get_default_security_scheme**(void)

Fetch the configuration structure for the default active security scheme for NVS encryption.

返回 Pointer to the default active security scheme configuration (NULL if no scheme is registered yet i.e. active)

esp_err_t **nvs_flash_generate_keys_v2**(*nvs_sec_scheme_t* *scheme_cfg, *nvs_sec_cfg_t* *cfg)

Generate (and store) the NVS keys using the specified key-protection scheme.

参数

- **scheme_cfg** -- **[in]** Security scheme specific configuration
- **cfg** -- **[out]** Security configuration (encryption keys)

返回

- ESP_OK, if cfg was populated successfully with generated encryption keys;
- ESP_ERR_INVALID_ARG, if scheme_cfg or cfg is NULL;
- ESP_FAIL, if the key generation process fails

esp_err_t **nvs_flash_read_security_cfg_v2** (*nvs_sec_scheme_t* *scheme_cfg, *nvs_sec_cfg_t* *cfg)

Read NVS security configuration set by the specified security scheme.

参数

- **scheme_cfg** -- **[in]** Security scheme specific configuration
- **cfg** -- **[out]** Security configuration (encryption keys)

返回

- ESP_OK, if cfg was read successfully;
- ESP_ERR_INVALID_ARG, if scheme_cfg or cfg is NULL;
- ESP_FAIL, if the key reading process fails

Structures

struct **nvs_sec_cfg_t**

Key for encryption and decryption.

Public Members

uint8_t **eky**[NVS_KEY_SIZE]

XTS encryption and decryption key

uint8_t **tky**[NVS_KEY_SIZE]

XTS tweak key

struct **nvs_sec_scheme_t**

NVS encryption: Security scheme configuration structure.

Public Members

int **scheme_id**

Security Scheme ID (E.g. HMAC)

void ***scheme_data**

Scheme-specific data (E.g. eFuse block for HMAC-based key generation)

nvs_flash_generate_keys_t **nvs_flash_key_gen**

Callback for the nvs_flash_key_gen implementation

nvs_flash_read_cfg_t **nvs_flash_read_cfg**

Callback for the nvs_flash_read_keys implementation

Macros

NVS_KEY_SIZE

Type Definitions

typedef *esp_err_t* (***nvs_flash_generate_keys_t***)(const void *scheme_data, *nvs_sec_cfg_t* *cfg)

Callback function prototype for generating the NVS encryption keys.

typedef *esp_err_t* (***nvs_flash_read_cfg_t***)(const void *scheme_data, *nvs_sec_cfg_t* *cfg)

Callback function prototype for reading the NVS encryption keys.

Header File

- components/nvs_flash/include/nvs.h
- This header file can be included with:

```
#include "nvs.h"
```

- This header file is a part of the API provided by the `nvs_flash` component. To declare that your component depends on `nvs_flash`, add the following to your CMakeLists.txt:

```
REQUIRES nvs_flash
```

or

```
PRIV_REQUIRES nvs_flash
```

Functions

esp_err_t **nvs_set_i8** (*nvs_handle_t* handle, const char *key, int8_t value)

set int8_t value for given key

Set value for the key, given its name. Note that the actual storage will not be updated until `nvs_commit` is called. Regardless whether key-value pair is created or updated, function always requires at least one nvs available entry. See `nvs_get_stats` . After create type of operation, the number of available entries is decreased by one. After update type of operation, the number of available entries remains the same.

参数

- **handle** -- **[in]** Handle obtained from nvs_open function. Handles that were opened read only cannot be used.
- **key** -- **[in]** Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.
- **value** -- **[in]** The value to set.

返回

- ESP_OK if value was set successfully
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if storage handle was opened as read only
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is not enough space in the underlying storage to save the value
- ESP_ERR_NVS_REMOVE_FAILED if the value wasn't updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn't fail again.

esp_err_t **nvs_set_u8** (*nvs_handle_t* handle, const char *key, uint8_t value)

set uint8_t value for given key

This function is the same as `nvs_set_i8` except for the data type.

esp_err_t **nvs_set_i16** (*nvs_handle_t* handle, const char *key, int16_t value)

set int16_t value for given key

This function is the same as `nvs_set_i8` except for the data type.

esp_err_t **nvs_set_u16** (*nvs_handle_t* handle, const char *key, uint16_t value)

set uint16_t value for given key

This function is the same as `nvs_set_i8` except for the data type.

esp_err_t **nvs_set_i32** (*nvs_handle_t* handle, const char *key, int32_t value)

set int32_t value for given key

This function is the same as `nvs_set_i8` except for the data type.

esp_err_t **nvs_set_u32** (*nvs_handle_t* handle, const char *key, uint32_t value)

set uint32_t value for given key

This function is the same as `nvs_set_i8` except for the data type.

esp_err_t **nvs_set_i64** (*nvs_handle_t* handle, const char *key, int64_t value)

set int64_t value for given key

This function is the same as `nvs_set_i8` except for the data type.

esp_err_t **nvs_set_u64** (*nvs_handle_t* handle, const char *key, uint64_t value)

set uint64_t value for given key

This function is the same as `nvs_set_i8` except for the data type.

esp_err_t **nvs_set_str** (*nvs_handle_t* handle, const char *key, const char *value)

set string for given key

Sets string value for the key. Function requires whole space for new data to be available as contiguous entries in same nvs page. Operation consumes 1 overhead entry and 1 entry per each 32 characters of new string including zero character to be set. In case of value update for existing key, entries occupied by the previous value and overhead entry are returned to the pool of available entries. Note that storage of long string values can fail due to fragmentation of nvs pages even if `available_entries` returned by `nvs_get_stats` suggests enough overall space available. Note that the underlying storage will not be updated until `nvs_commit` is called.

参数

- **handle** -- **[in]** Handle obtained from nvs_open function. Handles that were opened read only cannot be used.
- **key** -- **[in]** Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.
- **value** -- **[in]** The value to set. For strings, the maximum length (including null character) is 4000 bytes, if there is one complete page free for writing. This decreases, however, if the free space is fragmented.

返回

- ESP_OK if value was set successfully
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if storage handle was opened as read only
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is not enough space in the underlying storage to save the value
- ESP_ERR_NVS_REMOVE_FAILED if the value wasn't updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn't fail again.
- ESP_ERR_NVS_VALUE_TOO_LONG if the string value is too long

esp_err_t **nvs_get_i8** (*nvs_handle_t* handle, const char *key, int8_t *out_value)

get int8_t value for given key

These functions retrieve value for the key, given its name. If `key` does not exist, or the requested variable type doesn't match the type which was used when setting a value, an error is returned.

In case of any error, out_value is not modified.

`out_value` has to be a pointer to an already allocated variable of the given type.

```
// Example of using nvs_get_i32:
int32_t max_buffer_size = 4096; // default value
esp_err_t err = nvs_get_i32(my_handle, "max_buffer_size", &max_buffer_size);
assert(err == ESP_OK || err == ESP_ERR_NVS_NOT_FOUND);
// if ESP_ERR_NVS_NOT_FOUND was returned, max_buffer_size will still
// have its default value.
```

参数

- **handle** -- **[in]** Handle obtained from nvs_open function.
- **key** -- **[in]** Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.
- **out_value** -- Pointer to the output value. May be NULL for nvs_get_str and nvs_get_blob, in this case required length will be returned in length argument.

返回

- ESP_OK if the value was retrieved successfully
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_INVALID_LENGTH if length is not sufficient to store data

esp_err_t **nvs_get_u8** (*nvs_handle_t* handle, const char *key, uint8_t *out_value)

get uint8_t value for given key

This function is the same as `nvs_get_i8` except for the data type.

esp_err_t **nvs_get_i16** (*nvs_handle_t* handle, const char *key, int16_t *out_value)

get int16_t value for given key

This function is the same as `nvs_get_i8` except for the data type.

esp_err_t **nvs_get_u16** (*nvs_handle_t* handle, const char *key, uint16_t *out_value)

get uint16_t value for given key

This function is the same as `nvs_get_i8` except for the data type.

esp_err_t **nvs_get_i32** (*nvs_handle_t* handle, const char *key, int32_t *out_value)

get int32_t value for given key

This function is the same as `nvs_get_i8` except for the data type.

esp_err_t **nvs_get_u32** (*nvs_handle_t* handle, const char *key, uint32_t *out_value)

get uint32_t value for given key

This function is the same as `nvs_get_i8` except for the data type.

esp_err_t **nvs_get_i64** (*nvs_handle_t* handle, const char *key, int64_t *out_value)

get int64_t value for given key

This function is the same as `nvs_get_i8` except for the data type.

esp_err_t **nvs_get_u64** (*nvs_handle_t* handle, const char *key, uint64_t *out_value)

get uint64_t value for given key

This function is the same as `nvs_get_i8` except for the data type.

esp_err_t **nvs_get_str** (*nvs_handle_t* handle, const char *key, char *out_value, size_t *length)

get string value for given key

These functions retrieve the data of an entry, given its key. If key does not exist, or the requested variable type doesn't match the type which was used when setting a value, an error is returned.

In case of any error, out_value is not modified.

All functions expect out_value to be a pointer to an already allocated variable of the given type.

nvs_get_str and nvs_get_blob functions support WinAPI-style length queries. To get the size necessary to store the value, call nvs_get_str or nvs_get_blob with zero out_value and non-zero pointer to length. Variable pointed to by length argument will be set to the required length. For nvs_get_str, this length includes the zero terminator. When calling nvs_get_str and nvs_get_blob with non-zero out_value, length has to be non-zero and has to point to the length available in out_value. It is suggested that nvs_get/set_str is used for zero-terminated C strings, and nvs_get/set_blob used for arbitrary data structures.

```
// Example (without error checking) of using nvs_get_str to get a string into␣
↪dynamic array:
size_t required_size;
nvs_get_str(my_handle, "server_name", NULL, &required_size);
char* server_name = malloc(required_size);
nvs_get_str(my_handle, "server_name", server_name, &required_size);

// Example (without error checking) of using nvs_get_blob to get a binary data
into a static array:
uint8_t mac_addr[6];
size_t size = sizeof(mac_addr);
nvs_get_blob(my_handle, "dst_mac_addr", mac_addr, &size);
```

参数

- **handle** -- **[in]** Handle obtained from nvs_open function.
- **key** -- **[in]** Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.
- **out_value** -- **[out]** Pointer to the output value. May be NULL for nvs_get_str and nvs_get_blob, in this case required length will be returned in length argument.
- **length** -- **[inout]** A non-zero pointer to the variable holding the length of out_value. In case out_value a zero, will be set to the length required to hold the value. In case out_value is not zero, will be set to the actual length of the value written. For nvs_get_str this includes zero terminator.

返回

- ESP_OK if the value was retrieved successfully
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_INVALID_LENGTH if `length` is not sufficient to store data

esp_err_t **nvs_get_blob**(*nvs_handle_t* handle, const char *key, void *out_value, size_t *length)

get blob value for given key

This function behaves the same as `nvs_get_str`, except for the data type.

esp_err_t **nvs_open**(const char *namespace_name, *nvs_open_mode_t* open_mode, *nvs_handle_t* *out_handle)

Open non-volatile storage with a given namespace from the default NVS partition.

Multiple internal ESP-IDF and third party application modules can store their key-value pairs in the NVS module. In order to reduce possible conflicts on key names, each module can use its own namespace. The default NVS partition is the one that is labelled "nvs" in the partition table.

参数

- **namespace_name** -- **[in]** Namespace name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.
- **open_mode** -- **[in]** NVS_READWRITE or NVS_READONLY. If NVS_READONLY, will open a handle for reading only. All write requests will be rejected for this handle.
- **out_handle** -- **[out]** If successful (return code is zero), handle will be returned in this argument.

返回

- ESP_OK if storage handle was opened successfully
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized
- ESP_ERR_NVS_PART_NOT_FOUND if the partition with label "nvs" is not found
- ESP_ERR_NVS_NOT_FOUND id namespace doesn't exist yet and mode is NVS_READONLY
- ESP_ERR_NVS_INVALID_NAME if namespace name doesn't satisfy constraints
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is no space for a new entry or there are too many different namespaces (maximum allowed different namespaces: 254)
- ESP_ERR_NOT_ALLOWED if the NVS partition is read-only and mode is NVS_READWRITE
- ESP_ERR_INVALID_ARG if out_handle is equal to NULL
- other error codes from the underlying storage driver

esp_err_t **nvs_open_from_partition** (const char *part_name, const char *namespace_name, *nvs_open_mode_t* open_mode, *nvs_handle_t* *out_handle)

Open non-volatile storage with a given namespace from specified partition.

The behaviour is same as nvs_open() API. However this API can operate on a specified NVS partition instead of default NVS partition. Note that the specified partition must be registered with NVS using nvs_flash_init_partition() API.

参数

- **part_name** -- **[in]** Label (name) of the partition of interest for object read/write/erase
- **namespace_name** -- **[in]** Namespace name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.
- **open_mode** -- **[in]** NVS_READWRITE or NVS_READONLY. If NVS_READONLY, will open a handle for reading only. All write requests will be rejected for this handle.
- **out_handle** -- **[out]** If successful (return code is zero), handle will be returned in this argument.

返回

- ESP_OK if storage handle was opened successfully
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized
- ESP_ERR_NVS_PART_NOT_FOUND if the partition with specified name is not found
- ESP_ERR_NVS_NOT_FOUND id namespace doesn't exist yet and mode is NVS_READONLY
- ESP_ERR_NVS_INVALID_NAME if namespace name doesn't satisfy constraints
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is no space for a new entry or there are too many different namespaces (maximum allowed different namespaces: 254)
- ESP_ERR_NOT_ALLOWED if the NVS partition is read-only and mode is NVS_READWRITE
- ESP_ERR_INVALID_ARG if out_handle is equal to NULL
- other error codes from the underlying storage driver

esp_err_t **nvs_set_blob** (*nvs_handle_t* handle, const char *key, const void *value, size_t length)

set variable length binary value for given key

Sets variable length binary value for the key. Function uses 2 overhead and 1 entry per each 32 bytes of new data from the pool of available entries. See `nvs_get_stats` . In case of value update for existing key, space occupied by the existing value and 2 overhead entries are returned to the pool of available entries. Note that the underlying storage will not be updated until `nvs_commit` is called.

参数

- **handle** -- **[in]** Handle obtained from nvs_open function. Handles that were opened read only cannot be used.

- **key** -- **[in]** Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.
- **value** -- **[in]** The value to set.
- **length** -- **[in]** length of binary value to set, in bytes; Maximum length is 508000 bytes or (97.6% of the partition size - 4000) bytes whichever is lower.

返回

- ESP_OK if value was set successfully
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if storage handle was opened as read only
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is not enough space in the underlying storage to save the value
- ESP_ERR_NVS_REMOVE_FAILED if the value wasn't updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn't fail again.
- ESP_ERR_NVS_VALUE_TOO_LONG if the value is too long

esp_err_t **nvs_find_key**(*nvs_handle_t* handle, const char *key, *nvs_type_t* *out_type)

Lookup key-value pair with given key name.

Note that function may indicate both existence of the key as well as the data type of NVS entry if it is found.

参数

- **handle** -- **[in]** Storage handle obtained with nvs_open.
- **key** -- **[in]** Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.
- **out_type** -- **[out]** Pointer to the output variable populated with data type of NVS entry in case key was found. May be NULL, respective data type is then not provided.

返回

- ESP_OK if NVS entry for key provided was found
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- other error codes from the underlying storage driver

esp_err_t **nvs_erase_key**(*nvs_handle_t* handle, const char *key)

Erase key-value pair with given key name.

Note that actual storage may not be updated until nvs_commit function is called.

参数

- **handle** -- **[in]** Storage handle obtained with nvs_open. Handles that were opened read only cannot be used.
- **key** -- **[in]** Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn't be empty.

返回

- ESP_OK if erase operation was successful
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if handle was opened as read only
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
- other error codes from the underlying storage driver

esp_err_t **nvs_erase_all**(*nvs_handle_t* handle)

Erase all key-value pairs in a namespace.

Note that actual storage may not be updated until nvs_commit function is called.

参数 **handle** -- **[in]** Storage handle obtained with nvs_open. Handles that were opened read only cannot be used.

返回

- ESP_OK if erase operation was successful
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if handle was opened as read only
- other error codes from the underlying storage driver

esp_err_t **nvs_commit** (*nvs_handle_t* handle)

Write any pending changes to non-volatile storage.

After setting any values, nvs_commit() must be called to ensure changes are written to non-volatile storage. Individual implementations may write to storage at other times, but this is not guaranteed.

参数 **handle** -- **[in]** Storage handle obtained with nvs_open. Handles that were opened read only cannot be used.

返回

- ESP_OK if the changes have been written successfully
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- other error codes from the underlying storage driver

void **nvs_close** (*nvs_handle_t* handle)

Close the storage handle and free any allocated resources.

This function should be called for each handle opened with nvs_open once the handle is not in use any more. Closing the handle may not automatically write the changes to nonvolatile storage. This has to be done explicitly using nvs_commit function. Once this function is called on a handle, the handle should no longer be used.

参数 **handle** -- **[in]** Storage handle to close

esp_err_t **nvs_get_stats** (const char *part_name, *nvs_stats_t* *nvs_stats)

Fill structure *nvs_stats_t*. It provides info about memory used by NVS.

This function calculates the number of used entries, free entries, available entries, total entries and number of namespaces in partition.

```
// Example of nvs_get_stats() to get overview of actual statistics of data␣
↪entries :
nvs_stats_t nvs_stats;
nvs_get_stats(NULL, &nvs_stats);
printf("Count: UsedEntries = (%lu), FreeEntries = (%lu), AvailableEntries = (
↪%lu), AllEntries = (%lu)\n",
       nvs_stats.used_entries, nvs_stats.free_entries, nvs_stats.available_
↪entries, nvs_stats.total_entries);
```

参数

- **part_name** -- **[in]** Partition name NVS in the partition table. If pass a NULL than will use NVS_DEFAULT_PART_NAME (”nvs”).
- **nvs_stats** -- **[out]** Returns filled structure nvs_states_t. It provides info about used memory the partition.

返回

- ESP_OK if the changes have been written successfully. Return param nvs_stats will be filled.
- ESP_ERR_NVS_PART_NOT_FOUND if the partition with label ”name” is not found. Return param nvs_stats will be filled 0.
- ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized. Return param nvs_stats will be filled 0.
- ESP_ERR_INVALID_ARG if nvs_stats is equal to NULL.

- ESP_ERR_INVALID_STATE if there is page with the status of INVALID. Return param nvs_stats will be filled not with correct values because not all pages will be counted. Counting will be interrupted at the first INVALID page.

esp_err_t **nvs_get_used_entry_count** (*nvs_handle_t* handle, size_t *used_entries)

Calculate all entries in a namespace.

An entry represents the smallest storage unit in NVS. Strings and blobs may occupy more than one entry. Note that to find out the total number of entries occupied by the namespace, add one to the returned value used_entries (if err is equal to ESP_OK). Because the name space entry takes one entry.

```
// Example of nvs_get_used_entry_count() to get amount of all key-value pairs␣
↪in one namespace:
nvs_handle_t handle;
nvs_open("namespace1", NVS_READWRITE, &handle);
...
size_t used_entries;
size_t total_entries_namespace;
if(nvs_get_used_entry_count(handle, &used_entries) == ESP_OK){
// the total number of entries occupied by the namespace
    total_entries_namespace = used_entries + 1;
}
```

参数

- **handle** -- **[in]** Handle obtained from nvs_open function.
- **used_entries** -- **[out]** Returns amount of used entries from a namespace.

返回

- ESP_OK if the changes have been written successfully. Return param used_entries will be filled valid value.
- ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized. Return param used_entries will be filled 0.
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL. Return param used_entries will be filled 0.
- ESP_ERR_INVALID_ARG if used_entries is equal to NULL.
- Other error codes from the underlying storage driver. Return param used_entries will be filled 0.

esp_err_t **nvs_entry_find** (const char *part_name, const char *namespace_name, *nvs_type_t* type, *nvs_iterator_t* *output_iterator)

Create an iterator to enumerate NVS entries based on one or more parameters.

```
// Example of listing all the key-value pairs of any type under specified␣
↪partition and namespace
 nvs_iterator_t it = NULL;
 esp_err_t res = nvs_entry_find(<nvs_partition_name>, <namespace>, NVS_TYPE_
↪ANY, &it);
 while(res == ESP_OK) {
nvs_entry_info_t info;
     nvs_entry_info(it, &info); // Can omit error check if parameters are␣
↪guaranteed to be non-NULL
     printf("key '%s', type '%d' \n", info.key, info.type);
     res = nvs_entry_next(&it);
 }
 nvs_release_iterator(it);
```

参数

- **part_name** -- **[in]** Partition name
- **namespace_name** -- **[in]** Set this value if looking for entries with a specific namespace. Pass NULL otherwise.

- **type** -- **[in]** One of nvs_type_t values.
- **output_iterator** -- **[out]** Set to a valid iterator to enumerate all the entries found. Set to NULL if no entry for specified criteria was found. If any other error except ESP_ERR_INVALID_ARG occurs, `output_iterator` is NULL, too. If ESP_ERR_INVALID_ARG occurs, `output_iterator` is not changed. If a valid iterator is obtained through this function, it has to be released using `nvs_release_iterator` when not used any more, unless ESP_ERR_INVALID_ARG is returned.

返回

- ESP_OK if no internal error or programming error occurred.
- ESP_ERR_NVS_NOT_FOUND if no element of specified criteria has been found.
- ESP_ERR_NO_MEM if memory has been exhausted during allocation of internal structures.
- ESP_ERR_INVALID_ARG if any of the parameters is NULL. Note: don't release `output_iterator` in case ESP_ERR_INVALID_ARG has been returned

esp_err_t **nvs_entry_find_in_handle**(*nvs_handle_t* handle, *nvs_type_t* type, *nvs_iterator_t* *output_iterator)

Create an iterator to enumerate NVS entries based on a handle and type.

```
// Example of listing all the key-value pairs of any type under specified handle (which defines a partition and namespace)
 nvs_iterator_t it = NULL;
 esp_err_t res = nvs_entry_find_in_handle(<nvs_handle>, NVS_TYPE_ANY, &it);
 while(res == ESP_OK) {
nvs_entry_info_t info;
     nvs_entry_info(it, &info); // Can omit error check if parameters are guaranteed to be non-NULL
     printf("key '%s', type '%d' \n", info.key, info.type);
     res = nvs_entry_next(&it);
 }
 nvs_release_iterator(it);
```

参数

- **handle** -- **[in]** Handle obtained from nvs_open function.
- **type** -- **[in]** One of nvs_type_t values.
- **output_iterator** -- **[out]** Set to a valid iterator to enumerate all the entries found. Set to NULL if no entry for specified criteria was found. If any other error except ESP_ERR_INVALID_ARG occurs, `output_iterator` is NULL, too. If ESP_ERR_INVALID_ARG occurs, `output_iterator` is not changed. If a valid iterator is obtained through this function, it has to be released using `nvs_release_iterator` when not used any more, unless ESP_ERR_INVALID_ARG is returned.

返回

- ESP_OK if no internal error or programming error occurred.
- ESP_ERR_NVS_NOT_FOUND if no element of specified criteria has been found.
- ESP_ERR_NO_MEM if memory has been exhausted during allocation of internal structures.
- ESP_ERR_NVS_INVALID_HANDLE if unknown handle was specified.
- ESP_ERR_INVALID_ARG if output_iterator parameter is NULL. Note: don't release `output_iterator` in case ESP_ERR_INVALID_ARG has been returned

esp_err_t **nvs_entry_next**(*nvs_iterator_t* *iterator)

Advances the iterator to next item matching the iterator criteria.

Note that any copies of the iterator will be invalid after this call.

参数 **iterator** -- **[inout]** Iterator obtained from nvs_entry_find or nvs_entry_find_in_handle function. Must be non-NULL. If any error except ESP_ERR_INVALID_ARG occurs,

`iterator` is set to NULL. If ESP_ERR_INVALID_ARG occurs, `iterator` is not changed.

返回

- ESP_OK if no internal error or programming error occurred.
- ESP_ERR_NVS_NOT_FOUND if no next element matching the iterator criteria.
- ESP_ERR_INVALID_ARG if `iterator` is NULL.
- Possibly other errors in the future for internal programming or flash errors.

esp_err_t **nvs_entry_info**(const *nvs_iterator_t* iterator, *nvs_entry_info_t* *out_info)

Fills *nvs_entry_info_t* structure with information about entry pointed to by the iterator.

参数

- **iterator** -- **[in]** Iterator obtained from nvs_entry_find or nvs_entry_find_in_handle function. Must be non-NULL.
- **out_info** -- **[out]** Structure to which entry information is copied.

返回

- ESP_OK if all parameters are valid; current iterator data has been written to out_info
- ESP_ERR_INVALID_ARG if one of the parameters is NULL.

void **nvs_release_iterator**(*nvs_iterator_t* iterator)

Release iterator.

参数 **iterator** -- **[in]** Release iterator obtained from nvs_entry_find or nvs_entry_find_in_handle or nvs_entry_next function. NULL argument is allowed.

Structures

struct **nvs_entry_info_t**

information about entry obtained from nvs_entry_info function

Public Members

char **namespace_name**[NVS_NS_NAME_MAX_SIZE]

Namespace to which key-value belong

char **key**[NVS_KEY_NAME_MAX_SIZE]

Key of stored key-value pair

nvs_type_t **type**

Type of stored key-value pair

struct **nvs_stats_t**

备注: Info about storage space NVS.

Public Members

size_t **used_entries**

Number of used entries.

size_t **free_entries**

Number of free entries. It includes also reserved entries.

size_t **available_entries**

Number of entries available for data storage.

size_t **total_entries**

Number of all entries.

size_t **namespace_count**

Number of namespaces.

Macros

ESP_ERR_NVS_BASE

Starting number of error codes

ESP_ERR_NVS_NOT_INITIALIZED

The storage driver is not initialized

ESP_ERR_NVS_NOT_FOUND

A requested entry couldn't be found or namespace doesn't exist yet and mode is NVS_READONLY

ESP_ERR_NVS_TYPE_MISMATCH

The type of set or get operation doesn't match the type of value stored in NVS

ESP_ERR_NVS_READ_ONLY

Storage handle was opened as read only

ESP_ERR_NVS_NOT_ENOUGH_SPACE

There is not enough space in the underlying storage to save the value

ESP_ERR_NVS_INVALID_NAME

Namespace name doesn't satisfy constraints

ESP_ERR_NVS_INVALID_HANDLE

Handle has been closed or is NULL

ESP_ERR_NVS_REMOVE_FAILED

The value wasn't updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn't fail again.

ESP_ERR_NVS_KEY_TOO_LONG

Key name is too long

ESP_ERR_NVS_PAGE_FULL

Internal error; never returned by nvs API functions

ESP_ERR_NVS_INVALID_STATE

NVS is in an inconsistent state due to a previous error. Call nvs_flash_init and nvs_open again, then retry.

ESP_ERR_NVS_INVALID_LENGTH

String or blob length is not sufficient to store data

ESP_ERR_NVS_NO_FREE_PAGES

NVS partition doesn't contain any empty pages. This may happen if NVS partition was truncated. Erase the whole partition and call nvs_flash_init again.

ESP_ERR_NVS_VALUE_TOO_LONG

Value doesn't fit into the entry or string or blob length is longer than supported by the implementation

ESP_ERR_NVS_PART_NOT_FOUND

Partition with specified name is not found in the partition table

ESP_ERR_NVS_NEW_VERSION_FOUND

NVS partition contains data in new format and cannot be recognized by this version of code

ESP_ERR_NVS_XTS_ENCR_FAILED

XTS encryption failed while writing NVS entry

ESP_ERR_NVS_XTS_DECR_FAILED

XTS decryption failed while reading NVS entry

ESP_ERR_NVS_XTS_CFG_FAILED

XTS configuration setting failed

ESP_ERR_NVS_XTS_CFG_NOT_FOUND

XTS configuration not found

ESP_ERR_NVS_ENCR_NOT_SUPPORTED

NVS encryption is not supported in this version

ESP_ERR_NVS_KEYS_NOT_INITIALIZED

NVS key partition is uninitialized

ESP_ERR_NVS_CORRUPT_KEY_PART

NVS key partition is corrupt

ESP_ERR_NVS_WRONG_ENCRYPTION

NVS partition is marked as encrypted with generic flash encryption. This is forbidden since the NVS encryption works differently.

ESP_ERR_NVS_CONTENT_DIFFERS

Internal error; never returned by nvs API functions. NVS key is different in comparison

NVS_DEFAULT_PART_NAME

Default partition name of the NVS partition in the partition table

NVS_PART_NAME_MAX_SIZE

maximum length of partition name (excluding null terminator)

NVS_KEY_NAME_MAX_SIZE

Maximum length of NVS key name (including null terminator)

NVS_NS_NAME_MAX_SIZE

Maximum length of NVS namespace name (including null terminator)

NVS_GUARD_SYSVIEW_MACRO_EXPANSION_PUSH()

NVS_GUARD_SYSVIEW_MACRO_EXPANSION_POP()

Type Definitions

typedef uint32_t **nvs_handle_t**

Opaque pointer type representing non-volatile storage handle

typedef *nvs_handle_t* **nvs_handle**

typedef *nvs_open_mode_t* **nvs_open_mode**

typedef struct nvs_opaque_iterator_t ***nvs_iterator_t**

Opaque pointer type representing iterator to nvs entries

Enumerations

enum **nvs_open_mode_t**

Mode of opening the non-volatile storage.

Values:

enumerator **NVS_READONLY**

Read only

enumerator **NVS_READWRITE**

Read and write

enum **nvs_type_t**

Types of variables.

Values:

enumerator **NVS_TYPE_U8**

Type uint8_t

enumerator **NVS_TYPE_I8**

Type int8_t

enumerator **NVS_TYPE_U16**

Type uint16_t

enumerator **NVS_TYPE_I16**

Type int16_t

enumerator **NVS_TYPE_U32**

Type uint32_t

enumerator **NVS_TYPE_I32**

Type int32_t

enumerator **NVS_TYPE_U64**

Type uint64_t

enumerator **NVS_TYPE_I64**

Type int64_t

enumerator **NVS_TYPE_STR**

Type string

enumerator **NVS_TYPE_BLOB**

Type blob

enumerator **NVS_TYPE_ANY**

Must be last

2.8.4 NVS 加密

概述

本文档主要介绍 NVS 加密功能，这一功能有助于实现设备在 flash 中的安全存储。

存储在 NVS 分区中的数据可以用 XTS-AES 进行加密，与磁盘加密标准 IEEE P1619 中提到的加密方式类似。加密时，每个条目都被视作一个 `sector`，而条目的相对地址（相对于分区起始位置）作为 `sector-number` 输入加密算法。

根据要使用的具体方案，可以选择启用*CONFIG_NVS_ENCRYPTION* 和*CONFIG_NVS_SEC_KEY_PROTECTION_SCHEME* > `CONFIG_NVS_SEC_KEY_PROTECT_USING_FLASH_ENC` 或 `CONFIG_NVS_SEC_KEY_PROTECT_USING_HMAC` 实现 NVS 加密。

NVS 加密：基于 flash 加密的方案

在这个方案中，NVS 加密所需的密钥存储在另一个分区中，该分区用*flash 加密* 进行保护。因此，使用该方案时，必须先启用*flash 加密*。

启用*flash 加密* 时需同时启用 NVS 加密，因为 Wi-Fi 驱动程序会将凭据（如 SSID 和密码）储存在默认的 NVS 分区中。如已启用平台级加密，那么则需要同时启用 NVS 加密。

要用这一方案进行 NVS 加密，分区表中必须包含*NVS 密钥分区*。在分区表选项（`menuconfig > Partition Table`）中，有两个包含*NVS 密钥分区* 的分区表，可通过项目配置菜单（`idf.py menuconfig`）进行选择。要了解如何配置和使用 NVS 加密功能，请参考示例 security/flash_encryption。

NVS 密钥分区 应用如果要使用 NVS 加密（使用基于 flash 加密的方案）编译时，须使用类型为 `data` 和子类型为 `nvs_keys` 的密钥分区。该分区应被标记为 `encrypted` 且最小为 4 KB （最小分区大小）。参考*分区表* 了解详情。在分区表选项（`menuconfig > Partition Table`）中，有两个包含*NVS 密钥分区* 的额外分区表，可以直接用于 NVS 加密。分区的结构如下所示：

```
+-----------+-------------+-------------+----+
|              XTS encryption key (32)        |
+---------------------------------------------+
|              XTS tweak key (32)             |
+---------------------------------------------+
|                  CRC32 (4)                  |
+---------------------------------------------+
```

可以通过以下两种方式之一生成*NVS 密钥分区* 中的 XTS 加密密钥：

在 ESP32-P4 芯片上生成密钥

- 启用 NVS 加密时，可使用 API 函数*nvs_flash_init()* 来初始化加密的默认 NVS 分区。该 API 函数会内部生成 ESP 芯片的 XTS 加密密钥，并找到第一个*NVS 密钥分区*。
- 然后，该 API 函数使用 nvs_flash/include/nvs_flash.h 提供的*nvs_flash_generate_keys()* API 函数，自动生成 NVS 密钥并存储到该分区。只有当相应的密钥分区为空时，才会生成和存储新密钥。然后，就可以利用*nvs_flash_secure_init_partition()* 用此密钥分区来读取安全配置，以初始化自定义的加密 NVS 分区。
- API 函数*nvs_flash_secure_init()* 和*nvs_flash_secure_init_partition()* 不会内部生成密钥。如果要使用这两个 API 函数初始化加密的 NVS 分区，可以在启动后使用 `nvs_flash.h` 提供的*nvs_flash_generate_keys()* API 函数生成密钥，然后由该 API 函数将生成的密钥以加密形式写入密钥分区中。

备注： 请注意，在使用此方法启动应用程序前，必须完全擦除 `nvs_keys` 分区。否则，应用程序可能会生成*ESP_ERR_NVS_CORRUPT_KEY_PART* 错误代码，该代码假设 `nvs_keys` 分区不为空并且包含格式错误的数据。可以使用以下命令来实现：

```
parttool.py --port PORT --partition-table-file=PARTITION_TABLE_FILE --
↪partition-table-offset PARTITION_TABLE_OFFSET erase_partition --
↪partition-type=data --partition-subtype=nvs_keys
```

使用预生成的 NVS 密钥分区

如果*NVS 密钥分区* 中的密钥不是由应用程序生成，则需要使用预先生成的密钥分区。可以使用*NVS 分区生成程序* 生成包含 XTS 加密密钥的*NVS 密钥分区*。然后使用以下两个命令将预生成的密钥分区存储到 flash 上：

1. 构建并烧写分区表

```
idf.py partition-table partition-table-flash
```

2. 使用 parttool.py （参见*分区表* 中分区工具相关章节）将密钥存储在 flash 上的*NVS 密钥分区*中

```
parttool.py --port PORT --partition-table-offset PARTITION_TABLE_OFFSET␣
↪write_partition --partition-name="name of nvs_key partition" --input␣
↪NVS_KEY_PARTITION_FILE
```

备注： 如果设备是在 flash 加密开发模式下加密的，那么要更新 NVS 密钥分区就需要使用 parttool.py 来加密 NVS 密钥分区，并提供一个指向你构建目录中未加密分区表的指针 (build/partition_table)，因为设备上的分区表也是加密的。命令如下：

```
parttool.py --esptool-write-args encrypt --port PORT --partition-table-
↪file=PARTITION_TABLE_FILE --partition-table-offset PARTITION_TABLE_
↪OFFSET write_partition --partition-name="nvs_key 分区名称" --input NVS_
↪KEY_PARTITION_FILE
```

由于密钥分区被标记为 encrypted，且*flash 加密* 已启用，引导程序会在首次启动时使用 flash 加密密钥对此分区进行加密。

一个应用程序可以使用不同的密钥对不同的 NVS 分区进行加密，从而拥有多个密钥分区。应用程序应为加密或解密操作提供正确的密钥分区和密钥信息。

NVS 加密：基于 HMAC 外设的方案

此方案中，用于 NVS 加密的 XTS 密钥来自 eFuse 中编程的 HMAC 密钥，其目的是*esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_HMAC_UP*。由于加密密钥在运行时生成，不存储在 flash 中，因此这个功能不需要单独的*NVS 密钥分区*。

备注： 通过这个方案，**无需启用 flash 加密**就能在 ESP32-P4 上实现安全存储。

重要： 注意，此方案使用一个 eFuse 块来存储获取加密密钥所需的 HMAC 密钥。

- NVS 加密启用时后，可用 API 函数*nvs_flash_init()* 来初始化加密的默认 NVS 分区。该 API 函数首先检查*CONFIG_NVS_SEC_HMAC_EFUSE_KEY_ID* 处是否存在一个 HMAC 密钥。

备注： *CONFIG_NVS_SEC_HMAC_EFUSE_KEY_ID* 配置的有效范围为 0 (*hmac_key_id_t::HMAC_KEY0*) 到 5 (*hmac_key_id_t::HMAC_KEY5*)。默认情况下该配置为 6 (*hmac_key_id_t::HMAC_KEY_MAX*)，须在构建用户应用程序之前进行修改。

- 如果找不到密钥，会内部生成一个密钥，并储存在*CONFIG_NVS_SEC_HMAC_EFUSE_KEY_ID* 指定的 eFuse 块中。
- 如果找到用于*esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_HMAC_UP* 的密钥，该密钥也会用于 XTS 加密密钥的生成。
- 如果指定的 eFuse 块被*esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_HMAC_UP* 以外目的的密钥占用，则会引发错误。
- 然后，API *nvs_flash_init()* 使用 nvs_flash/include/nvs_flash.h 提供的*nvs_flash_generate_keys_v2()* API 函数，自动生成所需的 NVS 密钥。该密钥还可用于读取安全配置（参见*nvs_flash_read_security_cfg_v2()*）并通过*nvs_flash_secure_init_partition()* 初始化自定义的加密 NVS 分区。
- API 函数*nvs_flash_secure_init()* 和*nvs_flash_secure_init_partition()* 不会内部生成密钥。使用这些 API 函数初始化加密的 NVS 分区时，可在启动后用 API 函数*nvs_flash_generate_keys_v2()* 生成密钥，或使用*nvs_flash_read_security_cfg_v2()* 获取并填充 NVS 安全配置结构*nvs_sec_cfg_t*，将其输入到上述 API 中。

备注： 可以使用以下命令预先在 eFuse 中设置自己的 HMAC 密钥：

```
espefuse.py -p PORT burn_key <BLOCK_KEYN> <hmac_key_file.bin> HMAC_UP
```

加密读/写

NVS API 函数 nvs_get_* 或 nvs_set_* 也可用于读取和写入加密的 NVS 分区。

加密默认的 NVS 分区

- 要为默认 NVS 分区启用加密，无需额外的步骤。在启用*CONFIG_NVS_ENCRYPTION* 时，API 函数*nvs_flash_init()* 会根据使用的方案（由*CONFIG_NVS_SEC_KEY_PROTECTION_SCHEME* 设置）在内部执行一些额外步骤，为默认的 NVS 分区启用加密。

- 在基于 flash 加密的方案中，加密密钥由找到的第一个*NVS 密钥分区* 生成。在 HMAC 方案中，密钥由*CONFIG_NVS_SEC_HMAC_EFUSE_KEY_ID* 中烧录的 HMAC 密钥生成（参考 API 文档以了解更多详细信息）。

另外，还可使用 API 函数*nvs_flash_secure_init()* 为默认 NVS 分区启用加密。

加密自定义 NVS 分区

- 要为一个自定义的 NVS 分区启用加密，使用 API 函数*nvs_flash_secure_init_partition()* 代替*nvs_flash_init_partition()*。
- 使用 API 函数*nvs_flash_secure_init()* 和*nvs_flash_secure_init_partition()* 时，为了在启用加密的情况下执行 NVS 读/写操作，应用程序应遵守以下步骤：
  1. 填充 NVS 安全配置结构*nvs_sec_cfg_t*
     - 对基于 flash 加密的方案
       * 使用 API 函数 *esp_partition_find** 查找密钥分区和 NVS 数据分区。
       * 使用 API 函数*nvs_flash_read_security_cfg()* 或*nvs_flash_generate_keys()* 填充*nvs_sec_cfg_t* 结构体。
     - 对基于 HMAC 的方案
       * 用*nvs_sec_config_hmac_t* 为设置特定方案配置数据，并使用 API *nvs_sec_provider_register_hmac()* 注册此基于 HMAC 的方案。该 API 也将用于填充特定方案的句柄（参见*nvs_sec_scheme_t*）。
       * 使用 API 函数*nvs_flash_read_security_cfg_v2()* 或*nvs_flash_generate_keys_v2()* 填充*nvs_sec_cfg_t* 结构体。

```
nvs_sec_cfg_t cfg = {};
nvs_sec_scheme_t *sec_scheme_handle = NULL;

nvs_sec_config_hmac_t sec_scheme_cfg = {};
hmac_key_id_t hmac_key = HMAC_KEY0;
sec_scheme_cfg.hmac_key_id = hmac_key;

ret = nvs_sec_provider_register_hmac(&sec_scheme_cfg, &sec_scheme_
↪handle);
if (ret != ESP_OK) {
return ret;
}

ret = nvs_flash_read_security_cfg_v2(sec_scheme_handle, &cfg);
if (ret != ESP_OK) {
    if (ret == ESP_ERR_NVS_SEC_HMAC_KEY_NOT_FOUND) {
        ret = nvs_flash_generate_keys_v2(&sec_scheme_handle, &cfg);
        if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Failed to generate NVS encr-keys!");
            return ret;
        }
    }
    ESP_LOGE(TAG, "Failed to read NVS security cfg!");
    return ret;
}
```

  2. 使用 API 函数*nvs_flash_secure_init()* 或*nvs_flash_secure_init_partition()* 初始化 NVS flash 分区。
  3. 使用 API 函数*nvs_open()* 或*nvs_open_from_partition()* 打开一个命名空间。
  4. 使用 nvs_get_* 或 nvs_set_* 执行 NVS 读/写操作。
  5. 使用*nvs_flash_deinit()* 取消初始化 NVS 分区。

备注： 在采用基于 HMAC 的方案时，可以在不启用任何 NVS 加密的配置选项的情况下开始上述工作流：*CONFIG_NVS_ENCRYPTION*，*CONFIG_NVS_SEC_KEY_PROTECTION_SCHEME -> CONFIG_NVS_SEC_KEY_PROTECT_USING_HMAC* 和*CONFIG_NVS_SEC_HMAC_EFUSE_KEY_ID*，以使用*nvs_flash_secure_init()* API 加密默认分区及自定义的 NVS 分区。

NVS Security Provider

组件 nvs_sec_provider 存储了 NVS 加密方案的所有特定实现代码，并且适用于未来的方案。此组件充当 nvs_flash 组件处理加密密钥的接口。组件 nvs_sec_provider 有自己的配置菜单，选定的安全方案和相应设置基于这一菜单注册到 nvs_flash 组件。

该组件通过工厂函数注册了特殊的安全框架，可以实现出厂即用的安全方案。在该方案中，无需使用 API 来生成、读取加密密钥（如*nvs_sec_provider_register_hmac()*）。要了解 API 的使用，参考示例 security/nvs_encryption_hmac。

API 参考

Header File

- components/nvs_sec_provider/include/nvs_sec_provider.h
- This header file can be included with:

```
#include "nvs_sec_provider.h"
```

- This header file is a part of the API provided by the `nvs_sec_provider` component. To declare that your component depends on `nvs_sec_provider`, add the following to your CMakeLists.txt:

```
REQUIRES nvs_sec_provider
```

or

```
PRIV_REQUIRES nvs_sec_provider
```

Functions

esp_err_t **nvs_sec_provider_register_flash_enc**(const *nvs_sec_config_flash_enc_t* *sec_scheme_cfg, *nvs_sec_scheme_t* **sec_scheme_handle_out)

Register the Flash-Encryption based scheme for NVS Encryption.

参数

- **sec_scheme_cfg** -- **[in]** Security scheme specific configuration data
- **sec_scheme_handle_out** -- **[out]** Security scheme specific configuration handle

返回

- ESP_OK, if `sec_scheme_handle_out` was populated successfully with the scheme configuration;
- ESP_ERR_INVALID_ARG, if `scheme_cfg_hmac` is NULL;
- ESP_ERR_NO_MEM, No memory for the scheme-specific handle `sec_scheme_handle_out`
- ESP_ERR_NOT_FOUND, if no `nvs_keys` partition is found

esp_err_t **nvs_sec_provider_register_hmac**(const *nvs_sec_config_hmac_t* *sec_scheme_cfg, *nvs_sec_scheme_t* **sec_scheme_handle_out)

Register the HMAC-based scheme for NVS Encryption.

参数

- **sec_scheme_cfg** -- **[in]** Security scheme specific configuration data
- **sec_scheme_handle_out** -- **[out]** Security scheme specific configuration handle

返回

- ESP_OK, if `sec_scheme_handle_out` was populated successfully with the scheme configuration;
- ESP_ERR_INVALID_ARG, if `scheme_cfg_hmac` is NULL;
- ESP_ERR_NO_MEM, No memory for the scheme-specific handle `sec_scheme_handle_out`

esp_err_t **nvs_sec_provider_deregister**(*nvs_sec_scheme_t* *sec_scheme_handle)

Deregister the NVS encryption scheme registered with the given handle.

参数 **sec_scheme_handle** -- **[in]** Security scheme specific configuration handle

返回

- ESP_OK, if the scheme registered with `sec_scheme_handle` was deregistered successfully
- ESP_ERR_INVALID_ARG, if `sec_scheme_handle` is NULL;

Structures

struct **nvs_sec_config_flash_enc_t**

Flash encryption-based scheme specific configuration data.

Public Members

const *esp_partition_t* ***nvs_keys_part**

Partition of subtype `nvs_keys` holding the NVS encryption keys

struct **nvs_sec_config_hmac_t**

HMAC-based scheme specific configuration data.

Public Members

hmac_key_id_t **hmac_key_id**

HMAC Key ID used for generating the NVS encryption keys

Macros

ESP_ERR_NVS_SEC_BASE

Starting number of error codes

ESP_ERR_NVS_SEC_HMAC_KEY_NOT_FOUND

HMAC Key required to generate the NVS encryption keys not found

ESP_ERR_NVS_SEC_HMAC_KEY_BLK_ALREADY_USED

Provided eFuse block for HMAC key generation is already in use

ESP_ERR_NVS_SEC_HMAC_KEY_GENERATION_FAILED

Failed to generate/write the HMAC key to eFuse

ESP_ERR_NVS_SEC_HMAC_XTS_KEYS_DERIV_FAILED

Failed to derive the NVS encryption keys based on the HMAC-based scheme

NVS_SEC_PROVIDER_CFG_FLASH_ENC_DEFAULT()

Helper for populating the Flash encryption-based scheme specific configuration data.

NVS_SEC_PROVIDER_CFG_HMAC_DEFAULT()

Helper for populating the HMAC-based scheme specific configuration data.

Enumerations

enum **nvs_sec_scheme_id_t**

NVS Encryption Keys Protection Scheme.

Values:

enumerator **NVS_SEC_SCHEME_FLASH_ENC**

Protect NVS encryption keys using Flash Encryption

enumerator **NVS_SEC_SCHEME_HMAC**

Protect NVS encryption keys using HMAC peripheral

enumerator **NVS_SEC_SCHEME_MAX**

2.8.5 NVS 分区生成程序

介绍

NVS 分区生成程序 (nvs_flash/nvs_partition_generator/nvs_partition_gen.py) 根据 CSV 文件中的键值对生成二进制文件。该二进制文件与非易失性存储库 中定义的 NVS 结构兼容。

NVS 分区生成程序适合用于生成二进制数据 (blob)，其中包括设备生产时可从外部烧录的 ODM/OEM 数据。这也使得生产制造商在使用同一个应用固件的基础上，通过自定义参数，如序列号，为每个设备生成不同配置的二进制 NVS 分区。

准备工作

在加密模式下使用该程序，需安装下列软件包：

- `cryptography`

根目录下的 *requirements.txt* 包含必需 python 包，请预先安装。

CSV 文件格式 CSV 文件每行需包含四个参数，以逗号隔开。具体参数描述见下表：

序号	参数	描述	说明
1	Key	主键，应用程序可通过查询此键来获取数据。	
2	Type	支持 `file`、`data` 和 `namespace`。	
3	Encoding	支持 `u8`、`i8`、`u16`、`i16`、`u32`、`i32`、`u64`、`i64`、`string`、`hex2bin`、`base64` 和 `binary`。决定二进制 `bin` 文件中 value 被编码成的类型。`string` 和 `binary` 编码的区别在于，`string` 数据以 NULL 字符结尾，`binary` 数据则不是。	`file` 类型当前仅支持 `hex2bin`、`base64`、`string` 和 `binary` 编码。
4	Value	Data value	`namespace` 字段的 `encoding` 和 `value` 应为空。`namespace` 的 `encoding` 和 `value` 为固定值，不可设置。这些单元格中的所有值都会被忽视。

备注： CSV 文件的第一行应始终为列标题，不可设置。

此类 CSV 文件的 Dump 示例如下:

```
key,type,encoding,value      <-- 列标题
namespace_name,namespace,,   <-- 第一个条目为 "namespace"
key1,data,u8,1
key2,file,string,/path/to/file
```

备注:

请确保：

- 逗号',' 前后无空格;
- CSV 文件每行末尾无空格。

NVS 条目和命名空间 (namespace) 的关联

如 CSV 文件中出现命名空间条目，后续条目均会被视为该命名空间的一部分，直至找到下一个命名空间条目。找到新命名空间条目后，后续所有条目都会被视为新命名空间的一部分。

备注： CSV 文件中第一个条目应始终为 `namespace`。

支持多页 blob

默认情况下，二进制 blob 可跨多页，格式参考*条目结构* 章节。如需使用旧版格式，可在程序中禁用该功能。

支持加解密

NVS 分区生成程序还可使用 XTS-AES 加密生成二进制加密文件或对此类文件进行解密。更多信息详见*NVS 加密*。

运行程序

使用方法:

```
python nvs_partition_gen.py [-h] {generate,generate-key,encrypt,decrypt} ...
```

可选参数:

序号	参数	描述
1	`-h / --help`	显示帮助信息并退出

命令:

运行 `nvs_partition_gen.py {command} -h` 查看更多帮助信息

序号	参数	描述
1	`generate`	生成 NVS 分区
2	`generate-key`	生成加密密钥
3	`encrypt`	加密 NVS 分区
4	`decrypt`	解密 NVS 分区

生成 NVS 分区（默认模式）　使用方法:

```
python nvs_partition_gen.py generate [-h] [--version {1,2}] [--outdir OUTDIR]␣
↪input output size
```

位置参数:

参数	描述
input	待解析的 CSV 文件路径
output	NVS 二进制文件的输出路径
size	NVS 分区大小（以字节为单位，且为 4096 的整数倍）

可选参数:

参数	描述
-h / --help	显示帮助信息并退出
--version {1,2}	• 设置多页 blob 版本，默认为版本 2。 • 版本 1：禁用多页 blob； • 版本 2：启用多页 blob。
--outdir OUTDIR	输出目录，用于存储创建的文件。(默认当前目录)

运行如下命令创建 NVS 分区，该程序同时会提供 CSV 示例文件:

```
python nvs_partition_gen.py generate sample_singlepage_blob.csv sample.bin 0x3000
```

生成加密密钥分区　使用方法:

```
python nvs_partition_gen.py generate-key [-h] [--key_protect_hmac] [--kp_hmac_
↪keygen]
                                         [--kp_hmac_keyfile KP_HMAC_KEYFILE]␣
↪[--kp_hmac_inputkey KP_HMAC_INPUTKEY]
                                         [--keyfile KEYFILE] [--outdir OUTDIR]
```

可选参数:

参数	描述
-h / --help	显示帮助信息并退出
--keyfile KEYFILE	加密密钥分区文件的输出路径
--outdir OUTDIR	输出目录，用于存储创建的文件（默认当前目录）

可选参数（仅适用于 HMAC 方案）:

参数	描述
--key_protect_hmac	设置后使用基于 HMAC 的 NVS 加密密钥保护方案，否则使用基于 flash 加密的默认方案
--kp_hmac_keygen	为基于 HMAC 的加密方案生成 HMAC 密钥
--kp_hmac_keyfile KP_HMAC_KEYFILE	HMAC 密钥文件的输出路径
--kp_hmac_inputkey KP_HMAC_INPUTKEY	包含 HMAC 密钥的文件，用于生成 NVS 加密密钥

运行以下命令仅生成加密密钥分区:

```
python nvs_partition_gen.py generate-key
```

运行以下命令，为基于 HMAC 的方案生成加密密钥：

- 生成 HMAC 密钥和 NVS 加密密钥：

```
python nvs_partition_gen.py generate-key --key_protect_hmac --kp_hmac_keygen
```

备注： 上述命令生成 <outdir>/keys/keys-<timestamp>.bin 格式的加密密钥和 <outdir>/keys/hmac-keys-<timestamp>.bin 格式的 HMAC 密钥。

- 基于 HMAC 密钥生成 NVS 加密密钥：

```
python nvs_partition_gen.py generate-key --key_protect_hmac --kp_hmac_inputkey␣
↪testdata/sample_hmac_key.bin
```

备注： 可将自定义文件名作为参数提供给 HMAC 密钥和加密密钥。

生成 NVS 加密分区 使用方法：

```
python nvs_partition_gen.py encrypt [-h] [--version {1,2}] [--keygen]
                                    [--keyfile KEYFILE] [--inputkey INPUTKEY] [--
↪outdir OUTDIR]
                                    [--key_protect_hmac] [--kp_hmac_keygen]
                                    [--kp_hmac_keyfile KP_HMAC_KEYFILE] [--kp_hmac_
↪inputkey KP_HMAC_INPUTKEY]
                                    input output size
```

位置参数：

参数	描述
input	待解析的 CSV 文件路径
output	NVS 二进制文件的输出路径
size	NVS 分区大小（以字节为单位，且为 4096 的整数倍）

可选参数：

参数	描述
-h / --help	显示帮助信息并退出
--version {1,2}	• 设置多页 blob 版本，默认为版本 2。 • 版本 1：禁用多页 blob； • 版本 2：启用多页 blob。
--keygen	生成 NVS 分区加密密钥
--keyfile KEYFILE	密钥文件的输出路径
--inputkey INPUTKEY	内含 NVS 分区加密密钥的文件
--outdir OUTDIR	输出目录，用于存储创建的文件。(默认当前目录)

可选参数（仅适用于 HMAC 方案）：

参数	描述
--key_protect_hmac	设置后使用基于 HMAC 的 NVS 加密密钥保护方案，否则使用基于 flash 加密的默认方案
--kp_hmac_keygen	为基于 HMAC 的加密方案生成 HMAC 密钥
--kp_hmac_keyfile KP_HMAC_KEYFILE	HMAC 密钥文件的输出路径
--kp_hmac_inputkey KP_HMAC_INPUTKEY	包含 HMAC 密钥的文件，用于生成 NVS 加密密钥

运行以下命令加密 NVS 分区，该程序同时会提供一个 CSV 示例文件。

- 通过 NVS 分区生成程序生成加密密钥来加密:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin␣
↪0x3000 --keygen
```

备注: 创建的加密密钥格式为 <outdir>/keys/keys-<timestamp>.bin。

- 要使用基于 HMAC 的方案生成加密分区，可将上述命令与附加参数搭配使用。
  - 通过 NVS 分区生成程序生成加密密钥和 HMAC 密钥，从而进行加密:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.
↪bin 0x3000 --keygen --key_protect_hmac --kp_hmac_keygen
```

备注: 上述命令生成 <outdir>/keys/keys-<timestamp>.bin 格式的加密密钥和 <outdir>/keys/hmac-keys-<timestamp>.bin 格式的 HMAC 密钥。

  - 通过 NVS 分区生成程序使用用户提供的 HMAC 密钥生成加密密钥，从而进行加密:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.
↪bin 0x3000 --keygen --key_protect_hmac --kp_hmac_inputkey testdata/
↪sample_hmac_key.bin
```

备注: 可将自定义文件名作为参数提供给 HMAC 密钥和加密密钥。

- 通过 NVS 分区生成程序生成加密密钥，并将密钥存储于自定义的文件中:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin␣
↪0x3000 --keygen --keyfile sample_keys.bin
```

备注:

- 创建的加密密钥格式为 <outdir>/keys/sample_keys.bin。
- 加密密钥存储于新建文件的 keys/ 目录下，与 NVS 密钥分区结构兼容。更多信息请参考*NVS 密钥分区*。

- 将加密密钥用作二进制输入文件来进行加密:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin␣
↪0x3000 --inputkey sample_keys.bin
```

解密 NVS 分区 **使用方法**:

```
python nvs_partition_gen.py decrypt [-h] [--outdir OUTDIR] input key output
```

位置参数:

参数	描述
input	待解析的 NVS 加密分区文件路径
key	含有解密密钥的文件路径
output	已解密的二进制文件输出路径

可选参数:

参数	描述
-h / --help	显示帮助信息并退出
--outdir OUTDIR	输出目录，用于存储创建的文件（默认当前目录）

运行以下命令解密已加密的 NVS 分区:

```
python nvs_partition_gen.py decrypt sample_encr.bin sample_keys.bin sample_decr.bin
```

可以在命令中提供版本参数，选择格式版本号：

- 版本 1：禁用多页 blob
- 版本 2：启用多页 blob

版本 1：禁用多页 blob 如需禁用多页 blob，请按照如下命令将版本参数设置为 1，以此格式运行分区生成程序。该程序同时会提供一个 CSV 示例文件:

```
python nvs_partition_gen.py generate sample_singlepage_blob.csv sample.bin 0x3000 -
↪-version 1
```

版本 2：启用多页 blob 如需启用多页 blob，请按照如下命令将版本参数设置为 2，以此格式运行分区生成程序。该程序同时会提供一个 CSV 示例文件:

```
python nvs_partition_gen.py generate sample_multipage_blob.csv sample.bin 0x4000 --
↪version 2
```

备注:

- NVS 分区最小为 0x3000 字节。
- 将二进制文件烧录至设备时，请确保与应用的 sdkconfig 设置一致。

说明

- 分区生成程序不会对重复键进行检查，而将数据同时写入这两个重复键中。请注意不要使用同名的键；
- 新页面创建后，前一页的空白处不会再写入数据。CSV 文件中的字段须按次序排列以优化内存；
- 暂不支持 64 位数据类型。

2.8.6 NVS 分区解析程序

介绍

NVS 分区解析程序 nvs_flash/nvs_partition_tool/nvs_tool.py 加载并解析 NVS 存储分区，以便于调试和数据提取。该程序还支持完整性检查功能，可扫描分区中可能存在的错误。Blob 数据以 *base64* 格式进行编码。

加密分区

此程序不支持解密。如需解密 NVS 分区，请使用*NVS 分区生成程序*。该工具支持 NVS 分区加解密。

使用方法

该程序提供了 *-f* 或 *-format* 选项，对应两种不同的输出格式：

- *json* - 所有输出均以 JSON 格式打印。
- *text* - 输出以可读文本的格式打印，有以下输出格式可选。

针对 *text* 输出格式，该程序提供了 *-d* 或 *-dump* 选项，包含六种不同的输出方式：

- *all* (默认) - 打印所有带有元数据的条目。
- *written* - 只打印带有元数据的写入条目。
- *minimal* - 打印写入的 *namespace:key = value* 对。
- *namespaces* - 打印所有写入的命名空间。
- *blobs* - 打印所有 blob 和字符串（若 blob 和字符串是以分块的形式，则对其进行重组）。
- *storage_info* - 打印每一页面的条目状态计数。

注意： 该程序还提供 *none* 选项，该选项不会打印任何内容。如果 NVS 分区的内容并不相关，可以将该选项和完整性检查选项一起使用。

该程序支持完整性检查功能，选择选项 *-i* 或 *--integrity-check* 即可运行（该选项会导致 *json* 输出格式无效，因此只适用于 *text* 格式）。此功能可扫描整个分区，并打印出可能存在的错误。当此功能和 *-d none* 一起使用时，可只打印可能存在的错误。

2.8.7 SD/SDIO/MMC 驱动程序

概述

SD/SDIO/MMC 驱动是一种基于 SDMMC 和 SD SPI 主机驱动的协议级驱动程序，目前已支持 SD 存储器、SDIO 卡和 eMMC 芯片。

SDMMC 主机驱动和 SD SPI 主机驱动（esp_driver_sdmmc/include/driver/sdmmc_host.h 和 esp_driver_sdspi/include/driver/sdspi_host.h）为以下功能提供 API：

- 发送命令至从设备
- 接收和发送数据
- 处理总线错误

初始化函数及配置函数：

- 如需初始化和配置 SDMMC 主机，请参阅*SDMMC 主机 API*
- 如需初始化和配置 SD SPI 主机，请参阅*SD SPI 主机 API*

本文档中所述的 SDMMC 协议层仅处理 SD 协议相关事项，例如卡初始化和数据传输命令。

协议层通过 `sdmmc_host_t` 结构体和主机协同工作，该结构体包含指向主机各类函数的指针。

管脚配置

..only:: SOC_SDMMC_USE_IOMUX and not SOC_SDMMC_USE_GPIO_MATRIX

SDMMC 管脚为专用管脚，无需配置。

..only:: SOC_SDMMC_USE_GPIO_MATRIX and not SOC_SDMMC_USE_IOMUX

SDMMC 管脚信号通过 GPIO 交换矩阵配置，请在*sdmmc_slot_config_t* 中配置管脚。

..only:: esp32p4

SDMMC 有两个卡槽：

- 卡槽 0 管脚为 UHS-I 模式专用，但驱动程序尚不支持此模式。
- 卡槽 1 管脚可通过 GPIO 交换矩阵配置，用于 UHS-I 之外的情况。如要使用卡槽 1，请在*sdmmc_slot_config_t* 中配置管脚。

应用示例

ESP-IDF storage/sd_card 目录下提供了 SDMMC 驱动与 FatFs 库组合使用的示例，演示了先初始化卡，然后使用 POSIX 和 C 库 API 向卡读写数据。请参考示例目录下 README.md 文件，查看更多详细信息。

复合卡（存储 + IO） 该驱动程序不支持 SD 复合卡，复合卡会被视为 IO 卡。

线程安全 多数应用程序仅需在一个任务中使用协议层。因此，协议层在 `sdmmc_card_t` 结构体或在访问 SDMMC 或 SD SPI 主机驱动程序时不使用任何类型的锁。这种锁通常在较高层级实现，例如文件系统驱动程序。

协议层 API

协议层具备 `sdmmc_host_t` 结构体，此结构体描述了 SD/MMC 主机驱动，列出了其功能，并提供指向驱动程序函数的指针。协议层将卡信息储存于 `sdmmc_card_t` 结构体中。向 SD/MMC 主机发送命令时，协议层使用 `sdmmc_command_t` 结构体来描述命令、参数、预期返回值和需传输的数据（如有）。

用于 SD 存储卡的 API

1. 初始化主机，请调用主机驱动函数，例如*sdmmc_host_init()* 和*sdmmc_host_init_slot()*；
2. 初始化卡，请调用*sdmmc_card_init()*，并将参数 `host`（主机驱动信息）和参数 `card`（指向 `sdmmc_card_t` 结构体的指针）传递给此函数。函数运行结束后，将会向 `sdmmc_card_t` 结构体填充该卡的信息；
3. 读取或写入卡的扇区，请分别调用*sdmmc_read_sectors()* 和*sdmmc_write_sectors()*，并将参数 `card`（指向卡信息结构的指针）传递给函数；
4. 如果不再使用该卡，请调用主机驱动函数，例如*sdmmc_host_deinit()*，以禁用主机外设，并释放驱动程序分配的资源。

用于 eMMC 芯片的 API 从协议层的角度而言，eMMC 存储芯片与 SD 存储卡相同。尽管 eMMC 是芯片，不具备卡的外形，但由于协议相似 (*sdmmc_card_t*, *sdmmc_card_init*)，用于 SD 卡的一些概念同样适用于 eMMC 芯片。注意，eMMC 芯片不可通过 SPI 使用，因此它与 SD SPI 主机驱动不兼容。

如需初始化 eMMC 内存并执行读/写操作，请参照上一章节 SD 卡操作步骤。

用于 SDIO 卡的 API SDIO 卡初始化和检测过程与 SD 存储卡相同，唯一的区别是 SDIO 模式下数据传输命令不同。

在卡初始化和卡检测（通过运行*sdmmc_card_init()*）期间，驱动仅配置 IO 卡如下寄存器：

1. I/O 中止 (0x06) 寄存器：在该寄存器中设置 RES 位可重置卡的 IO 部分；
2. 总线接口控制 (0x07) 寄存器：如果主机和插槽配置中启用 4 线模式，则驱动程序会尝试在该寄存器中设置总线宽度字段。如果字段设置成功，则从机支持 4 线模式，主机也切换至 4 线模式；
3. 高速 (0x13) 寄存器：如果主机配置中启用高速模式，则该寄存器的 SHS 位会被设置。

注意，驱动程序不会在 (1) I/O 使能寄存器和 Int 使能寄存器，及 (2) I/O 块大小中，设置任何位。应用程序可通过调用*sdmmc_io_write_byte()* 来设置相关位。

如需卡配置或传输数据，请根据具体情况，选择下表函数：

操作	函数读取	函数写入
使用 IO_RW_DIRECT (CMD52) 读写单个字节。	*sdmmc_io_read_byte()*	*sdmmc_io_write_byte()*
使用 IO_RW_EXTENDED (CMD53) 的字节模式读写多个字节。	*sdmmc_io_read_bytes()*	*sdmmc_io_write_bytes()*
块模式下，使用 IO_RW_EXTENDED (CMD53) 读写数据块。	*sdmmc_io_read_blocks()*	*sdmmc_io_write_blocks()*

使用*sdmmc_io_enable_int()* 函数，应用程序可启用 SDIO 中断。在单线模式下使用 SDIO 时，还需要连接 D1 线来启用 SDIO 中断。

如果需要应用程序保持等待直至发生 SDIO 中断，请使用*sdmmc_io_wait_int()* 函数。

如果需要与 ESP32 的 SDIO 从设备通信，请使用 ESSL 组件（ESP 串行从设备链接）。请参阅*ESP 串行从机链路* 和 peripherals/sdio/host。

API 参考

Header File

- components/sdmmc/include/sdmmc_cmd.h
- This header file can be included with:

```
#include "sdmmc_cmd.h"
```

- This header file is a part of the API provided by the `sdmmc` component. To declare that your component depends on `sdmmc`, add the following to your CMakeLists.txt:

```
REQUIRES sdmmc
```

or

```
PRIV_REQUIRES sdmmc
```

Functions

esp_err_t **sdmmc_card_init**(const sdmmc_host_t *host, sdmmc_card_t *out_card)

Probe and initialize SD/MMC card using given host

备注： Only SD cards (SDSC and SDHC/SDXC) are supported now. Support for MMC/eMMC cards will be added later.

参数

- **host** -- pointer to structure defining host controller
- **out_card** -- pointer to structure which will receive information about the card when the function completes

返回

- ESP_OK on success
- One of the error codes from SDMMC host controller

void **sdmmc_card_print_info**(FILE *stream, const sdmmc_card_t *card)

Print information about the card to a stream.

参数

- **stream** -- stream obtained using fopen or fdopen
- **card** -- card information structure initialized using sdmmc_card_init

esp_err_t **sdmmc_get_status**(sdmmc_card_t *card)

Get status of SD/MMC card

参数 **card** -- pointer to card information structure previously initialized using sdmmc_card_init

返回

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_write_sectors**(sdmmc_card_t *card, const void *src, size_t start_sector, size_t sector_count)

Write given number of sectors to SD/MMC card

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **src** -- pointer to data buffer to read data from; data size must be equal to sector_count * card->csd.sector_size
- **start_sector** -- sector where to start writing
- **sector_count** -- number of sectors to write

返回

- ESP_OK on success or sector_count equal to 0
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_read_sectors**(sdmmc_card_t *card, void *dst, size_t start_sector, size_t sector_count)

Read given number of sectors from the SD/MMC card

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **dst** -- pointer to data buffer to write into; buffer size must be at least sector_count * card->csd.sector_size
- **start_sector** -- sector where to start reading
- **sector_count** -- number of sectors to read

返回

- ESP_OK on success or sector_count equal to 0
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_erase_sectors**(sdmmc_card_t *card, size_t start_sector, size_t sector_count, sdmmc_erase_arg_t arg)

Erase given number of sectors from the SD/MMC card

备注: When sdmmc_erase_sectors used with cards in SDSPI mode, it was observed that card requires re-init after erase operation.

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **start_sector** -- sector where to start erase
- **sector_count** -- number of sectors to erase

- **arg** -- erase command (CMD38) argument

返回

- ESP_OK on success or sector_count equal to 0
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_can_discard**(sdmmc_card_t *card)

Check if SD/MMC card supports discard

参数 **card** -- pointer to card information structure previously initialized using sdmmc_card_init

返回

- ESP_OK if supported by the card/device
- ESP_FAIL if not supported by the card/device

esp_err_t **sdmmc_can_trim**(sdmmc_card_t *card)

Check if SD/MMC card supports trim

参数 **card** -- pointer to card information structure previously initialized using sdmmc_card_init

返回

- ESP_OK if supported by the card/device
- ESP_FAIL if not supported by the card/device

esp_err_t **sdmmc_mmc_can_sanitize**(sdmmc_card_t *card)

Check if SD/MMC card supports sanitize

参数 **card** -- pointer to card information structure previously initialized using sdmmc_card_init

返回

- ESP_OK if supported by the card/device
- ESP_FAIL if not supported by the card/device

esp_err_t **sdmmc_mmc_sanitize**(sdmmc_card_t *card, uint32_t timeout_ms)

Sanitize the data that was unmapped by a Discard command

备注: Discard command has to precede sanitize operation. To discard, use MMC_DICARD_ARG with sdmmc_erase_sectors argument

---

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **timeout_ms** -- timeout value in milliseconds required to sanitize the selected range of sectors.

返回

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_full_erase**(sdmmc_card_t *card)

Erase complete SD/MMC card

参数 **card** -- pointer to card information structure previously initialized using sdmmc_card_init

返回

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_read_byte**(sdmmc_card_t *card, uint32_t function, uint32_t reg, uint8_t *out_byte)

Read one byte from an SDIO card using IO_RW_DIRECT (CMD52)

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **reg** -- byte address within IO function

- **out_byte** -- **[out]** output, receives the value read from the card

返回

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_write_byte**(sdmmc_card_t *card, uint32_t function, uint32_t reg, uint8_t in_byte, uint8_t *out_byte)

Write one byte to an SDIO card using IO_RW_DIRECT (CMD52)

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **reg** -- byte address within IO function
- **in_byte** -- value to be written
- **out_byte** -- **[out]** if not NULL, receives new byte value read from the card (read-after-write).

返回

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_read_bytes**(sdmmc_card_t *card, uint32_t function, uint32_t addr, void *dst, size_t size)

Read multiple bytes from an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs read operation using CMD53 in byte mode. For block mode, see sdmmc_io_read_blocks.

By default OP Code is set (incrementing address). To send CMD53 without this bit, OR the argument `addr` with `SDMMC_IO_FIXED_ADDR`.

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **addr** -- byte address within IO function where reading starts
- **dst** -- buffer which receives the data read from card. Aligned to 4 byte boundary unless `SDMMC_HOST_FLAG_ALLOC_ALIGNED_BUF` flag is set when calling `sdmmc_card_init`. The flag is mandatory when the buffer is behind the cache.
- **size** -- number of bytes to read, 1 to 512.

返回

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_write_bytes**(sdmmc_card_t *card, uint32_t function, uint32_t addr, const void *src, size_t size)

Write multiple bytes to an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs write operation using CMD53 in byte mode. For block mode, see sdmmc_io_write_blocks.

By default OP Code is set (incrementing address). To send CMD53 without this bit, OR the argument `addr` with `SDMMC_IO_FIXED_ADDR`.

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **addr** -- byte address within IO function where writing starts
- **src** -- data to be written. Aligned to 4 byte boundary unless `SDMMC_HOST_FLAG_ALLOC_ALIGNED_BUF` flag is set when calling

sdmmc_card_init. The flag is mandatory when the buffer is behind the cache.
- **size** -- number of bytes to write, 1 to 512.

返回
- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_read_blocks**(sdmmc_card_t *card, uint32_t function, uint32_t addr, void *dst, size_t size)

Read blocks of data from an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs read operation using CMD53 in block mode. For byte mode, see sdmmc_io_read_bytes.

By default OP Code is set (incrementing address). To send CMD53 without this bit, OR the argument addr with SDMMC_IO_FIXED_ADDR.

参数
- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **addr** -- byte address within IO function where writing starts
- **dst** -- buffer which receives the data read from card. Aligned to 4 byte boundary, and also cache line size if the buffer is behind the cache.
- **size** -- number of bytes to read, must be divisible by the card block size.

返回
- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size is not divisible by 512 bytes
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_write_blocks**(sdmmc_card_t *card, uint32_t function, uint32_t addr, const void *src, size_t size)

Write blocks of data to an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs write operation using CMD53 in block mode. For byte mode, see sdmmc_io_write_bytes.

By default OP Code is set (incrementing address). To send CMD53 without this bit, OR the argument addr with SDMMC_IO_FIXED_ADDR.

参数
- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **addr** -- byte address within IO function where writing starts
- **src** -- data to be written. Aligned to 4 byte boundary, and also cache line size if the buffer is behind the cache.
- **size** -- number of bytes to read, must be divisible by the card block size.

返回
- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size is not divisible by 512 bytes
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_enable_int**(sdmmc_card_t *card)

Enable SDIO interrupt in the SDMMC host

参数 **card** -- pointer to card information structure previously initialized using sdmmc_card_init

返回
- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if the host controller does not support IO interrupts

esp_err_t **sdmmc_io_wait_int**(sdmmc_card_t *card, TickType_t timeout_ticks)

Block until an SDIO interrupt is received

Slave uses D1 line to signal interrupt condition to the host. This function can be used to wait for the interrupt.

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **timeout_ticks** -- time to wait for the interrupt, in RTOS ticks

返回

- ESP_OK if the interrupt is received
- ESP_ERR_NOT_SUPPORTED if the host controller does not support IO interrupts
- ESP_ERR_TIMEOUT if the interrupt does not happen in timeout_ticks

esp_err_t **sdmmc_io_get_cis_data**(sdmmc_card_t *card, uint8_t *out_buffer, size_t buffer_size, size_t *inout_cis_size)

Get the data of CIS region of an SDIO card.

You may provide a buffer not sufficient to store all the CIS data. In this case, this function stores as much data into your buffer as possible. Also, this function will try to get and return the size required for you.

参数

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **out_buffer** -- Output buffer of the CIS data
- **buffer_size** -- Size of the buffer.
- **inout_cis_size** -- Mandatory, pointer to a size, input and output.
  - input: Limitation of maximum searching range, should be 0 or larger than buffer_size. The function searches for CIS_CODE_END until this range. Set to 0 to search infinitely.
  - output: The size required to store all the CIS data, if CIS_CODE_END is found.

返回

- ESP_OK: on success
- ESP_ERR_INVALID_RESPONSE: if the card does not (correctly) support CIS.
- ESP_ERR_INVALID_SIZE: CIS_CODE_END found, but buffer_size is less than required size, which is stored in the inout_cis_size then.
- ESP_ERR_NOT_FOUND: if the CIS_CODE_END not found. Increase input value of inout_cis_size or set it to 0, if you still want to search for the end; output value of inout_cis_size is invalid in this case.
- and other error code return from sdmmc_io_read_bytes

esp_err_t **sdmmc_io_print_cis_info**(uint8_t *buffer, size_t buffer_size, FILE *fp)

Parse and print the CIS information of an SDIO card.

备注: Not all the CIS codes and all kinds of tuples are supported. If you see some unresolved code, you can add the parsing of these code in sdmmc_io.c and contribute to the IDF through the Github repository.

```
using sdmmc_card_init
```

参数

- **buffer** -- Buffer to parse
- **buffer_size** -- Size of the buffer.
- **fp** -- File pointer to print to, set to NULL to print to stdout.

返回

- ESP_OK: on success
- ESP_ERR_NOT_SUPPORTED: if the value from the card is not supported to be parsed.
- ESP_ERR_INVALID_SIZE: if the CIS size fields are not correct.

Macros

SDMMC_IO_FIXED_ADDR

Call `sdmmc_io_read_bytes`, `sdmmc_io_write_bytes`, `sdmmc_io_read_blocks` or `sdmmc_io_write_bocks` APIs with address ORed by this flag to send CMD53 with OP Code clear (fixed address)

Header File

- components/esp_driver_sdmmc/include/driver/sdmmc_types.h
- This header file can be included with:

```
#include "driver/sdmmc_types.h"
```

- This header file is a part of the API provided by the `esp_driver_sdmmc` component. To declare that your component depends on `esp_driver_sdmmc`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_sdmmc
```

or

```
PRIV_REQUIRES esp_driver_sdmmc
```

2.8.8 分区 API

概述

`esp_partition` 组件提供了高层次的 API 函数，用于访问定义在分区表 中的分区。这些 API 基于*SPI flash API* 提供的低层次 API。

分区表 API

ESP-IDF 工程使用分区表保存 SPI flash 各区信息，包括引导程序、各种应用程序二进制文件、数据及文件系统等。请参阅分区表，查看详细信息。

该组件在 `esp_partition.h` 中声明了一些 API 函数，用以枚举在分区表中找到的分区，并对这些分区执行操作：

- *esp_partition_find()*：在分区表中查找特定类型的条目，返回一个不透明迭代器；
- *esp_partition_get()*：返回一个结构体，描述给定迭代器的分区；
- *esp_partition_next()*：将迭代器移至下一个找到的分区；
- *esp_partition_iterator_release()*：释放*esp_partition_find()* 中返回的迭代器；
- *esp_partition_find_first()*：返回描述*esp_partition_find()* 中找到的第一个分区的结构；
- *esp_partition_read()*、*esp_partition_write()* 和*esp_partition_erase_range()* 等同于*esp_flash_read()*、*esp_flash_write()* 和*esp_flash_erase_region()*，但在分区边界内执行。

另请参考

- 分区表
- 空中升级 *(OTA)* 提供了高层 API 用于更新存储在 flash 中的 app 固件。
- 非易失性存储库 提供了结构化 API 用于存储 SPI flash 中的碎片数据。

分区表 API 参考

Header File

- components/esp_partition/include/esp_partition.h
- This header file can be included with:

```
#include "esp_partition.h"
```

- This header file is a part of the API provided by the `esp_partition` component. To declare that your component depends on `esp_partition`, add the following to your CMakeLists.txt:

```
REQUIRES esp_partition
```

or

```
PRIV_REQUIRES esp_partition
```

Functions

esp_partition_iterator_t **esp_partition_find**(*esp_partition_type_t* type, *esp_partition_subtype_t* subtype, const char *label)

Find partition based on one or more parameters.

参数

- **type** -- Partition type, one of esp_partition_type_t values or an 8-bit unsigned integer. To find all partitions, no matter the type, use ESP_PARTITION_TYPE_ANY, and set subtype argument to ESP_PARTITION_SUBTYPE_ANY.
- **subtype** -- Partition subtype, one of esp_partition_subtype_t values or an 8-bit unsigned integer. To find all partitions of given type, use ESP_PARTITION_SUBTYPE_ANY.
- **label** -- (optional) Partition label. Set this value if looking for partition with a specific name. Pass NULL otherwise.

返回 iterator which can be used to enumerate all the partitions found, or NULL if no partitions were found. Iterator obtained through this function has to be released using esp_partition_iterator_release when not used any more.

const *esp_partition_t* ***esp_partition_find_first**(*esp_partition_type_t* type, *esp_partition_subtype_t* subtype, const char *label)

Find first partition based on one or more parameters.

参数

- **type** -- Partition type, one of esp_partition_type_t values or an 8-bit unsigned integer. To find all partitions, no matter the type, use ESP_PARTITION_TYPE_ANY, and set subtype argument to ESP_PARTITION_SUBTYPE_ANY.
- **subtype** -- Partition subtype, one of esp_partition_subtype_t values or an 8-bit unsigned integer To find all partitions of given type, use ESP_PARTITION_SUBTYPE_ANY.
- **label** -- (optional) Partition label. Set this value if looking for partition with a specific name. Pass NULL otherwise.

返回 pointer to *esp_partition_t* structure, or NULL if no partition is found. This pointer is valid for the lifetime of the application.

const *esp_partition_t* ***esp_partition_get**(*esp_partition_iterator_t* iterator)

Get *esp_partition_t* structure for given partition.

参数 **iterator** -- Iterator obtained using esp_partition_find. Must be non-NULL.

返回 pointer to *esp_partition_t* structure. This pointer is valid for the lifetime of the application.

esp_partition_iterator_t **esp_partition_next**(*esp_partition_iterator_t* iterator)

Move partition iterator to the next partition found.

Any copies of the iterator will be invalid after this call.

参数 **iterator** -- Iterator obtained using esp_partition_find. Must be non-NULL.

返回 NULL if no partition was found, valid esp_partition_iterator_t otherwise.

void **esp_partition_iterator_release**(*esp_partition_iterator_t* iterator)

Release partition iterator.

参数 **iterator** -- Iterator obtained using esp_partition_find. The iterator is allowed to be NULL, so it is not necessary to check its value before calling this function.

const *esp_partition_t* ***esp_partition_verify**(const *esp_partition_t* *partition)

Verify partition data.

Given a pointer to partition data, verify this partition exists in the partition table (all fields match.)

This function is also useful to take partition data which may be in a RAM buffer and convert it to a pointer to the permanent partition data stored in flash.

Pointers returned from this function can be compared directly to the address of any pointer returned from *esp_partition_get()*, as a test for equality.

参数 **partition** -- Pointer to partition data to verify. Must be non-NULL. All fields of this structure must match the partition table entry in flash for this function to return a successful match.

返回

- If partition not found, returns NULL.
- If found, returns a pointer to the *esp_partition_t* structure in flash. This pointer is always valid for the lifetime of the application.

esp_err_t **esp_partition_read**(const *esp_partition_t* *partition, size_t src_offset, void *dst, size_t size)

Read data from the partition.

Partitions marked with an encryption flag will automatically be be read and decrypted via a cache mapping.

参数

- **partition** -- Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **dst** -- Pointer to the buffer where data should be stored. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- **src_offset** -- Address of the data to be read, relative to the beginning of the partition.
- **size** -- Size of data to be read, in bytes.

返回 ESP_OK, if data was read successfully; ESP_ERR_INVALID_ARG, if src_offset exceeds partition size; ESP_ERR_INVALID_SIZE, if read would go out of bounds of the partition; or one of error codes from lower-level flash driver.

esp_err_t **esp_partition_write**(const *esp_partition_t* *partition, size_t dst_offset, const void *src, size_t size)

Write data to the partition.

Before writing data to flash, corresponding region of flash needs to be erased. This can be done using esp_partition_erase_range function.

Partitions marked with an encryption flag will automatically be written via the esp_flash_write_encrypted() function. If writing to an encrypted partition, all write offsets and lengths must be multiples of 16 bytes. See the esp_flash_write_encrypted() function for more details. Unencrypted partitions do not have this restriction.

备注: Prior to writing to flash memory, make sure it has been erased with esp_partition_erase_range call.

参数

- **partition** -- Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **dst_offset** -- Address where the data should be written, relative to the beginning of the partition.
- **src** -- Pointer to the source buffer. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- **size** -- Size of data to be written, in bytes.

返回 ESP_OK, if data was written successfully; ESP_ERR_INVALID_ARG, if dst_offset exceeds partition size; ESP_ERR_INVALID_SIZE, if write would go out of bounds of the partition; ESP_ERR_NOT_ALLOWED, if partition is read-only; or one of error codes from lower-level flash driver.

esp_err_t **esp_partition_read_raw**(const *esp_partition_t* *partition, size_t src_offset, void *dst, size_t size)

Read data from the partition without any transformation/decryption.

备注: This function is essentially the same as *esp_partition_read()* above. It just never decrypts data but returns it as is.

参数

- **partition** -- Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **dst** -- Pointer to the buffer where data should be stored. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- **src_offset** -- Address of the data to be read, relative to the beginning of the partition.
- **size** -- Size of data to be read, in bytes.

返回 ESP_OK, if data was read successfully; ESP_ERR_INVALID_ARG, if src_offset exceeds partition size; ESP_ERR_INVALID_SIZE, if read would go out of bounds of the partition; or one of error codes from lower-level flash driver.

esp_err_t **esp_partition_write_raw**(const *esp_partition_t* *partition, size_t dst_offset, const void *src, size_t size)

Write data to the partition without any transformation/encryption.

Before writing data to flash, corresponding region of flash needs to be erased. This can be done using esp_partition_erase_range function.

备注: This function is essentially the same as *esp_partition_write()* above. It just never encrypts data but writes it as is.

备注: Prior to writing to flash memory, make sure it has been erased with esp_partition_erase_range call.

参数

- **partition** -- Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **dst_offset** -- Address where the data should be written, relative to the beginning of the partition.
- **src** -- Pointer to the source buffer. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- **size** -- Size of data to be written, in bytes.

返回 ESP_OK, if data was written successfully; ESP_ERR_INVALID_ARG, if dst_offset exceeds partition size; ESP_ERR_INVALID_SIZE, if write would go out of bounds of the partition; ESP_ERR_NOT_ALLOWED, if partition is read-only; or one of the error codes from lower-level flash driver.

esp_err_t **esp_partition_erase_range**(const *esp_partition_t* *partition, size_t offset, size_t size)

Erase part of the partition.

参数

- **partition** -- Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **offset** -- Offset from the beginning of partition where erase operation should start. Must be aligned to partition->erase_size.
- **size** -- Size of the range which should be erased, in bytes. Must be divisible by partition->erase_size.

返回 ESP_OK, if the range was erased successfully; ESP_ERR_INVALID_ARG, if iterator or dst are NULL; ESP_ERR_INVALID_SIZE, if erase would go out of bounds of the partition; ESP_ERR_NOT_ALLOWED, if partition is read-only; or one of error codes from lower-level flash driver.

esp_err_t **esp_partition_mmap**(const *esp_partition_t* *partition, size_t offset, size_t size, *esp_partition_mmap_memory_t* memory, const void **out_ptr, *esp_partition_mmap_handle_t* *out_handle)

Configure MMU to map partition into data memory.

Unlike spi_flash_mmap function, which requires a 64kB aligned base address, this function doesn't impose such a requirement. If offset results in a flash address which is not aligned to 64kB boundary, address will be rounded to the lower 64kB boundary, so that mapped region includes requested range. Pointer returned via out_ptr argument will be adjusted to point to the requested offset (not necessarily to the beginning of mmap-ed region).

To release mapped memory, pass handle returned via out_handle argument to esp_partition_munmap function.

参数

- **partition** -- Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **offset** -- Offset from the beginning of partition where mapping should start.
- **size** -- Size of the area to be mapped.
- **memory** -- Memory space where the region should be mapped
- **out_ptr** -- Output, pointer to the mapped memory region
- **out_handle** -- Output, handle which should be used for esp_partition_munmap call

返回 ESP_OK, if successful

void **esp_partition_munmap**(*esp_partition_mmap_handle_t* handle)

Release region previously obtained using esp_partition_mmap.

备注: Calling this function will not necessarily unmap memory region. Region will only be unmapped when there are no other handles which reference this region. In case of partially overlapping regions it is possible that memory will be unmapped partially.

参数 **handle** -- Handle obtained from spi_flash_mmap

esp_err_t **esp_partition_get_sha256**(const *esp_partition_t* *partition, uint8_t *sha_256)

Get SHA-256 digest for required partition.

For apps with SHA-256 appended to the app image, the result is the appended SHA-256 value for the app image content. The hash is verified before returning, if app content is invalid then the function returns ESP_ERR_IMAGE_INVALID. For apps without SHA-256 appended to the image, the result is the SHA-256 of all bytes in the app image. For other partition types, the result is the SHA-256 of the entire partition.

参数

- **partition** -- **[in]** Pointer to info for partition containing app or data. (fields: address, size and type, are required to be filled).
- **sha_256** -- **[out]** Returned SHA-256 digest for a given partition.

返回

- ESP_OK: In case of successful operation.
- ESP_ERR_INVALID_ARG: The size was 0 or the sha_256 was NULL.
- ESP_ERR_NO_MEM: Cannot allocate memory for sha256 operation.

- ESP_ERR_IMAGE_INVALID: App partition doesn't contain a valid app image.
- ESP_FAIL: An allocation error occurred.

bool **esp_partition_check_identity**(const *esp_partition_t* *partition_1, const *esp_partition_t* *partition_2)

Check for the identity of two partitions by SHA-256 digest.

参数

- **partition_1** -- **[in]** Pointer to info for partition 1 containing app or data. (fields: address, size and type, are required to be filled).
- **partition_2** -- **[in]** Pointer to info for partition 2 containing app or data. (fields: address, size and type, are required to be filled).

返回

- True: In case of the two firmware is equal.
- False: Otherwise

esp_err_t **esp_partition_register_external**(*esp_flash_t* *flash_chip, size_t offset, size_t size, const char *label, *esp_partition_type_t* type, *esp_partition_subtype_t* subtype, const *esp_partition_t* **out_partition)

Register a partition on an external flash chip.

This API allows designating certain areas of external flash chips (identified by the *esp_flash_t* structure) as partitions. This allows using them with components which access SPI flash through the esp_partition API.

参数

- **flash_chip** -- Pointer to the structure identifying the flash chip
- **offset** -- Address in bytes, where the partition starts
- **size** -- Size of the partition in bytes
- **label** -- Partition name
- **type** -- One of the partition types (ESP_PARTITION_TYPE_*), or an integer. Note that applications can not be booted from external flash chips, so using ESP_PARTITION_TYPE_APP is not supported.
- **subtype** -- One of the partition subtypes (ESP_PARTITION_SUBTYPE_*), or an integer.
- **out_partition** -- **[out]** Output, if non-NULL, receives the pointer to the resulting *esp_partition_t* structure

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if memory allocation has failed
- ESP_ERR_INVALID_ARG if the new partition overlaps another partition on the same flash chip
- ESP_ERR_INVALID_SIZE if the partition doesn't fit into the flash chip size

esp_err_t **esp_partition_deregister_external**(const *esp_partition_t* *partition)

Deregister the partition previously registered using esp_partition_register_external.

参数 **partition** -- pointer to the partition structure obtained from esp_partition_register_external,

返回

- ESP_OK on success
- ESP_ERR_NOT_FOUND if the partition pointer is not found
- ESP_ERR_INVALID_ARG if the partition comes from the partition table
- ESP_ERR_INVALID_ARG if the partition was not registered using esp_partition_register_external function.

void **esp_partition_unload_all**(void)

Unload partitions and free space allocated by them.

Structures

struct **esp_partition_t**

partition information structure

This is not the format in flash, that format is esp_partition_info_t.

However, this is the format used by this API.

Public Members

esp_flash_t ***flash_chip**

SPI flash chip on which the partition resides

esp_partition_type_t **type**

partition type (app/data)

esp_partition_subtype_t **subtype**

partition subtype

uint32_t **address**

starting address of the partition in flash

uint32_t **size**

size of the partition, in bytes

uint32_t **erase_size**

size the erase operation should be aligned to

char **label**[17]

partition label, zero-terminated ASCII string

bool **encrypted**

flag is set to true if partition is encrypted

bool **readonly**

flag is set to true if partition is read-only

Macros

ESP_PARTITION_SUBTYPE_OTA(i)

Convenience macro to get esp_partition_subtype_t value for the i-th OTA partition.

Type Definitions

typedef uint32_t **esp_partition_mmap_handle_t**

Opaque handle for memory region obtained from esp_partition_mmap.

typedef struct esp_partition_iterator_opaque_ ***esp_partition_iterator_t**

Opaque partition iterator type.

Enumerations

enum **esp_partition_mmap_memory_t**

Enumeration which specifies memory space requested in an mmap call.

Values:

enumerator **ESP_PARTITION_MMAP_DATA**

map to data memory (Vaddr0), allows byte-aligned access, 4 MB total

enumerator **ESP_PARTITION_MMAP_INST**

map to instruction memory (Vaddr1-3), allows only 4-byte-aligned access, 11 MB total

enum **esp_partition_type_t**

Partition type.

备注： Partition types with integer value 0x00-0x3F are reserved for partition types defined by ESP-IDF. Any other integer value 0x40-0xFE can be used by individual applications, without restriction.

Values:

enumerator **ESP_PARTITION_TYPE_APP**

Application partition type.

enumerator **ESP_PARTITION_TYPE_DATA**

Data partition type.

enumerator **ESP_PARTITION_TYPE_ANY**

Used to search for partitions with any type.

enum **esp_partition_subtype_t**

Partition subtype.

Application-defined partition types (0x40-0xFE) can set any numeric subtype value.

备注： These ESP-IDF-defined partition subtypes apply to partitions of type ESP_PARTITION_TYPE_APP and ESP_PARTITION_TYPE_DATA.

Values:

enumerator **ESP_PARTITION_SUBTYPE_APP_FACTORY**

Factory application partition.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_MIN**

Base for OTA partition subtypes.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_0**

OTA partition 0.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_1**
OTA partition 1.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_2**
OTA partition 2.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_3**
OTA partition 3.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_4**
OTA partition 4.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_5**
OTA partition 5.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_6**
OTA partition 6.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_7**
OTA partition 7.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_8**
OTA partition 8.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_9**
OTA partition 9.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_10**
OTA partition 10.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_11**
OTA partition 11.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_12**
OTA partition 12.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_13**
OTA partition 13.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_14**
OTA partition 14.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_15**
OTA partition 15.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_MAX**
Max subtype of OTA partition.

enumerator **ESP_PARTITION_SUBTYPE_APP_TEST**
Test application partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_OTA**
OTA selection partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_PHY**
PHY init data partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_NVS**
NVS partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_COREDUMP**
COREDUMP partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_NVS_KEYS**
Partition for NVS keys.

enumerator **ESP_PARTITION_SUBTYPE_DATA_EFUSE_EM**
Partition for emulate eFuse bits.

enumerator **ESP_PARTITION_SUBTYPE_DATA_UNDEFINED**
Undefined (or unspecified) data partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_ESPHTTPD**
ESPHTTPD partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_FAT**
FAT partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_SPIFFS**
SPIFFS partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_LITTLEFS**
LITTLEFS partition.

enumerator **ESP_PARTITION_SUBTYPE_ANY**
Used to search for partitions with any subtype.

2.8.9 SPIFFS 文件系统

概述

SPIFFS 是一个用于 SPI NOR flash 设备的嵌入式文件系统，支持磨损均衡、文件系统一致性检查等功能。

说明

- 目前，SPIFFS 尚不支持目录，但可以生成扁平结构。如果 SPIFFS 挂载在 `/spiffs` 下，在 `/spiffs/tmp/myfile.txt` 路径下创建一个文件则会在 SPIFFS 中生成一个名为 `/tmp/myfile.txt` 的文件，而不是在 `/spiffs/tmp` 下生成名为 `myfile.txt` 的文件；
- SPIFFS 并非实时栈，每次写操作耗时不等；
- 目前，SPIFFS 尚不支持检测或处理已损坏的块。
- SPIFFS 只能稳定地使用约 75% 的指定分区容量。
- 当文件系统空间不足时，垃圾收集器会尝试多次扫描文件系统来寻找可用空间。根据所需空间的不同，写操作会被调用多次，每次函数调用将花费几秒。同一操作可能会花费不同时长的问题缘于 SPIFFS 的设计，且已在官方的 SPIFFS github 仓库 或是 https://github.com/espressif/esp-idf/issues/1737 中被多次报告。这个问题可以通过 SPIFFS 配置 部分缓解。
- 当垃圾收集器尝试多次（默认为 10 次）扫描整个文件系统以回收空间时，在每次扫描期间，如果有可用的数据块，则垃圾收集器会释放一个数据块。因此，如果为垃圾收集器设置的最大运行次数为 n（可通过 SPIFFS_GC_MAX_RUNS 选项配置，该选项位于 SPIFFS 配置 中），那么 n 倍数据块大小的空间将可用于写入数据。如果尝试写入超过 n 倍数据块大小的数据，写入操作可能会失败并返回错误。
- 如果 ESP32-P4 在文件系统操作期间断电，可能会导致 SPIFFS 损坏。但是仍可通过 `esp_spiffs_check` 函数恢复文件系统。详情请参阅官方 SPIFFS FAQ。

工具

`spiffsgen.py` spiffsgen.py （只写）是 SPIFFS 的一种 Python 实现，可用于从主机文件夹内容生成文件系统镜像。打开终端并运行以下命令即可使用 `spiffsgen.py`:

```
python spiffsgen.py <image_size> <base_dir> <output_file>
```

参数（必选）说明如下：

- **image_size**：分区大小，用于烧录生成的 SPIFFS 镜像；
- **base_dir**：创建 SPIFFS 镜像的目录；
- **output_file**：SPIFFS 镜像输出文件。

其他参数（可选）也参与控制镜像的生成，用户可以运行以下帮助命令，查看这些参数的具体信息:

```
python spiffsgen.py --help
```

上述可选参数对应 SPIFFS 构建配置选项。若想顺利生成可用的镜像，请确保使用的参数或配置与构建 SPIFFS 时所用的参数或配置相同。运行帮助命令将显示参数所对应的 SPIFFS 构建配置。如未指定参数，将使用帮助信息中的默认值。

镜像生成后，用户可以使用 `esptool.py` 或 `parttool.py` 烧录镜像。

用户可以在命令行或脚本中手动单独调用 `spiffsgen.py`，也可以直接从构建系统调用 `spiffs_create_partition_image` 来使用 `spiffsgen.py`:

```
spiffs_create_partition_image(<partition> <base_dir> [FLASH_IN_PROJECT] [DEPENDS␣
↪dep dep dep...])
```

在构建系统中使用 `spiffsgen.py` 更为方便，构建配置会自动传递给 `spiffsgen.py` 工具，确保生成的镜像可用于构建。比如，单独调用 `spiffsgen.py` 时需要用到 **image_size** 参数，但在构建系统中调用 `spiffs_create_partition_image` 时，仅需要 **partition** 参数，镜像大小将直接从工程分区表中获取。

使用 `spiffs_create_partition_image`，必须从组件 `CMakeLists.txt` 文件调用。

用户也可以指定 `FLASH_IN_PROJECT`，然后使用 `idf.py flash` 将镜像与应用程序二进制文件、分区表等一起自动烧录至设备，例如:

```
spiffs_create_partition_image(my_spiffs_partition my_folder FLASH_IN_PROJECT)
```

不指定 FLASH_IN_PROJECT/SPIFFS_IMAGE_FLASH_IN_PROJECT 也可以生成镜像，但须使用 `esptool.py`、`parttool.py` 或自定义构建系统目标手动烧录。

有时基本目录中的内容是在构建时生成的，用户可以使用 DEPENDS/SPIFFS_IMAGE_DEPENDS 指定目标，因此可以在生成镜像之前执行此目标:

```
add_custom_target(dep COMMAND ...)

spiffs_create_partition_image(my_spiffs_partition my_folder DEPENDS dep)
```

请参考 storage/spiffsgen，查看示例。

mkspiffs 用户也可以使用 mkspiffs 工具创建 SPIFFS 分区镜像。与 `spiffsgen.py` 相似，mkspiffs 也可以用于从指定文件夹中生成镜像，然后使用 `esptool.py` 烧录镜像。

该工具需要获取以下参数:

- **Block Size**：4096（SPI flash 标准）
- **Page Size**：256（SPI flash 标准）
- **Image Size**：分区大小（以字节为单位，可从分区表中获取）
- **Partition Offset**：分区起始地址（可从分区表中获取）

运行以下命令，将文件夹打包成 1 MB 大小的镜像:

```
mkspiffs -c [src_folder] -b 4096 -p 256 -s 0x100000 spiffs.bin
```

运行以下命令，将镜像烧录到 ESP32-P4（偏移量：0x110000）:

```
python esptool.py --chip esp32p4 --port [port] --baud [baud] write_flash -z␣
↪0x110000 spiffs.bin
```

选择合适的 SPIFFS 工具 上面介绍的两款 SPIFFS 工具功能相似，需根据实际情况，选择合适的一款。

以下情况优先选用 `spiffsgen.py` 工具:

1. 仅需在构建时简单生成 SPIFFS 镜像，请选择使用 `spiffsgen.py`，因为 `spiffsgen.py` 可以直接在构建系统中使用函数或命令生成 SPIFFS 镜像。
2. 主机没有可用的 C/C++ 编译器时，可以选择使用 `spiffsgen.py` 工具，因为 `spiffsgen.py` 不需要编译。

以下情况优先选用 `mkspiffs` 工具:

1. 如果用户除了需要生成镜像外，还需要拆包 SPIFFS 镜像，请选择使用 `mkspiffs` 工具，因为 `spiffsgen.py` 目前尚不支持此功能。
2. 如果用户当前环境中 Python 解释器不可用，但主机编译器可用，或者有预编译的 `mkspiffs` 二进制文件，此时请选择使用 `mkspiffs` 工具。但是，`mkspiffs` 没有集成到构建系统，用户必须自己完成以下工作：在构建期间编译 `mkspiffs`（如果未使用预编译的二进制文件），为输出文件创建构建规则或目标，将适当的参数传递给工具等。

另请参阅

- 分区表

应用示例

storage/spiffs 目录下提供了 SPIFFS 应用示例。该示例初始化并挂载了一个 SPIFFS 分区，然后使用 POSIX 和 C 库 API 写入和读取数据。请参考 `example` 目录下的 README.md 文件，获取详细信息。

高级 API 参考

Header File

- components/spiffs/include/esp_spiffs.h
- This header file can be included with:

```
#include "esp_spiffs.h"
```

- This header file is a part of the API provided by the `spiffs` component. To declare that your component depends on `spiffs`, add the following to your CMakeLists.txt:

```
REQUIRES spiffs
```

or

```
PRIV_REQUIRES spiffs
```

Functions

esp_err_t **esp_vfs_spiffs_register**(const *esp_vfs_spiffs_conf_t* *conf)

Register and mount SPIFFS to VFS with given path prefix.

参数 **conf** -- Pointer to *esp_vfs_spiffs_conf_t* configuration structure

返回

- ESP_OK if success
- ESP_ERR_NO_MEM if objects could not be allocated
- ESP_ERR_INVALID_STATE if already mounted or partition is encrypted
- ESP_ERR_NOT_FOUND if partition for SPIFFS was not found
- ESP_FAIL if mount or format fails

esp_err_t **esp_vfs_spiffs_unregister**(const char *partition_label)

Unregister and unmount SPIFFS from VFS

参数 **partition_label** -- Same label as passed to esp_vfs_spiffs_register.

返回

- ESP_OK if successful
- ESP_ERR_INVALID_STATE already unregistered

bool **esp_spiffs_mounted**(const char *partition_label)

Check if SPIFFS is mounted

参数 **partition_label** -- Optional, label of the partition to check. If not specified, first partition with subtype=spiffs is used.

返回

- true if mounted
- false if not mounted

esp_err_t **esp_spiffs_format**(const char *partition_label)

Format the SPIFFS partition

参数 **partition_label** -- Same label as passed to esp_vfs_spiffs_register.

返回

- ESP_OK if successful
- ESP_FAIL on error

esp_err_t **esp_spiffs_info**(const char *partition_label, size_t *total_bytes, size_t *used_bytes)

Get information for SPIFFS

参数

- **partition_label** -- Same label as passed to esp_vfs_spiffs_register
- **total_bytes** -- **[out]** Size of the file system
- **used_bytes** -- **[out]** Current used bytes in the file system

返回

- ESP_OK if success
- ESP_ERR_INVALID_STATE if not mounted

esp_err_t **esp_spiffs_check**(const char *partition_label)

Check integrity of SPIFFS

参数 **partition_label** -- Same label as passed to esp_vfs_spiffs_register

返回

- ESP_OK if successful
- ESP_ERR_INVALID_STATE if not mounted
- ESP_FAIL on error

esp_err_t **esp_spiffs_gc**(const char *partition_label, size_t size_to_gc)

Perform garbage collection in SPIFFS partition.

Call this function to run GC and ensure that at least the given amount of space is available in the partition. This function will fail with ESP_ERR_NOT_FINISHED if it is not possible to reclaim the requested space (that is, not enough free or deleted pages in the filesystem). This function will also fail if it fails to reclaim the requested space after CONFIG_SPIFFS_GC_MAX_RUNS number of GC iterations. On one GC iteration, SPIFFS will erase one logical block (4kB). Therefore the value of CONFIG_SPIFFS_GC_MAX_RUNS should be set at least to the maximum expected size_to_gc, divided by 4096. For example, if the application expects to make room for a 1MB file and calls esp_spiffs_gc(label, 1024 * 1024), CONFIG_SPIFFS_GC_MAX_RUNS should be set to at least 256. On the other hand, increasing CONFIG_SPIFFS_GC_MAX_RUNS value increases the maximum amount of time for which any SPIFFS GC or write operation may potentially block.

参数

- **partition_label** -- Label of the partition to be garbage-collected. The partition must be already mounted.
- **size_to_gc** -- The number of bytes that the GC process should attempt to make available.

返回

- ESP_OK on success
- ESP_ERR_NOT_FINISHED if GC fails to reclaim the size given by size_to_gc
- ESP_ERR_INVALID_STATE if the partition is not mounted
- ESP_FAIL on all other errors

Structures

struct **esp_vfs_spiffs_conf_t**

Configuration structure for esp_vfs_spiffs_register.

Public Members

const char ***base_path**

File path prefix associated with the filesystem.

const char ***partition_label**

Optional, label of SPIFFS partition to use. If set to NULL, first partition with subtype=spiffs will be used.

size_t **max_files**

Maximum files that could be open at the same time.

bool **format_if_mount_failed**

If true, it will format the file system if it fails to mount.

2.8.10 虚拟文件系统组件

概述

虚拟文件系统 (VFS) 组件为驱动程序提供一个统一接口，可以操作类文件对象。这类驱动程序可以是 FAT、SPIFFS 等真实文件系统，也可以是提供文件类接口的设备驱动程序。

VFS 组件支持 C 库函数（如 fopen 和 fprintf 等）与文件系统 (FS) 驱动程序协同工作。在高层级，每个 FS 驱动程序均与某些路径前缀相关联。当一个 C 库函数需要打开文件时，VFS 组件将搜索与该文件所在文件路径相关联的 FS 驱动程序，并将调用传递给该驱动程序。针对该文件的读取、写入等其他操作的调用也将传递给这个驱动程序。

例如，使用 `/fat` 前缀注册 FAT 文件系统驱动，之后即可调用 `fopen("/fat/file.txt", "w")`。之后，VFS 将调用 FAT 驱动的 `open` 函数，并将参数 `/file.txt` 和合适的打开模式传递给 `open` 函数；后续对返回的 `FILE*` 数据流调用 C 库函数也同样会传递给 FAT 驱动。

注册 FS 驱动程序

如需注册 FS 驱动程序，应用程序首先要定义一个*esp_vfs_t* 结构体实例，并用指向 FS API 的函数指针填充它。

```
esp_vfs_t myfs = {
    .flags = ESP_VFS_FLAG_DEFAULT,
    .write = &myfs_write,
    .open = &myfs_open,
    .fstat = &myfs_fstat,
    .close = &myfs_close,
    .read = &myfs_read,
};

ESP_ERROR_CHECK(esp_vfs_register("/data", &myfs, NULL));
```

在上述代码中需要用到 `read`、`write` 或 `read_p`、`write_p`，具体使用哪组函数由 FS 驱动程序 API 的声明方式决定。

示例 1：声明 API 函数时不带额外的上下文指针参数，即 FS 驱动程序为单例模式，此时使用 `write`

```
ssize_t myfs_write(int fd, const void * data, size_t size);

// In definition of esp_vfs_t:
    .flags = ESP_VFS_FLAG_DEFAULT,
    .write = &myfs_write,
// ... other members initialized

// When registering FS, context pointer (third argument) is NULL:
ESP_ERROR_CHECK(esp_vfs_register("/data", &myfs, NULL));
```

示例 2：声明 API 函数时需要一个额外的上下文指针作为参数，即可支持多个 FS 驱动程序实例，此时使用 `write_p`

```
ssize_t myfs_write(myfs_t* fs, int fd, const void * data, size_t size);

// In definition of esp_vfs_t:
    .flags = ESP_VFS_FLAG_CONTEXT_PTR,
    .write_p = &myfs_write,
// ... other members initialized

// When registering FS, pass the FS context pointer into the third argument
```

(下页继续)

(续上页)

```
// (hypothetical myfs_mount function is used for illustrative purposes)
myfs_t* myfs_inst1 = myfs_mount(partition1->offset, partition1->size);
ESP_ERROR_CHECK(esp_vfs_register("/data1", &myfs, myfs_inst1));

// Can register another instance:
myfs_t* myfs_inst2 = myfs_mount(partition2->offset, partition2->size);
ESP_ERROR_CHECK(esp_vfs_register("/data2", &myfs, myfs_inst2));
```

同步输入/输出多路复用 VFS 组件支持通过 `select()` 进行同步输入/输出多路复用，其实现方式如下：

1. 调用 `select()`，使用时提供的文件描述符可以属于不同的 VFS 驱动。
2. 文件描述符被分为几组，每组属于一个 VFS 驱动。
3. 非套接字 VFS 驱动的文件描述符由 `start_select()` 移交给指定的 VFS 驱动，后文会对此进行详述。该函数代表指定驱动 `select()` 的实现。这是一个非阻塞的调用，意味着在设置好检查与指定文件描述符相关事件的环境后，该函数应该立即返回。
4. 套接字 VFS 驱动的文件描述符由 `socket_select()` 移交给套接字 VFS 驱动，后文会对此进行详述。这是一个阻塞调用，意味着只有当有一个与套接字文件描述符相关的事件或非套接字驱动发出信号让 `socket_select()` 退出时，它才会返回。
5. 从各个 VFS 驱动程序收集结果，并通过对事件检查环境取消初始化来终止所有驱动程序。
6. `select()` 调用结束并返回适当的结果。

非套接字 VFS 驱动 如果要使用非套接字 VFS 驱动的文件描述符调用 `select()`，那么需要用函数 `start_select()` 和 `end_select()` 注册该驱动，具体如下：

```
// In definition of esp_vfs_t:
    .start_select = &uart_start_select,
    .end_select = &uart_end_select,
// ... other members initialized
```

调用 `start_select()` 函数可以设置环境，检测指定 VFS 驱动的文件描述符读取/写入/错误条件。

调用 `end_select()` 函数可以终止/取消初始化/释放由 `start_select()` 设置的环境。

备注： 在少数情况下，在调用 `end_select()` 之前可能并没有调用过 `start_select()`。因此 `end_select()` 的实现必须在该情况下返回错误而不能崩溃。

如需获取更多信息，请参考 esp_driver_uart/src/uart_vfs.c 中 UART 外设的 VFS 驱动，尤其是函数 *`uart_vfs_dev_register()`*、`uart_start_select()` 和 `uart_end_select()`。

请参考以下示例，查看如何使用 VFS 文件描述符调用 `select()`：

- peripherals/uart/uart_select
- system/select

套接字 VFS 驱动 套接字 VFS 驱动会使用自实现的 `socket_select()` 函数，在读取/写入/错误条件时，非套接字 VFS 驱动会通知该函数。

可通过定义以下函数注册套接字 VFS 驱动：

```
// In definition of esp_vfs_t:
    .socket_select = &lwip_select,
    .get_socket_select_semaphore = &lwip_get_socket_select_semaphore,
    .stop_socket_select = &lwip_stop_socket_select,
    .stop_socket_select_isr = &lwip_stop_socket_select_isr,
// ... other members initialized
```

函数 socket_select() 是套接字驱动对 select() 的内部实现。该函数只对套接字 VFS 驱动的文件描述符起作用。

get_socket_select_semaphore() 返回信号对象 (semaphore)，用于非套接字驱动程序中，以终止 socket_select() 的等待。

stop_socket_select() 通过传递 get_socket_select_semaphore() 函数返回的对象来终止 socket_select() 函数的等待。

stop_socket_select_isr() 与 stop_socket_select() 的作用相似，但是前者可在 ISR 中使用。

请参考 lwip/port/esp32xx/vfs_lwip.c 以了解使用 LWIP 的套接字驱动参考实现。

备注： 如果 select() 用于套接字文件描述符，可以禁用*CONFIG_VFS_SUPPORT_SELECT* 选项来减少代码量，提高性能。不要在 select() 调用过程中更改套接字驱动，否则会出现一些未定义行为。

路径

已注册的 FS 驱动程序均有一个路径前缀与之关联，此路径前缀即为分区的挂载点。

如果挂载点中嵌套了其他挂载点，则在打开文件时使用具有最长匹配路径前缀的挂载点。例如，假设以下文件系统已在 VFS 中注册：

- 在 /data 下注册 FS 驱动程序 1
- 在 /data/static 下注册 FS 驱动程序 2

那么：

- 打开 /data/log.txt 会调用驱动程序 FS 1；
- 打开 /data/static/index.html 需调用 FS 驱动程序 2；
- 即便 FS 驱动程序 2 中没有 /index.html，也不会在 FS 驱动程序 1 中查找 /static/index.html。

挂载点名称必须以路径分隔符 (/) 开头，且分隔符后至少包含一个字符。但在以下情况中，VFS 同样支持空的挂载点名称：1. 应用程序需要提供一个”最后方案“下使用的文件系统；2. 应用程序需要同时覆盖 VFS 功能。如果没有与路径匹配的前缀，就会使用到这种文件系统。

VFS 不会对路径中的点 (.) 进行特殊处理，也不会将 .. 视为对父目录的引用。在上述示例中，使用 /data/static/../log.txt 路径不会调用 FS 驱动程序 1 打开 /log.txt。特定的 FS 驱动程序（如 FATFS）可能以不同的方式处理文件名中的点。

执行打开文件操作时，FS 驱动程序仅得到文件的相对路径（挂载点前缀已经被去除）：

1. 以 /data 为路径前缀注册 myfs 驱动；
2. 应用程序调用 fopen("/data/config.json", ...)；
3. VFS 调用 myfs_open("/config.json", ...)；
4. myfs 驱动打开 /config.json 文件。

VFS 对文件路径长度没有限制，但文件系统路径前缀受 ESP_VFS_PATH_MAX 限制，即路径前缀上限为 ESP_VFS_PATH_MAX。各个文件系统驱动则可能会对自己的文件名长度设置一些限制。

文件描述符

文件描述符是一组很小的正整数，从 0 到 FD_SETSIZE - 1，FD_SETSIZE 定义在 sys/select.h。最大文件描述符由 CONFIG_LWIP_MAX_SOCKETS 定义，且为套接字保留。VFS 中包含一个名为 s_fd_table 的查找表，用于将全局文件描述符映射至 s_vfs 数组中注册的 VFS 驱动索引。

标准 IO 流 (stdin, stdout, stderr)

如果 menuconfig 中 UART for console output 选项没有设置为 None，则 stdin、stdout 和 stderr 将默认从 UART 读取或写入。UART0 或 UART1 可用作标准 IO。默认情况下，UART0 使用 115200 波特率，TX 管脚为 GPIO1，RX 管脚为 GPIO3。上述参数可以在 menuconfig 中更改。

对 stdout 或 stderr 执行写入操作将会向 UART 发送 FIFO 发送字符，对 stdin 执行读取操作则会从 UART 接收 FIFO 中取出字符。

默认情况下，VFS 使用简单的函数对 UART 进行读写操作。在所有数据放进 UART FIFO 之前，写操作将处于 busy-wait 状态，读操处于非阻塞状态，仅返回 FIFO 中已有数据。由于读操作为非阻塞，高层级 C 库函数调用（如 `fscanf("%d\n", &var);`）可能获取不到所需结果。

如果应用程序使用 UART 驱动，则可以调用*uart_vfs_dev_use_driver()* 函数来指导 VFS 使用驱动中断、读写阻塞功能等，也可以调用*uart_vfs_dev_use_nonblocking()* 来恢复非阻塞函数。

VFS 还为输入和输出提供换行符转换功能（可选）。多数应用程序在程序内部发送或接收以 LF ("n") 结尾的行，但不同的终端程序可能需要不同的换行符，比如 CR 或 CRLF。应用程序可以通过 menuconfig 或者调用*uart_vfs_dev_port_set_rx_line_endings()* 和*uart_vfs_dev_port_set_tx_line_endings()* 为输入输出配置换行符。

标准流和 FreeRTOS 任务 stdin、stdout 和 stderr 的 FILE 对象在所有 FreeRTOS 任务之间共享，指向这些对象的指针分别存储在每个任务的 struct _reent 中。

预处理器把如下代码解释为 `fprintf(__getreent()->_stderr, "42\n");`：

```
fprintf(stderr, "42\n");
```

其中 __getreent() 函数将为每个任务返回一个指向 newlib libc 中 struct _reent 的指针。每个任务的 TCB 均拥有一个 struct _reent 结构体，任务初始化后，struct _reent 结构体中的 _stdin、_stdout 和 _stderr 将会被赋予 _GLOBAL_REENT 中 _stdin、_stdout 和 _stderr 的值，_GLOBAL_REENT 即为 FreeRTOS 启动之前所用结构体。

这样设计带来的结果是：

- 允许设置给定任务的 stdin、stdout 和 stderr，而不影响其他任务，例如通过 `stdin = fopen("/dev/uart/1", "r");`
- 但使用 fclose 关闭默认 stdin、stdout 或 stderr 将同时关闭相应的 FILE 流对象，因此会影响其他任务；
- 如需更改新任务的默认 stdin、stdout 和 stderr 流，请在创建新任务之前修改 _GLOBAL_REENT->_stdin (_stdout、_stderr)。

eventfd()

eventfd() 是一个很强大的工具，可以循环通知基于 select() 的自定义事件。在 ESP-IDF 中，eventfd() 的实现大体上与 man(2) eventfd 中的描述相同，主要区别如下：

- 在调用 eventfd() 之前必须先调用 esp_vfs_eventfd_register()；
- 标志中没有 EFD_CLOEXEC、EFD_NONBLOCK 和 EFD_SEMAPHORE 选项；
- EFD_SUPPORT_ISR 选项已经被添加到标志中。在中断处理程序中读取和写入 eventfd 需要这个标志。

注意，用 EFD_SUPPORT_ISR 创建 eventfd 将导致在读取、写入文件时，以及在设置这个文件的 select() 开始和结束时，暂时禁用中断。

API 参考

Header File

- components/vfs/include/esp_vfs.h
- This header file can be included with:

```
#include "esp_vfs.h"
```

- This header file is a part of the API provided by the `vfs` component. To declare that your component depends on `vfs`, add the following to your CMakeLists.txt:

```
REQUIRES vfs
```

or

```
PRIV_REQUIRES vfs
```

Functions

ssize_t **esp_vfs_write**(struct _reent *r, int fd, const void *data, size_t size)

These functions are to be used in newlib syscall table. They will be called by newlib when it needs to use any of the syscalls.

off_t **esp_vfs_lseek**(struct _reent *r, int fd, off_t size, int mode)

ssize_t **esp_vfs_read**(struct _reent *r, int fd, void *dst, size_t size)

int **esp_vfs_open**(struct _reent *r, const char *path, int flags, int mode)

int **esp_vfs_close**(struct _reent *r, int fd)

int **esp_vfs_fstat**(struct _reent *r, int fd, struct stat *st)

int **esp_vfs_stat**(struct _reent *r, const char *path, struct stat *st)

int **esp_vfs_link**(struct _reent *r, const char *n1, const char *n2)

int **esp_vfs_unlink**(struct _reent *r, const char *path)

int **esp_vfs_rename**(struct _reent *r, const char *src, const char *dst)

int **esp_vfs_utime**(const char *path, const struct utimbuf *times)

esp_err_t **esp_vfs_register**(const char *base_path, const *esp_vfs_t* *vfs, void *ctx)

Register a virtual filesystem for given path prefix.

参数

- **base_path** -- file path prefix associated with the filesystem. Must be a zero-terminated C string, may be empty. If not empty, must be up to ESP_VFS_PATH_MAX characters long, and at least 2 characters long. Name must start with a ”/” and must not end with ”/”. For example, ”/data” or ”/dev/spi” are valid. These VFSes would then be called to handle file paths such as ”/data/myfile.txt” or ”/dev/spi/0”. In the special case of an empty base_path, a ”fallback” VFS is registered. Such VFS will handle paths which are not matched by any other registered VFS.
- **vfs** -- Pointer to *esp_vfs_t*, a structure which maps syscalls to the filesystem driver functions. VFS component doesn’t assume ownership of this pointer.
- **ctx** -- If vfs->flags has ESP_VFS_FLAG_CONTEXT_PTR set, a pointer which should be passed to VFS functions. Otherwise, NULL.

返回 ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered.

esp_err_t **esp_vfs_register_fd_range**(const *esp_vfs_t* *vfs, void *ctx, int min_fd, int max_fd)

Special case function for registering a VFS that uses a method other than open() to open new file descriptors from the interval <min_fd; max_fd).

This is a special-purpose function intended for registering LWIP sockets to VFS.

参数

- **vfs** -- Pointer to *esp_vfs_t*. Meaning is the same as for esp_vfs_register().
- **ctx** -- Pointer to context structure. Meaning is the same as for esp_vfs_register().
- **min_fd** -- The smallest file descriptor this VFS will use.

- **max_fd** -- Upper boundary for file descriptors this VFS will use (the biggest file descriptor plus one).

返回 ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered, ESP_ERR_INVALID_ARG if the file descriptor boundaries are incorrect.

esp_err_t **esp_vfs_register_with_id**(const *esp_vfs_t* *vfs, void *ctx, *esp_vfs_id_t* *vfs_id)

Special case function for registering a VFS that uses a method other than open() to open new file descriptors. In comparison with esp_vfs_register_fd_range, this function doesn't pre-registers an interval of file descriptors. File descriptors can be registered later, by using esp_vfs_register_fd.

参数

- **vfs** -- Pointer to *esp_vfs_t*. Meaning is the same as for esp_vfs_register().
- **ctx** -- Pointer to context structure. Meaning is the same as for esp_vfs_register().
- **vfs_id** -- Here will be written the VFS ID which can be passed to esp_vfs_register_fd for registering file descriptors.

返回 ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered, ESP_ERR_INVALID_ARG if the file descriptor boundaries are incorrect.

esp_err_t **esp_vfs_unregister**(const char *base_path)

Unregister a virtual filesystem for given path prefix

参数 **base_path** -- file prefix previously used in esp_vfs_register call

返回 ESP_OK if successful, ESP_ERR_INVALID_STATE if VFS for given prefix hasn't been registered

esp_err_t **esp_vfs_unregister_with_id**(*esp_vfs_id_t* vfs_id)

Unregister a virtual filesystem with the given index

参数 **vfs_id** -- The VFS ID returned by esp_vfs_register_with_id

返回 ESP_OK if successful, ESP_ERR_INVALID_STATE if VFS for the given index hasn't been registered

esp_err_t **esp_vfs_register_fd**(*esp_vfs_id_t* vfs_id, int *fd)

Special function for registering another file descriptor for a VFS registered by esp_vfs_register_with_id. This function should only be used to register permanent file descriptors (socket fd) that are not removed after being closed.

参数

- **vfs_id** -- VFS identificator returned by esp_vfs_register_with_id.
- **fd** -- The registered file descriptor will be written to this address.

返回 ESP_OK if the registration is successful, ESP_ERR_NO_MEM if too many file descriptors are registered, ESP_ERR_INVALID_ARG if the arguments are incorrect.

esp_err_t **esp_vfs_register_fd_with_local_fd**(*esp_vfs_id_t* vfs_id, int local_fd, bool permanent, int *fd)

Special function for registering another file descriptor with given local_fd for a VFS registered by esp_vfs_register_with_id.

参数

- **vfs_id** -- VFS identificator returned by esp_vfs_register_with_id.
- **local_fd** -- The fd in the local vfs. Passing -1 will set the local fd as the (*fd) value.
- **permanent** -- Whether the fd should be treated as permannet (not removed after close())
- **fd** -- The registered file descriptor will be written to this address.

返回 ESP_OK if the registration is successful, ESP_ERR_NO_MEM if too many file descriptors are registered, ESP_ERR_INVALID_ARG if the arguments are incorrect.

esp_err_t **esp_vfs_unregister_fd**(*esp_vfs_id_t* vfs_id, int fd)

Special function for unregistering a file descriptor belonging to a VFS registered by esp_vfs_register_with_id.

参数

- **vfs_id** -- VFS identificator returned by esp_vfs_register_with_id.
- **fd** -- File descriptor which should be unregistered.

返回 ESP_OK if the registration is successful, ESP_ERR_INVALID_ARG if the arguments are incorrect.

int **esp_vfs_select**(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout)

Synchronous I/O multiplexing which implements the functionality of POSIX select() for VFS.

参数

- **nfds** -- Specifies the range of descriptors which should be checked. The first nfds descriptors will be checked in each set.
- **readfds** -- If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for being ready to read, and on output indicates which descriptors are ready to read.
- **writefds** -- If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for being ready to write, and on output indicates which descriptors are ready to write.
- **errorfds** -- If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for error conditions, and on output indicates which descriptors have error conditions.
- **timeout** -- If not NULL, then points to timeval structure which specifies the time period after which the functions should time-out and return. If it is NULL, then the function will not time-out. Note that the timeout period is rounded up to the system tick and incremented by one.

返回 The number of descriptors set in the descriptor sets, or -1 when an error (specified by errno) have occurred.

void **esp_vfs_select_triggered**(*esp_vfs_select_sem_t* sem)

Notification from a VFS driver about a read/write/error condition.

This function is called when the VFS driver detects a read/write/error condition as it was requested by the previous call to start_select.

参数 **sem** -- semaphore structure which was passed to the driver by the start_select call

void **esp_vfs_select_triggered_isr**(*esp_vfs_select_sem_t* sem, BaseType_t *woken)

Notification from a VFS driver about a read/write/error condition (ISR version)

This function is called when the VFS driver detects a read/write/error condition as it was requested by the previous call to start_select.

参数

- **sem** -- semaphore structure which was passed to the driver by the start_select call
- **woken** -- is set to pdTRUE if the function wakes up a task with higher priority

ssize_t **esp_vfs_pread**(int fd, void *dst, size_t size, off_t offset)

Implements the VFS layer of POSIX pread()

参数

- **fd** -- File descriptor used for read
- **dst** -- Pointer to the buffer where the output will be written
- **size** -- Number of bytes to be read
- **offset** -- Starting offset of the read

返回 A positive return value indicates the number of bytes read. -1 is return on failure and errno is set accordingly.

ssize_t **esp_vfs_pwrite**(int fd, const void *src, size_t size, off_t offset)

Implements the VFS layer of POSIX pwrite()

参数

- **fd** -- File descriptor used for write
- **src** -- Pointer to the buffer from where the output will be read
- **size** -- Number of bytes to write
- **offset** -- Starting offset of the write

返回 A positive return value indicates the number of bytes written. -1 is return on failure and errno is set accordingly.

void **esp_vfs_dump_fds**(FILE *fp)

Dump the existing VFS FDs data to FILE* fp.

Dump the FDs in the format:

```
        <VFS Path Prefix>-<FD seen by App>-<FD seen by driver>

   where:
    VFS Path Prefix   : file prefix used in the esp_vfs_register call
    FD seen by App    : file descriptor returned by the vfs to the application␣
↪for the path prefix
    FD seen by driver : file descriptor used by the driver for the same file␣
↪prefix.
```

参数 **fp** -- File descriptor where data will be dumped

Structures

struct **esp_vfs_select_sem_t**

VFS semaphore type for select()

Public Members

bool **is_sem_local**

type of "sem" is SemaphoreHandle_t when true, defined by socket driver otherwise

void ***sem**

semaphore instance

struct **esp_vfs_t**

VFS definition structure.

This structure should be filled with pointers to corresponding FS driver functions.

VFS component will translate all FDs so that the filesystem implementation sees them starting at zero. The caller sees a global FD which is prefixed with an pre-filesystem-implementation.

Some FS implementations expect some state (e.g. pointer to some structure) to be passed in as a first argument. For these implementations, populate the members of this structure which have _p suffix, set flags member to ESP_VFS_FLAG_CONTEXT_PTR and provide the context pointer to esp_vfs_register function. If the implementation doesn't use this extra argument, populate the members without _p suffix and set flags member to ESP_VFS_FLAG_DEFAULT.

If the FS driver doesn't provide some of the functions, set corresponding members to NULL.

Public Members

int **flags**

ESP_VFS_FLAG_CONTEXT_PTR and/or ESP_VFS_FLAG_READONLY_FS or ESP_VFS_FLAG_DEFAULT

ssize_t (***write_p**)(void *p, int fd, const void *data, size_t size)

Write with context pointer

ssize_t (***write**)(int fd, const void *data, size_t size)

Write without context pointer

off_t (***lseek_p**)(void *p, int fd, off_t size, int mode)

Seek with context pointer

off_t (***lseek**)(int fd, off_t size, int mode)

Seek without context pointer

ssize_t (***read_p**)(void *ctx, int fd, void *dst, size_t size)

Read with context pointer

ssize_t (***read**)(int fd, void *dst, size_t size)

Read without context pointer

ssize_t (***pread_p**)(void *ctx, int fd, void *dst, size_t size, off_t offset)

pread with context pointer

ssize_t (***pread**)(int fd, void *dst, size_t size, off_t offset)

pread without context pointer

ssize_t (***pwrite_p**)(void *ctx, int fd, const void *src, size_t size, off_t offset)

pwrite with context pointer

ssize_t (***pwrite**)(int fd, const void *src, size_t size, off_t offset)

pwrite without context pointer

int (***open_p**)(void *ctx, const char *path, int flags, int mode)

open with context pointer

int (***open**)(const char *path, int flags, int mode)

open without context pointer

int (***close_p**)(void *ctx, int fd)

close with context pointer

int (***close**)(int fd)

close without context pointer

int (***fstat_p**)(void *ctx, int fd, struct *stat* *st)

fstat with context pointer

int (***fstat**)(int fd, struct *stat* *st)

fstat without context pointer

int (***stat_p**)(void *ctx, const char *path, struct *stat* *st)

stat with context pointer

int (***stat**)(const char *path, struct *stat* *st)
stat without context pointer

int (***link_p**)(void *ctx, const char *n1, const char *n2)
link with context pointer

int (***link**)(const char *n1, const char *n2)
link without context pointer

int (***unlink_p**)(void *ctx, const char *path)
unlink with context pointer

int (***unlink**)(const char *path)
unlink without context pointer

int (***rename_p**)(void *ctx, const char *src, const char *dst)
rename with context pointer

int (***rename**)(const char *src, const char *dst)
rename without context pointer

DIR *(***opendir_p**)(void *ctx, const char *name)
opendir with context pointer

DIR *(***opendir**)(const char *name)
opendir without context pointer

struct dirent *(***readdir_p**)(void *ctx, DIR *pdir)
readdir with context pointer

struct dirent *(***readdir**)(DIR *pdir)
readdir without context pointer

int (***readdir_r_p**)(void *ctx, DIR *pdir, struct dirent *entry, struct dirent **out_dirent)
readdir_r with context pointer

int (***readdir_r**)(DIR *pdir, struct dirent *entry, struct dirent **out_dirent)
readdir_r without context pointer

long (***telldir_p**)(void *ctx, DIR *pdir)
telldir with context pointer

long (***telldir**)(DIR *pdir)
telldir without context pointer

void (***seekdir_p**)(void *ctx, DIR *pdir, long offset)
seekdir with context pointer

void (***seekdir**)(DIR *pdir, long offset)
seekdir without context pointer

int (***closedir_p**)(void *ctx, DIR *pdir)
closedir with context pointer

int (***closedir**)(DIR *pdir)
closedir without context pointer

int (***mkdir_p**)(void *ctx, const char *name, mode_t mode)
mkdir with context pointer

int (***mkdir**)(const char *name, mode_t mode)
mkdir without context pointer

int (***rmdir_p**)(void *ctx, const char *name)
rmdir with context pointer

int (***rmdir**)(const char *name)
rmdir without context pointer

int (***fcntl_p**)(void *ctx, int fd, int cmd, int arg)
fcntl with context pointer

int (***fcntl**)(int fd, int cmd, int arg)
fcntl without context pointer

int (***ioctl_p**)(void *ctx, int fd, int cmd, va_list args)
ioctl with context pointer

int (***ioctl**)(int fd, int cmd, va_list args)
ioctl without context pointer

int (***fsync_p**)(void *ctx, int fd)
fsync with context pointer

int (***fsync**)(int fd)
fsync without context pointer

int (***access_p**)(void *ctx, const char *path, int amode)
access with context pointer

int (***access**)(const char *path, int amode)
access without context pointer

int (***truncate_p**)(void *ctx, const char *path, off_t length)
truncate with context pointer

int (***truncate**)(const char *path, off_t length)
truncate without context pointer

int (***ftruncate_p**)(void *ctx, int fd, off_t length)
ftruncate with context pointer

int (***ftruncate**)(int fd, off_t length)
ftruncate without context pointer

int (***utime_p**)(void *ctx, const char *path, const struct utimbuf *times)
utime with context pointer

int (***utime**)(const char *path, const struct utimbuf *times)
utime without context pointer

int (***tcsetattr_p**)(void *ctx, int fd, int optional_actions, const struct termios *p)
tcsetattr with context pointer

int (***tcsetattr**)(int fd, int optional_actions, const struct termios *p)
tcsetattr without context pointer

int (***tcgetattr_p**)(void *ctx, int fd, struct termios *p)
tcgetattr with context pointer

int (***tcgetattr**)(int fd, struct termios *p)
tcgetattr without context pointer

int (***tcdrain_p**)(void *ctx, int fd)
tcdrain with context pointer

int (***tcdrain**)(int fd)
tcdrain without context pointer

int (***tcflush_p**)(void *ctx, int fd, int select)
tcflush with context pointer

int (***tcflush**)(int fd, int select)
tcflush without context pointer

int (***tcflow_p**)(void *ctx, int fd, int action)
tcflow with context pointer

int (***tcflow**)(int fd, int action)
tcflow without context pointer

pid_t (***tcgetsid_p**)(void *ctx, int fd)
tcgetsid with context pointer

pid_t (*`tcgetsid`)(int fd)

tcgetsid without context pointer

int (*`tcsendbreak_p`)(void *ctx, int fd, int duration)

tcsendbreak with context pointer

int (*`tcsendbreak`)(int fd, int duration)

tcsendbreak without context pointer

esp_err_t (*`start_select`)(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, *esp_vfs_select_sem_t* sem, void **end_select_args)

start_select is called for setting up synchronous I/O multiplexing of the desired file descriptors in the given VFS

int (*`socket_select`)(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout)

socket select function for socket FDs with the functionality of POSIX select(); this should be set only for the socket VFS

void (*`stop_socket_select`)(void *sem)

called by VFS to interrupt the socket_select call when select is activated from a non-socket VFS driver; set only for the socket driver

void (*`stop_socket_select_isr`)(void *sem, BaseType_t *woken)

stop_socket_select which can be called from ISR; set only for the socket driver

void *(*`get_socket_select_semaphore`)(void)

end_select is called to stop the I/O multiplexing and deinitialize the environment created by start_select for the given VFS

esp_err_t (*`end_select`)(void *end_select_args)

get_socket_select_semaphore returns semaphore allocated in the socket driver; set only for the socket driver

Macros

`MAX_FDS`

Maximum number of (global) file descriptors.

`ESP_VFS_PATH_MAX`

Maximum length of path prefix (not including zero terminator)

`ESP_VFS_FLAG_DEFAULT`

Default value of flags member in *esp_vfs_t* structure.

`ESP_VFS_FLAG_CONTEXT_PTR`

Flag which indicates that FS needs extra context pointer in syscalls.

`ESP_VFS_FLAG_READONLY_FS`

Flag which indicates that FS is located on read-only partition.

Type Definitions

typedef int **esp_vfs_id_t**

Header File

- components/vfs/include/esp_vfs_dev.h
- This header file can be included with:

```
#include "esp_vfs_dev.h"
```

- This header file is a part of the API provided by the `vfs` component. To declare that your component depends on `vfs`, add the following to your CMakeLists.txt:

```
REQUIRES vfs
```

or

```
PRIV_REQUIRES vfs
```

Functions

void **esp_vfs_dev_uart_register**(void)

void **esp_vfs_dev_uart_use_nonblocking**(int uart_num)

void **esp_vfs_dev_uart_use_driver**(int uart_num)

int **esp_vfs_dev_uart_port_set_rx_line_endings**(int uart_num, esp_line_endings_t mode)

int **esp_vfs_dev_uart_port_set_tx_line_endings**(int uart_num, esp_line_endings_t mode)

void **esp_vfs_dev_uart_set_rx_line_endings**(esp_line_endings_t mode)

Set the line endings expected to be received on UART.

This specifies the conversion between line endings received on UART and newlines ('

', LF) passed into stdin:

- ESP_LINE_ENDINGS_CRLF: convert CRLF to LF
- ESP_LINE_ENDINGS_CR: convert CR to LF
- ESP_LINE_ENDINGS_LF: no modification

备注： this function is not thread safe w.r.t. reading from UART

参数 **mode** -- line endings expected on UART

void **esp_vfs_dev_uart_set_tx_line_endings**(esp_line_endings_t mode)

Set the line endings to sent to UART.

This specifies the conversion between newlines ('

', LF) on stdout and line endings sent over UART:

- ESP_LINE_ENDINGS_CRLF: convert LF to CRLF
- ESP_LINE_ENDINGS_CR: convert LF to CR

- ESP_LINE_ENDINGS_LF: no modification

备注： this function is not thread safe w.r.t. writing to UART

参数 **mode** -- line endings to send to UART

void **esp_vfs_usb_serial_jtag_use_driver**(void)

set VFS to use USB-SERIAL-JTAG driver for reading and writing

备注： application must configure USB-SERIAL-JTAG driver before calling these functions With these functions, read and write are blocking and interrupt-driven.

void **esp_vfs_usb_serial_jtag_use_nonblocking**(void)

set VFS to use simple functions for reading and writing UART Read is non-blocking, write is busy waiting until TX FIFO has enough space. These functions are used by default.

Header File

- components/esp_driver_uart/include/driver/uart_vfs.h
- This header file can be included with:

```
#include "driver/uart_vfs.h"
```

- This header file is a part of the API provided by the `esp_driver_uart` component. To declare that your component depends on `esp_driver_uart`, add the following to your CMakeLists.txt:

```
REQUIRES esp_driver_uart
```

or

```
PRIV_REQUIRES esp_driver_uart
```

Functions

void **uart_vfs_dev_register**(void)

Add /dev/uart virtual filesystem driver.

This function is called from startup code to enable serial output

int **uart_vfs_dev_port_set_rx_line_endings**(int uart_num, esp_line_endings_t mode)

Set the line endings expected to be received on specified UART.

This specifies the conversion between line endings received on UART and newlines ('

', LF) passed into stdin:

- ESP_LINE_ENDINGS_CRLF: convert CRLF to LF
- ESP_LINE_ENDINGS_CR: convert CR to LF
- ESP_LINE_ENDINGS_LF: no modification

备注： this function is not thread safe w.r.t. reading from UART

参数

- **uart_num** -- the UART number
- **mode** -- line endings to send to UART

返回 0 if succeeded, or -1 when an error (specified by errno) have occurred.

int **uart_vfs_dev_port_set_tx_line_endings**(int uart_num, esp_line_endings_t mode)

Set the line endings to sent to specified UART.

This specifies the conversion between newlines ('

', LF) on stdout and line endings sent over UART:

- ESP_LINE_ENDINGS_CRLF: convert LF to CRLF
- ESP_LINE_ENDINGS_CR: convert LF to CR
- ESP_LINE_ENDINGS_LF: no modification

备注: this function is not thread safe w.r.t. writing to UART

参数

- **uart_num** -- the UART number
- **mode** -- line endings to send to UART

返回 0 if succeeded, or -1 when an error (specified by errno) have occurred.

void **uart_vfs_dev_use_nonblocking**(int uart_num)

set VFS to use simple functions for reading and writing UART

Read is non-blocking, write is busy waiting until TX FIFO has enough space. These functions are used by default.

参数 **uart_num** -- UART peripheral number

void **uart_vfs_dev_use_driver**(int uart_num)

set VFS to use UART driver for reading and writing

备注: Application must configure UART driver before calling these functions With these functions, read and write are blocking and interrupt-driven.

参数 **uart_num** -- UART peripheral number

Header File

- components/vfs/include/esp_vfs_eventfd.h
- This header file can be included with:

```
#include "esp_vfs_eventfd.h"
```

- This header file is a part of the API provided by the `vfs` component. To declare that your component depends on `vfs`, add the following to your CMakeLists.txt:

```
REQUIRES vfs
```

or

```
PRIV_REQUIRES vfs
```

Functions

esp_err_t **esp_vfs_eventfd_register**(const *esp_vfs_eventfd_config_t* *config)

Registers the event vfs.

返回 ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered.

esp_err_t **esp_vfs_eventfd_unregister**(void)

Unregisters the event vfs.

返回 ESP_OK if successful, ESP_ERR_INVALID_STATE if VFS for given prefix hasn't been registered

int **eventfd**(unsigned int initval, int flags)

Structures

struct **esp_vfs_eventfd_config_t**

Eventfd vfs initialization settings.

Public Members

size_t **max_fds**

The maximum number of eventfds supported

Macros

EFD_SUPPORT_ISR

ESP_VFS_EVENTD_CONFIG_DEFAULT()

2.8.11 磨损均衡 API

概述

ESP32-P4 所使用的 flash，特别是 SPI flash，多数具备扇区结构，且每个扇区仅允许有限次数的擦除/修改操作。为了避免过度使用某一扇区，乐鑫提供了磨损均衡组件，无需用户介入即可帮助用户均衡各个扇区之间的磨损。

磨损均衡组件包含了通过分区组件对外部 SPI flash 进行数据读取、写入、擦除和存储器映射相关的 API 函数。磨损均衡组件还具有软件上更高级别的 API 函数，与*FAT 文件系统* 协同工作。

磨损均衡组件与 FAT 文件系统组件共用 FAT 文件系统的扇区，扇区大小为 4096 字节，是标准 flash 扇区的大小。在这种模式下，磨损均衡组件性能达到最佳，但需要在 RAM 中占用更多内存。

为了节省内存，磨损均衡组件还提供了另外两种模式，均使用 512 字节大小的扇区：

- **性能模式**：先将数据保存在 RAM 中，擦除扇区，然后将数据存储回 flash。如果设备在扇区擦写过程中突然断电，则整个扇区（4096 字节）数据将全部丢失。
- **安全模式**：数据先保存在 flash 中空余扇区，擦除扇区后，数据即存储回去。如果设备断电，上电后可立即恢复数据。

设备默认设置如下：

- 定义扇区大小为 512 字节
- 默认使用性能模式

您可以使用配置菜单更改设置。

磨损均衡组件不会将数据缓存在 RAM 中。写入和擦除函数直接修改 flash，函数返回后，flash 即完成修改。

磨损均衡访问 API

处理 flash 数据常用的 API 如下所示：

- `wl_mount` - 为指定分区挂载并初始化磨损均衡模块
- `wl_unmount` - 卸载分区并释放磨损均衡模块
- `wl_erase_range` - 擦除 flash 中指定的地址范围
- `wl_write` - 将数据写入分区
- `wl_read` - 从分区读取数据
- `wl_size` - 返回可用内存的大小（以字节为单位）
- `wl_sector_size` - 返回一个扇区的大小

请尽量避免直接使用原始磨损均衡函数，建议您使用文件系统特定的函数。

内存大小

内存大小是根据分区参数在磨损均衡模块中计算所得，由于模块使用 flash 部分扇区存储内部数据，因此计算所得内存大小有少许偏差。

另请参阅

- *FAT 文件系统*
- 分区表

应用示例

storage/wear_levelling 中提供了一款磨损均衡驱动与 FatFs 库结合使用的示例。该示例初始化磨损均衡驱动，挂载 FAT 文件系统分区，并使用 POSIX（可移植操作系统接口）和 C 库 API 从中写入和读取数据。如需了解更多信息，请参考 storage/wear_levelling/README.md。

高级 API 参考

头文件

- fatfs/vfs/esp_vfs_fat.h

有关高级磨损均衡函数*esp_vfs_fat_spiflash_mount_rw_wl()*、*esp_vfs_fat_spiflash_unmount_rw_wl()* 和结构体*esp_vfs_fat_mount_config_t* 的详细内容，请参见*FAT 文件系统*。

中层 API 参考

Header File

- components/wear_levelling/include/wear_levelling.h
- This header file can be included with:

```
#include "wear_levelling.h"
```

- This header file is a part of the API provided by the `wear_levelling` component. To declare that your component depends on `wear_levelling`, add the following to your CMakeLists.txt:

```
REQUIRES wear_levelling
```

or

```
PRIV_REQUIRES wear_levelling
```

Functions

esp_err_t **wl_mount**(const *esp_partition_t* *partition, *wl_handle_t* *out_handle)

Mount WL for defined partition.

参数

- **partition** -- that will be used for access
- **out_handle** -- handle of the WL instance

返回

- ESP_OK, if the WL allocation is successful;
- ESP_ERR_INVALID_ARG, if the arguments for WL configuration are not valid;
- ESP_ERR_NO_MEM, if the WL allocation fails because of insufficient memory;

esp_err_t **wl_unmount**(*wl_handle_t* handle)

Unmount WL for defined partition.

参数 **handle** -- WL partition handle

返回

- ESP_OK, if the operation is successful;
- or one of error codes from lower-level flash driver.

esp_err_t **wl_erase_range**(*wl_handle_t* handle, size_t start_addr, size_t size)

Erase part of the WL storage.

参数

- **handle** -- WL handle that are related to the partition
- **start_addr** -- Address from where erase operation should start. Must be aligned to the result of function wl_sector_size(...).
- **size** -- Size of the range which should be erased, in bytes. Must be divisible by the result of function wl_sector_size(...)..

返回

- ESP_OK, if the given range was erased successfully;
- ESP_ERR_INVALID_ARG, if iterator or dst are NULL;
- ESP_ERR_INVALID_SIZE, if erase would go out of bounds of the partition;
- or one of error codes from lower-level flash driver.

esp_err_t **wl_write**(*wl_handle_t* handle, size_t dest_addr, const void *src, size_t size)

Write data to the WL storage.

Before writing data to flash, corresponding region of flash needs to be erased. This can be done using wl_erase_range function.

备注: Prior to writing to WL storage, make sure it has been erased with wl_erase_range call.

参数

- **handle** -- WL handle corresponding to the WL partition
- **dest_addr** -- Address where the data should be written, relative to the beginning of the partition.
- **src** -- Pointer to the source buffer. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- **size** -- Size of data to be written, in bytes.

返回

- ESP_OK, if data was written successfully;
- ESP_ERR_INVALID_ARG, if dst_offset exceeds partition size;

- ESP_ERR_INVALID_SIZE, if write would go out of bounds of the partition;
- or one of error codes from lower-level flash driver.

esp_err_t **wl_read**(*wl_handle_t* handle, size_t src_addr, void *dest, size_t size)

Read data from the WL storage.

参数

- **handle** -- WL module instance that was initialized before
- **dest** -- Pointer to the buffer where data should be stored. The Pointer must be non-NULL and the buffer must be at least 'size' bytes long.
- **src_addr** -- Address of the data to be read, relative to the beginning of the partition.
- **size** -- Size of data to be read, in bytes.

返回

- ESP_OK, if data was read successfully;
- ESP_ERR_INVALID_ARG, if src_offset exceeds partition size;
- ESP_ERR_INVALID_SIZE, if read would go out of bounds of the partition;
- or one of error codes from lower-level flash driver.

size_t **wl_size**(*wl_handle_t* handle)

Get the actual flash size in use for the WL storage partition.

参数 **handle** -- WL module handle that was initialized before
返回 usable size, in bytes

size_t **wl_sector_size**(*wl_handle_t* handle)

Get sector size of the WL instance.

参数 **handle** -- WL module handle that was initialized before
返回 sector size, in bytes

Macros

WL_INVALID_HANDLE

Type Definitions

typedef int32_t **wl_handle_t**

wear levelling handle

此部分 API 代码示例存放在 ESP-IDF 示例项目的 storage 目录下。

2.9 系统 API

2.9.1 应用程序镜像格式

应用程序镜像结构

应用程序镜像包含以下内容：

1. *esp_image_header_t* 结构体描述了 SPI flash 的模式和内存段的数量。
2. *esp_image_segment_header_t* 结构体描述了每个段、每个段的长度及其在 ESP32-P4 内存中的位置，此描述后接长度为 `data_len` 的数据。镜像中每个段的数据偏移量的计算方式如下：
   - 0 段的偏移量 = sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*)

- 1 段的偏移量 = 0 段的偏移量 + 0 段长度 + sizeof(*esp_image_segment_header_t*)
- 2 段的偏移量 = 1 段的偏移量 + 1 段长度 + sizeof(*esp_image_segment_header_t*)
- ...

segment_count 字段定义了每个段的数量，存储在*esp_image_header_t* 中。各段段数量不能超过*ESP_IMAGE_MAX_SEGMENTS*。

运行以下命令，获取镜像段列表：

```
esptool.py --chip esp32p4 image_info build/app.bin
```

```
esptool.py v2.3.1
Image version: 1
Entry point: 40080ea4
13 segments

Segment 1: len 0x13ce0 load 0x3f400020 file_offs 0x00000018 SOC_DROM
Segment 2: len 0x00000 load 0x3ff80000 file_offs 0x00013d00 SOC_RTC_DRAM
Segment 3: len 0x00000 load 0x3ff80000 file_offs 0x00013d08 SOC_RTC_DRAM
Segment 4: len 0x028e0 load 0x3ffb0000 file_offs 0x00013d10 DRAM
Segment 5: len 0x00000 load 0x3ffb28e0 file_offs 0x000165f8 DRAM
Segment 6: len 0x00400 load 0x40080000 file_offs 0x00016600 SOC_IRAM
Segment 7: len 0x09600 load 0x40080400 file_offs 0x00016a08 SOC_IRAM
Segment 8: len 0x62e4c load 0x400d0018 file_offs 0x00020010 SOC_IROM
Segment 9: len 0x06cec load 0x40089a00 file_offs 0x00082e64 SOC_IROM
Segment 10: len 0x00000 load 0x400c0000 file_offs 0x00089b58 SOC_RTC_IRAM
Segment 11: len 0x00004 load 0x50000000 file_offs 0x00089b60 SOC_RTC_DATA
Segment 12: len 0x00000 load 0x50000004 file_offs 0x00089b6c SOC_RTC_DATA
Segment 13: len 0x00000 load 0x50000004 file_offs 0x00089b74 SOC_RTC_DATA
Checksum: e8 (valid)
Validation Hash: 407089ca0eae2bbf83b4120979d3354b1c938a49cb7a0c997f240474ef2ec76b␣
↪(valid)
```

应用程序启动时，ESP-IDF 日志中也会包含段的相关信息：

```
I (443) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x13ce0 (␣
↪81120) map
I (489) esp_image: segment 1: paddr=0x00033d08 vaddr=0x3ff80000 size=0x00000 ( 0)␣
↪load
I (530) esp_image: segment 2: paddr=0x00033d10 vaddr=0x3ff80000 size=0x00000 ( 0)␣
↪load
I (571) esp_image: segment 3: paddr=0x00033d18 vaddr=0x3ffb0000 size=0x028e0 (␣
↪10464) load
I (612) esp_image: segment 4: paddr=0x00036600 vaddr=0x3ffb28e0 size=0x00000 ( 0)␣
↪load
I (654) esp_image: segment 5: paddr=0x00036608 vaddr=0x40080000 size=0x00400 (␣
↪1024) load
I (695) esp_image: segment 6: paddr=0x00036a10 vaddr=0x40080400 size=0x09600 (␣
↪38400) load
I (737) esp_image: segment 7: paddr=0x00040018 vaddr=0x400d0018 size=0x62e4c␣
↪(405068) map
I (847) esp_image: segment 8: paddr=0x000a2e6c vaddr=0x40089a00 size=0x06cec (␣
↪27884) load
I (888) esp_image: segment 9: paddr=0x000a9b60 vaddr=0x400c0000 size=0x00000 ( 0)␣
↪load
I (929) esp_image: segment 10: paddr=0x000a9b68 vaddr=0x50000000 size=0x00004 ( 4)␣
↪load
I (971) esp_image: segment 11: paddr=0x000a9b74 vaddr=0x50000004 size=0x00000 ( 0)␣
↪load
I (1012) esp_image: segment 12: paddr=0x000a9b7c vaddr=0x50000004 size=0x00000 (␣
↪0) load
```

有关内存段类型和地址范围的详细信息，请参阅 **ESP32-P4 技术参考手册 > 系统和存储器 > 内部存储器**

[PDF]。

3. 镜像有一个校验和字节，位于最后一个段之后。此字节写在一个十六字节填充边界上，因此应用程序镜像可能需要填充。
4. 如果在*esp_image_header_t* 中设置了 hash_appended 字段，则会附加 SHA256 校验和字段。SHA256 哈希值的计算范围是从第一个字节开始，到这个字段为止。该字段长度为 32 字节。
5. 如果选项*CONFIG_SECURE_SIGNED_APPS_SCHEME* 设置为 ECDSA，那么应用程序镜像将有额外的 68 字节用于 ECDSA 签名，其中包括：
   - 版本号（4 字节）
   - 签名数据（64 字节）
6. 如果选项*CONFIG_SECURE_SIGNED_APPS_SCHEME* 设置为 RSA 或 ECDSA (V2)，则应用程序镜像将有一个额外的签名扇区，大小为 4K 字节。关于此签名扇区格式的更多内容，请参考*Signature Block Format*。

应用程序描述

应用程序二进制文件的 DROM 段从*esp_app_desc_t* 结构体开始，该结构体中包含了用于描述应用程序的特定字段，如下所示：

- magic_word：*esp_app_desc_t* 结构体的魔术词
- secure_version：参见*防回滚*
- version：参见*应用程序版本*[1]
- project_name：通过 PROJECT_NAME 填充[1]
- time 和 date：编译时间和日期
- idf_ver：ESP-IDF 的版本[1]
- app_elf_sha256：包含应用程序 ELF 文件的 sha256 哈希

这个结构体有助于识别通过空中升级 (OTA) 上传的镜像，因为其中包含一个固定的偏移量，大小为 sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*)。一旦设备接收到包含此结构体的第一个段，就能根据其中的充分信息来确定是否应继续更新。

要获取当前运行的应用程序的*esp_app_desc_t* 结构体，请调用*esp_app_get_description()*。

要获取另一个 OTA 分区的*esp_app_desc_t* 结构体，请调用*esp_ota_get_partition_description()*。

向应用程序添加自定义结构体

也可以自定义类似的结构体，并使其相对于镜像起始位置有一个固定的偏移量。

采用以下方式向镜像添加自定义结构体：

```
const __attribute__((section(".rodata_custom_desc"))) esp_custom_app_desc_t custom_
↪app_desc = { ... }
```

自定义结构体的偏移量为 sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*) + sizeof(*esp_app_desc_t*)。

需在 CMakeLists.txt 中添加 target_link_libraries(${COMPONENT_TARGET} "-u custom_app_desc")，确保自定义结构体在未使用时也位于镜像中。

API 参考

Header File

- components/bootloader_support/include/esp_app_format.h
- This header file can be included with:

[1] 最大长度为 32 个字符，其中包括 null 终止符。也就是说，如果 PROJECT_NAME 的长度超过 31 个字符，超出的字符将被忽略。

```
#include "esp_app_format.h"
```

- This header file is a part of the API provided by the `bootloader_support` component. To declare that your component depends on `bootloader_support`, add the following to your CMakeLists.txt:

```
REQUIRES bootloader_support
```

or

```
PRIV_REQUIRES bootloader_support
```

Structures

struct **esp_image_header_t**

Main header of binary image.

Public Members

uint8_t **magic**

Magic word ESP_IMAGE_HEADER_MAGIC

uint8_t **segment_count**

Count of memory segments

uint8_t **spi_mode**

flash read mode (esp_image_spi_mode_t as uint8_t)

uint8_t **spi_speed**

flash frequency (esp_image_spi_freq_t as uint8_t)

uint8_t **spi_size**

flash chip size (esp_image_flash_size_t as uint8_t)

uint32_t **entry_addr**

Entry address

uint8_t **wp_pin**

WP pin when SPI pins set via efuse (read by ROM bootloader, the IDF bootloader uses software to configure the WP pin and sets this field to 0xEE=disabled)

uint8_t **spi_pin_drv**[3]

Drive settings for the SPI flash pins (read by ROM bootloader)

esp_chip_id_t **chip_id**

Chip identification number

uint8_t **min_chip_rev**

Minimal chip revision supported by image After the Major and Minor revision eFuses were introduced into the chips, this field is no longer used. But for compatibility reasons, we keep this field and the data in it. Use min_chip_rev_full instead. The software interprets this as a Major version for most of the chips and as a Minor version for the ESP32-C3.

uint16_t **min_chip_rev_full**

Minimal chip revision supported by image, in format: major * 100 + minor

uint16_t **max_chip_rev_full**

Maximal chip revision supported by image, in format: major * 100 + minor

uint8_t **reserved**[4]

Reserved bytes in additional header space, currently unused

uint8_t **hash_appended**

If 1, a SHA256 digest "simple hash" (of the entire image) is appended after the checksum. Included in image length. This digest is separate to secure boot and only used for detecting corruption. For secure boot signed images, the signature is appended after this (and the simple hash is included in the signed data).

struct **esp_image_segment_header_t**

Header of binary image segment.

Public Members

uint32_t **load_addr**

Address of segment

uint32_t **data_len**

Length of data

Macros

ESP_IMAGE_HEADER_MAGIC

The magic word for the *esp_image_header_t* structure.

ESP_IMAGE_MAX_SEGMENTS

Max count of segments in the image.

Enumerations

enum **esp_chip_id_t**

ESP chip ID.

Values:

enumerator **ESP_CHIP_ID_ESP32**

chip ID: ESP32

enumerator **ESP_CHIP_ID_ESP32S2**

chip ID: ESP32-S2

enumerator **ESP_CHIP_ID_ESP32C3**

chip ID: ESP32-C3

enumerator **ESP_CHIP_ID_ESP32S3**
chip ID: ESP32-S3

enumerator **ESP_CHIP_ID_ESP32C2**
chip ID: ESP32-C2

enumerator **ESP_CHIP_ID_ESP32C6**
chip ID: ESP32-C6

enumerator **ESP_CHIP_ID_ESP32H2**
chip ID: ESP32-H2

enumerator **ESP_CHIP_ID_ESP32P4**
chip ID: ESP32-P4

enumerator **ESP_CHIP_ID_INVALID**
Invalid chip ID (we defined it to make sure the esp_chip_id_t is 2 bytes size)

enum **esp_image_spi_mode_t**
SPI flash mode, used in *esp_image_header_t*.

Values:

enumerator **ESP_IMAGE_SPI_MODE_QIO**
SPI mode QIO

enumerator **ESP_IMAGE_SPI_MODE_QOUT**
SPI mode QOUT

enumerator **ESP_IMAGE_SPI_MODE_DIO**
SPI mode DIO

enumerator **ESP_IMAGE_SPI_MODE_DOUT**
SPI mode DOUT

enumerator **ESP_IMAGE_SPI_MODE_FAST_READ**
SPI mode FAST_READ

enumerator **ESP_IMAGE_SPI_MODE_SLOW_READ**
SPI mode SLOW_READ

enum **esp_image_spi_freq_t**
SPI flash clock division factor.

Values:

enumerator **ESP_IMAGE_SPI_SPEED_DIV_2**
The SPI flash clock frequency is divided by 2 of the clock source

enumerator **ESP_IMAGE_SPI_SPEED_DIV_3**
The SPI flash clock frequency is divided by 3 of the clock source

enumerator **ESP_IMAGE_SPI_SPEED_DIV_4**

The SPI flash clock frequency is divided by 4 of the clock source

enumerator **ESP_IMAGE_SPI_SPEED_DIV_1**

The SPI flash clock frequency equals to the clock source

enum **esp_image_flash_size_t**

Supported SPI flash sizes.

Values:

enumerator **ESP_IMAGE_FLASH_SIZE_1MB**

SPI flash size 1 MB

enumerator **ESP_IMAGE_FLASH_SIZE_2MB**

SPI flash size 2 MB

enumerator **ESP_IMAGE_FLASH_SIZE_4MB**

SPI flash size 4 MB

enumerator **ESP_IMAGE_FLASH_SIZE_8MB**

SPI flash size 8 MB

enumerator **ESP_IMAGE_FLASH_SIZE_16MB**

SPI flash size 16 MB

enumerator **ESP_IMAGE_FLASH_SIZE_32MB**

SPI flash size 32 MB

enumerator **ESP_IMAGE_FLASH_SIZE_64MB**

SPI flash size 64 MB

enumerator **ESP_IMAGE_FLASH_SIZE_128MB**

SPI flash size 128 MB

enumerator **ESP_IMAGE_FLASH_SIZE_MAX**

SPI flash size MAX

2.9.2 引导加载程序镜像的格式

引导加载程序镜像与应用程序镜像具有相同的结构，参见应用程序镜像结构。二者唯一的区别在于描述引导加载程序的结构体 不同。

要查看关于引导加载程序镜像的更多内容，请运行以下命令：

```
esptool.py --chip esp32p4 image_info build/bootloader/bootloader.bin --version 2
```

输出结果如下形式所示：

```
File size: 26576 (bytes)

ESP32 image header
==================
Image version: 1
Entry point: 0x40080658
Segments: 4
Flash size: 2MB
Flash freq: 40m
Flash mode: DIO

ESP32 extended image header
===========================
WP pin: 0xee
Flash pins drive settings: clk_drv: 0x0, q_drv: 0x0, d_drv: 0x0, cs0_drv: 0x0, hd_
↪drv: 0x0, wp_drv: 0x0
Chip ID: 0
Minimal chip revision: v0.0, (legacy min_rev = 0)
Maximal chip revision: v3.99

Segments information
====================
Segment   Length   Load addr   File offs   Memory types
-------  -------  ----------  ----------  ------------
      1  0x01bb0  0x3fff0030  0x00000018  BYTE_ACCESSIBLE, DRAM, DIRAM_DRAM
      2  0x03c90  0x40078000  0x00001bd0  CACHE_APP
      3  0x00004  0x40080400  0x00005868  IRAM
      4  0x00f2c  0x40080404  0x00005874  IRAM

ESP32 image footer
==================
Checksum: 0x65 (valid)
Validation hash: 6f31a7f8512f26f6bce7c3b270f93bf6cf1ee4602c322998ca8ce27433527e92␣
↪(valid)

Bootloader information
======================
Bootloader version: 1
ESP-IDF: v5.1-dev-4304-gcb51a3b-dirty
Compile time: Mar 30 2023 19:14:17
```

引导加载程序描述

引导加载程序二进制文件的 DRAM0 段起始位置为*esp_bootloader_desc_t* 结构体，其中包含描述引导加载程序的特定字段。此结构体位置具有固定偏移量，大小为 sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*)。

- magic_byte：esp_bootloader_desc 结构体的魔术字节
- reserved：保留供 IDF 未来使用
- version：引导加载程序版本，参见*CONFIG_BOOTLOADER_PROJECT_VER*
- idf_ver：IDF 版本。[1]
- date 和 time：编译日期和时间
- reserved2：保留供 IDF 未来使用

如需从正在运行的引导加载程序中获取*esp_bootloader_desc_t* 结构体，请使用*esp_bootloader_get_description()*。

如需从正在运行的应用程序中获取*esp_bootloader_desc_t* 结构体，请使用*esp_ota_get_bootloader_description()*。

[1] 最大长度为 32 个字符，包括空终止符。

API 参考

Header File

- components/esp_bootloader_format/include/esp_bootloader_desc.h
- This header file can be included with:

```
#include "esp_bootloader_desc.h"
```

- This header file is a part of the API provided by the `esp_bootloader_format` component. To declare that your component depends on `esp_bootloader_format`, add the following to your CMakeLists.txt:

```
REQUIRES esp_bootloader_format
```

or

```
PRIV_REQUIRES esp_bootloader_format
```

Functions

const *esp_bootloader_desc_t* ***esp_bootloader_get_description**(void)

Return esp_bootloader_desc structure.

Intended for use by the bootloader.

返回 Pointer to esp_bootloader_desc structure.

Structures

struct **esp_bootloader_desc_t**

Bootloader description structure.

Public Members

uint8_t **magic_byte**

Magic byte ESP_BOOTLOADER_DESC_MAGIC_BYTE

uint8_t **reserved**[3]

reserved for IDF

uint32_t **version**

Bootloader version

char **idf_ver**[32]

Version IDF

char **date_time**[24]

Compile date and time

uint8_t **reserved2**[16]

reserved for IDF

Macros

ESP_BOOTLOADER_DESC_MAGIC_BYTE

The magic byte for the esp_bootloader_desc structure that is in DRAM.

2.9.3 应用级追踪

概述

ESP-IDF 支持用于程序行为分析的 **应用级追踪**功能。在 menuconfig 中启用该功能后，即可通过 JTAG 接口在主机和 ESP32-P4 之间传输任意数据，最小化程序的执行开销。

在程序运行时，开发人员可使用该库向主机发送应用程序的特定执行状态，并接收来自应用程序的命令或其他信息。该库的主要用例包括：

1. 收集特定应用程序的数据，参见特定应用程序的跟踪。
2. 向主机发送轻量级日志，参见记录日志到主机。
3. 系统行为分析，参见基于 *SEGGER SystemView* 的系统行为分析。

API 参考

Header File

- components/app_trace/include/esp_app_trace.h
- This header file can be included with:

```
#include "esp_app_trace.h"
```

- This header file is a part of the API provided by the `app_trace` component. To declare that your component depends on `app_trace`, add the following to your CMakeLists.txt:

```
REQUIRES app_trace
```

or

```
PRIV_REQUIRES app_trace
```

Functions

esp_err_t **esp_apptrace_init**(void)

Initializes application tracing module.

备注： Should be called before any esp_apptrace_xxx call.

返回 ESP_OK on success, otherwise see esp_err_t

void **esp_apptrace_down_buffer_config**(uint8_t *buf, uint32_t size)

Configures down buffer.

备注： Needs to be called before attempting to receive any data using esp_apptrace_down_buffer_get and esp_apptrace_read. This function does not protect internal data by lock.

参数

- **buf** -- Address of buffer to use for down channel (host to target) data.
- **size** -- Size of the buffer.

uint8_t ***esp_apptrace_buffer_get**(*esp_apptrace_dest_t* dest, uint32_t size, uint32_t tmo)

Allocates buffer for trace data. Once the data in the buffer is ready to be sent, esp_apptrace_buffer_put must be called to indicate it.

参数

- **dest** -- Indicates HW interface to send data.
- **size** -- Size of data to write to trace buffer.
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 non-NULL on success, otherwise NULL.

esp_err_t **esp_apptrace_buffer_put**(*esp_apptrace_dest_t* dest, uint8_t *ptr, uint32_t tmo)

Indicates that the data in the buffer is ready to be sent. This function is a counterpart of and must be preceded by esp_apptrace_buffer_get.

参数

- **dest** -- Indicates HW interface to send data. Should be identical to the same parameter in call to esp_apptrace_buffer_get.
- **ptr** -- Address of trace buffer to release. Should be the value returned by call to esp_apptrace_buffer_get.
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

esp_err_t **esp_apptrace_write**(*esp_apptrace_dest_t* dest, const void *data, uint32_t size, uint32_t tmo)

Writes data to trace buffer.

参数

- **dest** -- Indicates HW interface to send data.
- **data** -- Address of data to write to trace buffer.
- **size** -- Size of data to write to trace buffer.
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

int **esp_apptrace_vprintf_to**(*esp_apptrace_dest_t* dest, uint32_t tmo, const char *fmt, va_list ap)

vprintf-like function to send log messages to host via specified HW interface.

参数

- **dest** -- Indicates HW interface to send data.
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.
- **fmt** -- Address of format string.
- **ap** -- List of arguments.

返回 Number of bytes written.

int **esp_apptrace_vprintf**(const char *fmt, va_list ap)

vprintf-like function to send log messages to host.

参数

- **fmt** -- Address of format string.
- **ap** -- List of arguments.

返回 Number of bytes written.

esp_err_t **esp_apptrace_flush**(*esp_apptrace_dest_t* dest, uint32_t tmo)

Flushes remaining data in trace buffer to host.

参数

- **dest** -- Indicates HW interface to flush data on.
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

esp_err_t **esp_apptrace_flush_nolock**(*esp_apptrace_dest_t* dest, uint32_t min_sz, uint32_t tmo)

Flushes remaining data in trace buffer to host without locking internal data. This is a special version of esp_apptrace_flush which should be called from panic handler.

参数

- **dest** -- Indicates HW interface to flush data on.
- **min_sz** -- Threshold for flushing data. If current filling level is above this value, data will be flushed. TRAX destinations only.
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

esp_err_t **esp_apptrace_read**(*esp_apptrace_dest_t* dest, void *data, uint32_t *size, uint32_t tmo)

Reads host data from trace buffer.

参数

- **dest** -- Indicates HW interface to read the data on.
- **data** -- Address of buffer to put data from trace buffer.
- **size** -- Pointer to store size of read data. Before call to this function pointed memory must hold requested size of data
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

uint8_t ***esp_apptrace_down_buffer_get**(*esp_apptrace_dest_t* dest, uint32_t *size, uint32_t tmo)

Retrieves incoming data buffer if any. Once data in the buffer is processed, esp_apptrace_down_buffer_put must be called to indicate it.

参数

- **dest** -- Indicates HW interface to receive data.
- **size** -- Address to store size of available data in down buffer. Must be initialized with requested value.
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 non-NULL on success, otherwise NULL.

esp_err_t **esp_apptrace_down_buffer_put**(*esp_apptrace_dest_t* dest, uint8_t *ptr, uint32_t tmo)

Indicates that the data in the down buffer is processed. This function is a counterpart of and must be preceded by esp_apptrace_down_buffer_get.

参数

- **dest** -- Indicates HW interface to receive data. Should be identical to the same parameter in call to esp_apptrace_down_buffer_get.
- **ptr** -- Address of trace buffer to release. Should be the value returned by call to esp_apptrace_down_buffer_get.
- **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

bool **esp_apptrace_host_is_connected**(*esp_apptrace_dest_t* dest)

Checks whether host is connected.

参数 **dest** -- Indicates HW interface to use.

返回 true if host is connected, otherwise false

void ***esp_apptrace_fopen**(*esp_apptrace_dest_t* dest, const char *path, const char *mode)

Opens file on host. This function has the same semantic as 'fopen' except for the first argument.

参数

- **dest** -- Indicates HW interface to use.
- **path** -- Path to file.
- **mode** -- Mode string. See fopen for details.

返回 non zero file handle on success, otherwise 0

int **esp_apptrace_fclose**(*esp_apptrace_dest_t* dest, void *stream)

Closes file on host. This function has the same semantic as 'fclose' except for the first argument.

参数

- **dest** -- Indicates HW interface to use.
- **stream** -- File handle returned by esp_apptrace_fopen.

返回 Zero on success, otherwise non-zero. See fclose for details.

size_t **esp_apptrace_fwrite**(*esp_apptrace_dest_t* dest, const void *ptr, size_t size, size_t nmemb, void *stream)

Writes to file on host. This function has the same semantic as 'fwrite' except for the first argument.

参数

- **dest** -- Indicates HW interface to use.
- **ptr** -- Address of data to write.
- **size** -- Size of an item.
- **nmemb** -- Number of items to write.
- **stream** -- File handle returned by esp_apptrace_fopen.

返回 Number of written items. See fwrite for details.

size_t **esp_apptrace_fread**(*esp_apptrace_dest_t* dest, void *ptr, size_t size, size_t nmemb, void *stream)

Read file on host. This function has the same semantic as 'fread' except for the first argument.

参数

- **dest** -- Indicates HW interface to use.
- **ptr** -- Address to store read data.
- **size** -- Size of an item.
- **nmemb** -- Number of items to read.
- **stream** -- File handle returned by esp_apptrace_fopen.

返回 Number of read items. See fread for details.

int **esp_apptrace_fseek**(*esp_apptrace_dest_t* dest, void *stream, long offset, int whence)

Set position indicator in file on host. This function has the same semantic as 'fseek' except for the first argument.

参数

- **dest** -- Indicates HW interface to use.
- **stream** -- File handle returned by esp_apptrace_fopen.
- **offset** -- Offset. See fseek for details.
- **whence** -- Position in file. See fseek for details.

返回 Zero on success, otherwise non-zero. See fseek for details.

int **esp_apptrace_ftell**(*esp_apptrace_dest_t* dest, void *stream)

Get current position indicator for file on host. This function has the same semantic as 'ftell' except for the first argument.

参数

- **dest** -- Indicates HW interface to use.
- **stream** -- File handle returned by esp_apptrace_fopen.

返回 Current position in file. See ftell for details.

int **esp_apptrace_fstop**(*esp_apptrace_dest_t* dest)

Indicates to the host that all file operations are complete. This function should be called after all file operations are finished and indicate to the host that it can perform cleanup operations (close open files etc.).

参数 **dest** -- Indicates HW interface to use.

返回 ESP_OK on success, otherwise see esp_err_t

int **esp_apptrace_feof**(*esp_apptrace_dest_t* dest, void *stream)

Test end-of-file indicator on a stream. This function has the same semantic as 'feof' except for the first argument.

参数

- **dest** -- Indicates HW interface to use.
- **stream** -- File handle returned by esp_apptrace_fopen.

返回 Non-Zero if end-of-file indicator is set for stream. See feof for details.

void **esp_gcov_dump**(void)

Triggers gcov info dump. This function waits for the host to connect to target before dumping data.

Enumerations

enum **esp_apptrace_dest_t**

Application trace data destinations bits.

Values:

enumerator **ESP_APPTRACE_DEST_JTAG**

JTAG destination.

enumerator **ESP_APPTRACE_DEST_TRAX**

xxx_TRAX name is obsolete, use more common xxx_JTAG

enumerator **ESP_APPTRACE_DEST_UART**

UART destination.

enumerator **ESP_APPTRACE_DEST_MAX**

enumerator **ESP_APPTRACE_DEST_NUM**

Header File

- components/app_trace/include/esp_sysview_trace.h
- This header file can be included with:

```
#include "esp_sysview_trace.h"
```

- This header file is a part of the API provided by the `app_trace` component. To declare that your component depends on `app_trace`, add the following to your CMakeLists.txt:

```
REQUIRES app_trace
```

or

```
PRIV_REQUIRES app_trace
```

Functions

static inline *esp_err_t* **esp_sysview_flush**(uint32_t tmo)

Flushes remaining data in SystemView trace buffer to host.

参数 **tmo** -- Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinetly.

返回 ESP_OK.

int **esp_sysview_vprintf**(const char *format, va_list args)

vprintf-like function to sent log messages to the host.

参数

- **format** -- Address of format string.
- **args** -- List of arguments.

返回 Number of bytes written.

esp_err_t **esp_sysview_heap_trace_start**(uint32_t tmo)

Starts SystemView heap tracing.

参数 **tmo** -- Timeout (in us) to wait for the host to be connected. Use -1 to wait forever.
返回 ESP_OK on success, ESP_ERR_TIMEOUT if operation has been timed out.

esp_err_t **esp_sysview_heap_trace_stop**(void)

Stops SystemView heap tracing.

返回 ESP_OK.

void **esp_sysview_heap_trace_alloc**(void *addr, uint32_t size, const void *callers)

Sends heap allocation event to the host.

参数

- **addr** -- Address of allocated block.
- **size** -- Size of allocated block.
- **callers** -- Pointer to array with callstack addresses. Array size must be CONFIG_HEAP_TRACING_STACK_DEPTH.

void **esp_sysview_heap_trace_free**(void *addr, const void *callers)

Sends heap de-allocation event to the host.

参数

- **addr** -- Address of de-allocated block.
- **callers** -- Pointer to array with callstack addresses. Array size must be CONFIG_HEAP_TRACING_STACK_DEPTH.

2.9.4 使用外部堆栈调用函数

概述

执行某个给定函数时，可以使用用户分配的堆栈空间，且该堆栈空间独立于当前任务的堆栈。这一机制能够节省在调用常用函数时浪费大量堆栈空间，例如 `printf` 函数。具体而言，将给定函数作为参数传入*esp_execute_shared_stack_function()* 中，给定函数便会在共享堆栈空间内作为回调函数延迟执行。

警告： *esp_execute_shared_stack_function()* 只会为所提供的共享堆栈内存做最基础的设置。传递给该函数以在共享堆栈空间上执行的函数，或调用该函数的任何函数，都不应执行以下任何操作：

- 使用线程局部存储 (TLS)
- 使用浮点运算器 (FPU)
- 使用 AI 协处理器
- 调用 vTaskDelete(NULL) 删除当前运行的任务

此外，从共享堆栈上运行的函数或调用该函数的任何函数来调用回溯，回溯信息都可能不正确。这方面的限制十分严格，因此将来*esp_execute_shared_stack_function()* 可能会被弃用。如有用例必须使用*esp_execute_shared_stack_function()* 函数才能实现，请提交 GitHub Issue。

使用方法

esp_execute_shared_stack_function() 需要四个参数：

- 由调用者分配的互斥锁，防止同一函数共享分配的堆栈

- 指向分配的堆栈顶部的指针
- 以字节为单位的堆栈的大小
- 指向需要共享堆栈的函数的指针

通过这一功能，用户指定的函数被作为回调函数延迟执行，并在用户分配的空间中调用，无需从当前任务堆栈中获取空间。

该函数的使用方式如下所示：

```
void external_stack_function(void)
{
    printf("Executing this printf from external stack! \n");
}

//假设要使用一个单独的堆栈空间来调用 printf 函数
//允许应用程序减小其堆栈大小
void app_main()
{
    //从堆中或以静态方式分配一个堆栈 buffer:
    StackType_t *shared_stack = malloc(8192 * sizeof(StackType_t));
    assert(shared_stack != NULL);

    //分配一个互斥锁:
    SemaphoreHandle_t printf_lock = xSemaphoreCreateMutex();
    assert(printf_lock != NULL);

    //使用宏助手调用所需函数:
    esp_execute_shared_stack_function(printf_lock,
                                      shared_stack,
                                      8192,
                                      external_stack_function);

    vSemaphoreDelete(printf_lock);
    free(shared_stack);
}
```

API 参考

Header File

- components/esp_system/include/esp_expression_with_stack.h
- This header file can be included with:

```
#include "esp_expression_with_stack.h"
```

Functions

void **esp_execute_shared_stack_function**(*SemaphoreHandle_t* lock, void *stack, size_t stack_size, *shared_stack_function* function)

Calls function on user defined shared stack space.

After returning, the original stack is used again.

备注： if either lock, stack or stack size is invalid, the expression will be called using the current stack.

警告： This function does minimal preparation of the provided piece of memory (`stack`). DO NOT do any of the following in `function` or any of its callees:

- Use Thread-local storage

- Use the Floating-point unit on ESP32-P4
- Use the AI co-processor on ESP32-P4
- Call vTaskDelete(NULL) (deleting the currently running task) Furthermore, backtraces will be wrong when called from `function` or any of its callees. The limitations are quite sever, so that we might deprecate this function in the future. If you have any use case which can only be implemented using this function, please open an issue on github.

参数

- **`lock`** -- Mutex object to protect in case of shared stack
- **`stack`** -- Pointer to user allocated stack
- **`stack_size`** -- Size of current stack in bytes
- **`function`** -- pointer to the shared stack function to be executed

Macros

`ESP_EXECUTE_EXPRESSION_WITH_STACK`(lock, stack, stack_size, expression)

Type Definitions

typedef void (*__`shared_stack_function`__)(void)

2.9.5 芯片版本

概述

ESP32-P4 可能包含多个芯片版本。引入不同的版本主要用于修复问题，有时也用于添加新功能。*版本控制方案* 描述了这些芯片修订的版本以及运行时读取版本信息所用的 API。

须考虑应用程序、ESP-IDF 版本和芯片版本之间的兼容性：

- 应用程序可能依赖于某个芯片版本的修复内容或功能。
- 较新版本的硬件可能与较早版本的 ESP-IDF 不兼容。

本文档的*ESP-IDF 兼容性检查* 部分描述了应用程序如何明确芯片版本需求，以及 ESP-IDF 如何对兼容性进行检查。文档最后提供了此方案的故障排除相关信息。

版本控制方案

芯片版本号通常以 `vX.Y` 的形式表示，其中：

- `X` 表示主版本号。该号码变更表示在旧版芯片上使用的软件与新版芯片不兼容，必须升级软件方可使用。
- `Y` 表示次版本号。该号码变更表示在旧版芯片上使用的软件与新版芯片兼容，无需升级软件。

如果新版芯片不包含重大变更，则可以兼容旧版芯片上运行的软件。此时，新版芯片的主版本号不变，次版本号加 1。例如，版本号从 `v1.1` 变为 `v1.2`。

相反，如果新版芯片包含重大变更，则 **无法兼容**旧版芯片的软件。此时，新版芯片的主版本号增加，次版本号设置为 0。例如，版本号从 `v1.1` 变为 `v2.0`。

此版本控制方式能够表示芯片版本之间的衍生关系，并清晰表明芯片变更是否为重大变更。

在芯片发生非重大变更时，ESP-IDF 的执行逻辑应与此前最近版本的逻辑相同，从而达到无缝衔接。在这样便可直接将编译后的二进制文件迁移到更新了次版本的芯片上，无需升级 ESP-IDF 版本并重新编译整个项目。

当二进制文件在更新了主版本的芯片上意外执行时，软件也能够根据主版本报告问题。此主次版本方案还允许硬件变更分支化。

备注： 目前的主次版本号芯片版本方案是从 ESP-IDF v5.0 开始引入的。因此，早期 ESP-IDF 创建的引导加载程序将继续使用晶圆版本表示的芯片版本方案。

用于芯片版本的 eFuse 位 芯片使用以下几个 eFuse 字段来进行版本控制：

- 主芯片版本号 (`WAFER_VERSION_MAJOR` eFuse)
- 次芯片版本号 (`WAFER_VERSION_MINOR` eFuse)
- 忽略最大芯片版本限制位 (`DISABLE_WAFER_VERSION_MAJOR` eFuse)。请参考*ESP-IDF 兼容性检查* 了解此 eFuse 字段。

备注： 此前的版本控制逻辑基于单一的 eFuse 版本字段，即 `WAFER_VERSION`。这种方式无法表明芯片的更新是否为重大更新，是一种线性的版本控制逻辑。

用于 eFuse 块版本的 eFuse 位 EFuse 既有以下几个版本字段：

- 主 eFuse 块版本号 (`BLK_VERSION_MAJOR` eFuse)
- 次 efuse 块版本号 (`BLK_VERSION_MINOR` eFuse)
- 忽略最大 efuse 块版本限制位 (`DISABLE_BLK_VERSION_MAJOR` eFuse). 请参考*ESP-IDF 兼容性检查* 了解此 eFuse 字段。

芯片版本 API 使用下列 API 从 eFuse 中读取芯片版本：

- *`efuse_hal_chip_revision()`*，返回的版本格式为 `major * 100 + minor`。
- *`efuse_hal_get_major_chip_version()`* 返回主芯片版本号。
- *`efuse_hal_get_minor_chip_version()`* 返回次芯片版本号。

下列 Kconfig 选项（格式为 `major * 100 + minor`）可以将芯片版本依赖添加到代码中：

- `CONFIG_ESP32P4_REV_MIN_FULL`
- `CONFIG_ESP_REV_MIN_FULL`
- `CONFIG_ESP32P4_REV_MAX_FULL`
- `CONFIG_ESP_REV_MAX_FULL`

EFuse 块版本 API 使用下列 API 从 eFuse 中读取 eFuse 块版本：

- *`efuse_hal_blk_version()`*，返回的版本格式为 `major * 100 + minor`。
- `efuse_ll_get_blk_version_major()`。返回主 eFuse 块版本号
- `efuse_ll_get_blk_version_minor()`。返回次 eFuse 块版本号

下列 Kconfig 选项（格式为 `major * 100 + minor`）可以将 eFuse 块版本依赖添加到代码中：

- `CONFIG_ESP_EFUSE_BLOCK_REV_MIN_FULL`
- `CONFIG_ESP_EFUSE_BLOCK_REV_MAX_FULL`

ESP-IDF 兼容性检查

如果构建的应用程序需要支持特定芯片的多个版本，可通过 Kconfig 指定支持的最小和最大芯片版本号。

最小芯片版本号可以通过 Kconfig 选项*CONFIG_ESP32P4_REV_MIN* 来选择。设置最小芯片版本后，软件只能在较新的芯片版本上运行，以便支持某些功能或修复某些错误。

最大芯片版本号无法指定，只能由当前使用的 ESP-IDF 版本自动决定。ESP-IDF 会拒绝启动任何超过最大芯片版本号的芯片版本。由于特定版本的 ESP-IDF 无法预知未来的芯片版本更新，因此最大芯片版本

号通常设置为 maximum supported MAJOR version + 99。可以设置“忽略最大版本” eFuse 来绕过最大版本限制，但这不能确保软件正常工作。

EFuse 块版本号与芯片版本号类似，但是它主要影响在 eFuse 中指定的参数（如 ADC 校正参数等）。

下文介绍了芯片版本未通过兼容性检查时显示的故障排除信息及解决方法，并描述了在早期 ESP-IDF 版本中与软件行为和兼容性检查相关的技术细节。

故障排除

1. 如果第二阶段引导加载程序所运行的芯片版本低于镜像（如软件镜像）中指定的最小版本，会发生重启并显示以下消息：

```
Image requires chip rev >= v3.0, but chip is v1.0
```

要解决此问题，

- 确保使用的芯片达到了要求的最低版本及以上。
- 减小*CONFIG_ESP32P4_REV_MIN* 的值并重建镜像，使镜像的版本与当前芯片版本兼容。

1. 如果应用程序所需的芯片版本不处于最小和最大芯片版本的区间范围内，会发生重启并显示以下消息：

```
Image requires chip rev <= v2.99, but chip is v3.0
```

为解决此问题，需更新 ESP-IDF 到较新版本，以支持该芯片版本 (CONFIG_ESP32P4_REV_MAX_FULL)。也可以在 eFuse 中设置 Ignore maximal revision 位，或使用与当前 ESP-IDF 版本兼容的其他芯片版本。

二进制镜像的常见版本需求 芯片版本号检查主要根据二级引导程序和应用程序二进制镜像中包含的*esp_image_header_t* 标头，其中记录了可以运行该软件的芯片版本号。这一标头有 3 个与版本相关的字段：

- min_chip_rev - 镜像所需芯片的最小主版本号（但对于 ESP32-C3，该字段指次版本号）。其值由*CONFIG_ESP32P4_REV_MIN* 确定。
- min_chip_rev_full - 镜像所需芯片的最小版本号，格式为 major * 100 + minor。其值由*CONFIG_ESP32P4_REV_MIN* 确定。
- max_chip_rev_full - 镜像所需芯片的最大版本号，格式为 major * 100 + minor。其值由 CONFIG_ESP32P4_REV_MAX_FULL 确定。用户无法对其进行修改，仅当 ESP-IDF 支持新版本时由乐鑫官方进行更改。

而 eFuse 块版本的要求则存储在*esp_app_desc_t* 结构体中。该结构体对象位于应用程序的二进制进项文件中。由于 eFuse 块版本信息主要影响 ADC 校准，而二级引导程序的镜像不涉及 ADC，因此我们只需要检查应用程序镜像的 eFuse 块版本信息。有 2 个与 eFuse 块版本相关的字段：

- min_efuse_blk_rev_full - 镜像所需 eFuse 块的最小版本号，格式为 major * 100 + minor。其值由 CONFIG_ESP_EFUSE_BLOCK_REV_MIN_FULL 确定。
- max_efuse_blk_rev_full - 镜像所需 eFuse 块的最大版本号，格式为 major * 100 + minor。其值由 CONFIG_ESP_EFUSE_BLOCK_REV_MAX_FULL 确定。它反映了当前 ESP-IDF 版本能支持的最大 eFuse 块版本号，不应由用户修改。

最大和最小版本限制 应用启动过程中，检查最小和最大版本的顺序如下：

1. 在运行第 2 阶段引导启动程序前，第 1 阶段引导启动程序（ROM 引导启动程序）不会在*esp_image_header_t* 中检查最小和最大版本字段。
2. 在第 2 阶段引导启动程序的初始化阶段，会检查引导程序自身是否可以在此版本的芯片上启动。它从引导启动程序镜像的标头中读取最小版本，并与 eFuse 中的芯片版本进行比较。如果芯片版本低于最小版本，引导启动程序会拒绝启动并中止运行。此阶段不检查最大版本。
3. 然后，第 2 阶段引导启动程序会检查应用程序的版本要求。它从应用程序镜像的标头中读取支持的芯片最小和最大版本，以及从段的标头中读取 eFuse 块版本信息，并与 eFuse 中的芯片版本进行比较。如果该芯片版本或 eFuse 块版本低于各自的最小版本或高于最大版本，引导程序会拒绝启动并

中止。然而，如果设置了忽略最大版本位，则可以忽略最大版本限制。软件确定可以使用此芯片版本时，用户可以自行设置忽略位。

4. 在空中升级 (OTA) 阶段，运行中的应用程序会检查新软件是否与芯片版本及 eFuse 块版本相匹配。它会从新应用程序镜像的标头中提取最小和最大芯片版本，以及应用程序描述中提取最大和最小 eFuse 块版本，并与 eFuse 中的芯片版本和块版本进行比较。应用程序检查版本匹配的方式与引导启动程序相同，即芯片版本和 eFuse 块版本须处在最小和最大版本之间（忽略最大版本的逻辑也相同）。

向后兼容旧版 ESP-IDF 构建的引导启动程序 请使用 `esptool chip_id` 命令查看芯片版本。

参考链接

- 芯片版本编号方案兼容指南
- ESP-IDF 版本与乐鑫芯片版本兼容性
- SoC Errata
- *ESP-IDF 版本简介*

API 参考

Header File

- components/hal/include/hal/efuse_hal.h
- This header file can be included with:

```
#include "hal/efuse_hal.h"
```

Functions

void **efuse_hal_get_mac**(uint8_t *mac)

get factory mac address

uint32_t **efuse_hal_chip_revision**(void)

Returns chip version.

返回 Chip version in format: Major * 100 + Minor

uint32_t **efuse_hal_blk_version**(void)

Return block version.

返回 Block version in format: Major * 100 + Minor

bool **efuse_hal_flash_encryption_enabled**(void)

Is flash encryption currently enabled in hardware?

Flash encryption is enabled if the FLASH_CRYPT_CNT efuse has an odd number of bits set.

返回 true if flash encryption is enabled.

bool **efuse_hal_get_disable_wafer_version_major**(void)

Returns the status of whether the bootloader (and OTA) will check the maximum chip version or not.

返回 true - Skip the maximum chip version check.

bool **efuse_hal_get_disable_blk_version_major**(void)

Returns the status of whether the app start-up (and OTA) will check the efuse block version or not.

返回 true - Skip the efuse block version check.

uint32_t **efuse_hal_get_major_chip_version**(void)

Returns major chip version.

uint32_t **efuse_hal_get_minor_chip_version**(void)

Returns minor chip version.

void **efuse_hal_set_ecdsa_key**(int efuse_key_blk)

Set the efuse block that should be used as ECDSA private key.

备注： The efuse block must be burnt with key purpose ECDSA_KEY

参数 **efuse_key_blk** -- Efuse key block number (Must be in [EFUSE_BLK_KEY0...EFUSE_BLK_KEY_MAX - 1] range)

2.9.6 控制台终端

ESP-IDF 提供了 `console` 组件，它包含了开发基于串口的交互式控制终端所需要的所有模块，主要支持以下功能：

- 行编辑，由 linenoise 库具体实现，它支持处理退格键和方向键，支持回看命令的历史记录，支持命令的自动补全和参数提示。
- 将命令行拆分为参数列表。
- 参数解析，由 argtable3 库具体实现，该库提供解析 GNU 样式的命令行参数的 API。
- 用于注册和调度命令的函数。
- 帮助创建 REPL (Read-Evaluate-Print-Loop) 环境的函数。

备注： 这些功能模块可以一起使用，也可以独立使用，例如仅使用行编辑和命令注册的功能，然后使用 `getopt` 函数或者自定义的函数来实现参数解析，而不是直接使用 argtable3 库。同样地，还可以使用更简单的命令输入方法（比如 `fgets` 函数）和其他用于命令分割和参数解析的方法。

备注： 当在支持硬件 USB 串行接口的芯片上使用控制台应用程序时，建议禁用次级串口控制台输出。次级输出仅用于显示，对于交互式应用程序没有任何作用。

行编辑

行编辑功能允许用户通过按键输入来编辑命令，使用退格键删除符号，使用左/右键在命令中移动光标，使用上/下键导航到之前输入的命令，使用制表键（“Tab”）来自动补全命令。

备注： 此功能依赖于终端应用程序对 ANSI 转义符的支持。因此，显示原始 UART 数据的串口监视器不能与行编辑库一同使用。如果运行 system/console 示例程序的时候看到的输出结果是 `[6n` 或者类似的转义字符而不是命令行提示符 `esp>` 时，就表明当前的串口监视器不支持 ANSI 转义字符。已知可用的串口监视程序有 GNU screen、minicom 和 esp-idf-monitor（可以通过在项目目录下执行 `idf.py monitor` 来调用）。

前往这里可以查看 linenoise 库提供的所有函数的描述。

配置 Linenoise 库不需要显式地初始化，但是在调用行编辑函数之前，可能需要对某些配置的默认值稍作修改。

- `linenoiseClearScreen()`
  使用转义字符清除终端屏幕，并将光标定位在左上角。

- `linenoiseSetMultiLine()`
  在单行和多行编辑模式之间进行切换。单行模式下，如果命令的长度超过终端的宽度，会在行内滚动命令文本以显示文本的结尾，在这种情况下，文本的开头部分会被隐藏。单行模式在每次按下按键时发送给屏幕刷新的数据比较少，与多行模式相比更不容易发生故障。另一方面，在单行模式下编辑命令和复制命令将变得更加困难。默认情况下开启的是单行模式。
- `linenoiseAllowEmpty()`
  设置 linenoise 库收到空行的解析行为，设置为 `true` 时返回长度为零的字符串 (`""`)，设置为 `false` 时返回 `NULL`。默认情况下，将返回长度为零的字符串。
- `linenoiseSetMaxLineLen()`
  设置 linenoise 库中每行的最大长度，默认长度为 4096 字节，可以通过更新该默认值来优化 RAM 内存的使用。

主循环

- `linenoise()`
  在大多数情况下，控制台应用程序都会具有相同的工作形式——在某个循环中不断读取输入的内容，然后解析再处理。`linenoise()` 是专门用来获取用户按键输入的函数，当回车键被按下后会便返回完整的一行内容。因此可以用它来完成前面循环中的“读取”任务。
- `linenoiseFree()`
  必须调用此函数才能释放从 `linenoise()` 函数获取的命令行缓冲区。

提示和补全

- `linenoiseSetCompletionCallback()`
  当用户按下制表键时，linenoise 会调用 **补全回调函数**，该回调函数会检查当前已经输入的内容，然后调用 `linenoiseAddCompletion()` 函数来提供所有可能的补全后的命令列表。启用补全功能，需要事先调用 `linenoiseSetCompletionCallback()` 函数来注册补全回调函数。
  `console` 组件提供了一个现成的函数来为注册的命令提供补全功能*esp_console_get_completion()*（见下文）。
- `linenoiseAddCompletion()`
  补全回调函数会通过调用此函数来通知 linenoise 库当前键入命令所有可能的补全结果。
- `linenoiseSetHintsCallback()`
  每当用户的输入改变时，linenoise 就会调用此回调函数，检查到目前为止输入的命令行内容，然后提供带有提示信息的字符串（例如命令参数列表），然后会在同一行上用不同的颜色显示出该文本。
- `linenoiseSetFreeHintsCallback()`
  如果 **提示回调函数** 返回的提示字符串是动态分配的或者需要以其它方式回收，就需要使用 `linenoiseSetFreeHintsCallback()` 注册具体的清理函数。

历史记录

- `linenoiseHistorySetMaxLen()`
  该函数设置要保留在内存中的最近输入的命令的数量。用户通过使用向上/向下箭头来导航历史记录。
- `linenoiseHistoryAdd()`
  Linenoise 不会自动向历史记录中添加命令，应用程序需要调用此函数来将命令字符串添加到历史记录中。
- `linenoiseHistorySave()`
  该函数将命令的历史记录从 RAM 中保存为文本文件，例如保存到 SD 卡或者 flash 的文件系统中。
- `linenoiseHistoryLoad()`
  与 `linenoiseHistorySave` 相对应，从文件中加载历史记录。
- `linenoiseHistoryFree()`
  释放用于存储命令历史记录的内存。当使用完 linenoise 库后需要调用此函数。

将命令行拆分成参数列表

`console` 组件提供*esp_console_split_argv()* 函数来将命令行字符串拆分为参数列表。该函数会返回参数的数量 (`argc`) 和一个指针数组，该指针数组可以作为 `argv` 参数传递给任何接受 `argc`, `argv`

格式参数的函数。

根据以下规则来将命令行拆分成参数列表：

- 参数由空格分隔
- 如果参数本身需要使用空格，可以使用 \ （反斜杠）对它们进行转义
- 其它能被识别的转义字符有 \\ （显示反斜杠本身）和 \" （显示双引号）
- 可以使用双引号来引用参数，引号只可能出现在参数的开头和结尾。参数中的引号必须如上所述进行转义。参数周围的引号会被*esp_console_split_argv()* 函数删除

示例：

- `abc def 1 20 .3` > `[abc, def, 1, 20, .3]`
- `abc "123 456" def` > `[abc, 123 456, def]`
- `` `a\ b\\c\" `` > `[a b\c"]`

参数解析

对于参数解析，`console` 组件使用 argtable3 库。有关 argtable3 的介绍请查看 教程 或者 Github 仓库中的 示例代码 。

命令的注册与调度

`console` 组件包含了一些工具函数，用来注册命令，将用户输入的命令和已经注册的命令进行匹配，使用命令行输入的参数调用命令。

应用程序首先调用*esp_console_init()* 来初始化命令注册模块，然后调用*esp_console_cmd_register()* 函数注册命令处理程序。

对于每个命令，应用程序需要提供以下信息（需要以 `esp_console_cmd_t` 结构体的形式给出）：

- 命令名字（不含空格的字符串）
- 帮助文档，解释该命令的用途
- 可选的提示文本，列出命令的参数。如果应用程序使用 `Argtable3` 库来解析参数，则可以通过提供指向 argtable 参数定义结构体的指针来自动生成提示文本
- 命令处理函数（无上下文），或
- 命令处理函数（有上下文）。如要使用此函数，则必须在调用其他命令 **之前** 调用 `esp_console_cmd_set_context()` 初始化上下文。

备注： 使用接受上下文的命令处理函数或者不接受上下文的命令处理函数均可，但两者不能同时使用。如果使用接受上下文的命令处理程序函数，则必须调用 `esp_console_cmd_set_context()` 初始化上下文，否则该函数可能会访问未初始化的上下文。

命令注册模块还提供了其它函数：

- *esp_console_run()*
  该函数接受命令行字符串，使用*esp_console_split_argv()* 函数将其拆分为 argc/argv 形式的参数列表，在已经注册的组件列表中查找命令，如果找到，则执行其对应的处理程序。
- *esp_console_register_help_command()*
  将 `help` 命令添加到已注册命令列表中，此命令将会以列表的方式打印所有注册的命令及其参数和帮助文本。
- *esp_console_get_completion()*
  与 linenoise 库中的 `linenoiseSetCompletionCallback()` 一同使用的回调函数，根据已经注册的命令列表为 linenoise 提供补全功能。
- *esp_console_get_hint()*
  与 linenoise 库中 `linenoiseSetHintsCallback()` 一同使用的回调函数，为 linenoise 提供已经注册的命令的参数提示功能。

初始化 REPL 环境

除了上述的各种函数，`console` 组件还提供了一些 API 来帮助创建一个基本的 REPL 环境。

在一个典型的 console 应用中，你只需要调用*`esp_console_new_repl_uart()`*，它会为你初始化好构建在 UART 基础上的 REPL 环境，其中包括安装 UART 驱动，基本的 console 配置，创建一个新的线程来执行 REPL 任务，注册一些基本的命令（比如 *help* 命令）。

之后你可以使用*`esp_console_cmd_register()`* 来注册其它命令。REPL 环境在初始化后需要再调用*`esp_console_start_repl()`* 函数才能开始运行。

同样，如果 REPL 环境是构建在 USB_SERIAL_JTAG 设备基础上，你只需要先调用*`esp_console_new_repl_usb_serial_jtag()`* 函数进行初始化，然后再照常调用其它函数。

应用程序示例

system/console 目录下提供了 `console` 组件的示例应用程序，展示了具体的使用方法。该示例介绍了如何初始化 UART 和 VFS 的功能，设置 linenoise 库，从 UART 中读取命令并加以处理，然后将历史命令存储到 flash 中。更多信息，请参阅示例代码目录中的 README.md 文件。

此外，ESP-IDF 还提供了众多基于 `console` 组件的示例程序，它们可以辅助应用程序的开发。例如，peripherals/i2c/i2c_tools，wifi/iperf 等等。

API 参考

Header File

- components/console/esp_console.h
- This header file can be included with:

```
#include "esp_console.h"
```

- This header file is a part of the API provided by the `console` component. To declare that your component depends on `console`, add the following to your CMakeLists.txt:

```
REQUIRES console
```

or

```
PRIV_REQUIRES console
```

Functions

esp_err_t **esp_console_init** (const *esp_console_config_t* *config)

initialize console module

备注： Call this once before using other console module features

参数 **config** -- console configuration

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_INVALID_STATE if already initialized
- ESP_ERR_INVALID_ARG if the configuration is invalid

esp_err_t **esp_console_deinit**(void)

de-initialize console module

备注: Call this once when done using console module functions

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if not initialized yet

esp_err_t **esp_console_cmd_register**(const *esp_console_cmd_t* *cmd)

Register console command.

备注: If the member `func_w_context` of cmd is set instead of func, then the member `context` is passed to the function pointed to by `func_w_context`.

参数 **cmd** -- pointer to the command description; can point to a temporary value

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_INVALID_ARG if command description includes invalid arguments
- ESP_ERR_INVALID_ARG if both func and func_w_context members of cmd are non-NULL
- ESP_ERR_INVALID_ARG if both func and func_w_context members of cmd are NULL

esp_err_t **esp_console_run**(const char *cmdline, int *cmd_ret)

Run command line.

参数

- **cmdline** -- command line (command name followed by a number of arguments)
- **cmd_ret** -- **[out]** return code from the command (set if command was run)

返回

- ESP_OK, if command was run
- ESP_ERR_INVALID_ARG, if the command line is empty, or only contained whitespace
- ESP_ERR_NOT_FOUND, if command with given name wasn't registered
- ESP_ERR_INVALID_STATE, if esp_console_init wasn't called

size_t **esp_console_split_argv**(char *line, char **argv, size_t argv_size)

Split command line into arguments in place.

```
* - This function finds whitespace-separated arguments in the given input line.
*
*     'abc def 1 20 .3' -> [ 'abc', 'def', '1', '20', '.3' ]
*
* - Argument which include spaces may be surrounded with quotes. In this case
*   spaces are preserved and quotes are stripped.
*
*     'abc "123 456" def' -> [ 'abc', '123 456', 'def' ]
*
* - Escape sequences may be used to produce backslash, double quote, and space:
*
*     'a\ b\\c\"' -> [ 'a b\c"' ]
*
```

备注: Pointers to at most argv_size - 1 arguments are returned in argv array. The pointer after the last one (i.e. argv[argc]) is set to NULL.

参数

- **line** -- pointer to buffer to parse; it is modified in place
- **argv** -- array where the pointers to arguments are written
- **argv_size** -- number of elements in argv_array (max. number of arguments)

返回 number of arguments found (argc)

void **esp_console_get_completion**(const char *buf, *linenoiseCompletions* *lc)

Callback which provides command completion for linenoise library.

When using linenoise for line editing, command completion support can be enabled like this:

linenoiseSetCompletionCallback(&esp_console_get_completion);

参数

- **buf** -- the string typed by the user
- **lc** -- linenoiseCompletions to be filled in

const char ***esp_console_get_hint**(const char *buf, int *color, int *bold)

Callback which provides command hints for linenoise library.

When using linenoise for line editing, hints support can be enabled as follows:

linenoiseSetHintsCallback((linenoiseHintsCallback*) &esp_console_get_hint);

The extra cast is needed because linenoiseHintsCallback is defined as returning a char* instead of const char*.

参数

- **buf** -- line typed by the user
- **color** -- **[out]** ANSI color code to be used when displaying the hint
- **bold** -- **[out]** set to 1 if hint has to be displayed in bold

返回 string containing the hint text. This string is persistent and should not be freed (i.e. linenoiseSetFreeHintsCallback should not be used).

esp_err_t **esp_console_register_help_command**(void)

Register a 'help' command.

Default 'help' command prints the list of registered commands along with hints and help strings if no additional argument is given. If an additional argument is given, the help command will look for a command with the same name and only print the hints and help strings of that command.

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE, if esp_console_init wasn't called

esp_err_t **esp_console_new_repl_uart**(const *esp_console_dev_uart_config_t* *dev_config, const *esp_console_repl_config_t* *repl_config, *esp_console_repl_t* **ret_repl)

Establish a console REPL environment over UART driver.

Attention This function is meant to be used in the examples to make the code more compact. Applications which use console functionality should be based on the underlying linenoise and esp_console functions.

备注: This is an all-in-one function to establish the environment needed for REPL, includes:

- Install the UART driver on the console UART (8n1, 115200, REF_TICK clock source)
- Configures the stdin/stdout to go through the UART driver
- Initializes linenoise

- Spawn new thread to run REPL in the background

参数

- **dev_config** -- **[in]** UART device configuration
- **repl_config** -- **[in]** REPL configuration
- **ret_repl** -- **[out]** return REPL handle after initialization succeed, return NULL otherwise

返回

- ESP_OK on success
- ESP_FAIL Parameter error

esp_err_t **esp_console_new_repl_usb_cdc**(const *esp_console_dev_usb_cdc_config_t* *dev_config, const *esp_console_repl_config_t* *repl_config, *esp_console_repl_t* **ret_repl)

Establish a console REPL environment over USB CDC.

Attention This function is meant to be used in the examples to make the code more compact. Applications which use console functionality should be based on the underlying linenoise and esp_console functions.

备注: This is a all-in-one function to establish the environment needed for REPL, includes:

- Initializes linenoise
- Spawn new thread to run REPL in the background

参数

- **dev_config** -- **[in]** USB CDC configuration
- **repl_config** -- **[in]** REPL configuration
- **ret_repl** -- **[out]** return REPL handle after initialization succeed, return NULL otherwise

返回

- ESP_OK on success
- ESP_FAIL Parameter error

esp_err_t **esp_console_new_repl_usb_serial_jtag**(const *esp_console_dev_usb_serial_jtag_config_t* *dev_config, const *esp_console_repl_config_t* *repl_config, *esp_console_repl_t* **ret_repl)

Establish a console REPL (Read-eval-print loop) environment over USB-SERIAL-JTAG.

Attention This function is meant to be used in the examples to make the code more compact. Applications which use console functionality should be based on the underlying linenoise and esp_console functions.

备注: This is an all-in-one function to establish the environment needed for REPL, includes:

- Initializes linenoise
- Spawn new thread to run REPL in the background

参数

- **dev_config** -- **[in]** USB-SERIAL-JTAG configuration
- **repl_config** -- **[in]** REPL configuration
- **ret_repl** -- **[out]** return REPL handle after initialization succeed, return NULL otherwise

返回

- ESP_OK on success
- ESP_FAIL Parameter error

esp_err_t **esp_console_start_repl**(*esp_console_repl_t* *repl)

Start REPL environment.

备注: Once the REPL gets started, it won't be stopped until the user calls repl->del(repl) to destroy the REPL environment.

参数 **repl** -- **[in]** REPL handle returned from esp_console_new_repl_xxx

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE, if repl has started already

Structures

struct **esp_console_config_t**

Parameters for console initialization.

Public Members

size_t **max_cmdline_length**

length of command line buffer, in bytes

size_t **max_cmdline_args**

maximum number of command line arguments to parse

uint32_t **heap_alloc_caps**

where to (e.g. MALLOC_CAP_SPIRAM) allocate heap objects such as cmds used by esp_console

int **hint_color**

ASCII color code of hint text.

int **hint_bold**

Set to 1 to print hint text in bold.

struct **esp_console_repl_config_t**

Parameters for console REPL (Read Eval Print Loop)

Public Members

uint32_t **max_history_len**

maximum length for the history

const char ***history_save_path**

file path used to save history commands, set to NULL won't save to file system

uint32_t **task_stack_size**

repl task stack size

uint32_t **task_priority**

repl task priority

BaseType_t **task_core_id**

repl task affinity, i.e. which core the task is pinned to

const char ***prompt**

prompt (NULL represents default: "esp> ")

size_t **max_cmdline_length**

maximum length of a command line. If 0, default value will be used

struct **esp_console_dev_uart_config_t**

Parameters for console device: UART.

Public Members

int **channel**

UART channel number (count from zero)

int **baud_rate**

Comunication baud rate.

int **tx_gpio_num**

GPIO number for TX path, -1 means using default one.

int **rx_gpio_num**

GPIO number for RX path, -1 means using default one.

struct **esp_console_dev_usb_cdc_config_t**

Parameters for console device: USB CDC.

备注: It's an empty structure for now, reserved for future

struct **esp_console_dev_usb_serial_jtag_config_t**

Parameters for console device: USB-SERIAL-JTAG.

备注: It's an empty structure for now, reserved for future

struct **esp_console_cmd_t**

Console command description.

Public Members

const char ***command**

Command name. Must not be NULL, must not contain spaces. The pointer must be valid until the call to esp_console_deinit.

const char ***help**

Help text for the command, shown by help command. If set, the pointer must be valid until the call to esp_console_deinit. If not set, the command will not be listed in 'help' output.

const char ***hint**

Hint text, usually lists possible arguments. If set to NULL, and 'argtable' field is non-NULL, hint will be generated automatically

esp_console_cmd_func_t **func**

Pointer to a function which implements the command.

备注: : Setting both `func` and `func_w_context` is not allowed.

void ***argtable**

Array or structure of pointers to arg_xxx structures, may be NULL. Used to generate hint text if 'hint' is set to NULL. Array/structure which this field points to must end with an arg_end. Only used for the duration of esp_console_cmd_register call.

esp_console_cmd_func_with_context_t **func_w_context**

Pointer to a context aware function which implements the command.

备注: : Setting both `func` and `func_w_context` is not allowed.

void ***context**

Context pointer to user-defined per-command context data. This is used if context aware function `func_w_context` is set.

struct **esp_console_repl_s**

Console REPL base structure.

Public Members

esp_err_t (***del**)(*esp_console_repl_t* *repl)

Delete console REPL environment.

Param repl **[in]** REPL handle returned from esp_console_new_repl_xxx

Return

- ESP_OK on success
- ESP_FAIL on errors

Macros

ESP_CONSOLE_CONFIG_DEFAULT()

Default console configuration value.

ESP_CONSOLE_REPL_CONFIG_DEFAULT()

Default console repl configuration value.

ESP_CONSOLE_DEV_UART_CONFIG_DEFAULT()

ESP_CONSOLE_DEV_CDC_CONFIG_DEFAULT()

ESP_CONSOLE_DEV_USB_SERIAL_JTAG_CONFIG_DEFAULT()

Type Definitions

typedef struct *linenoiseCompletions* **linenoiseCompletions**

typedef int (***esp_console_cmd_func_t**)(int argc, char **argv)

Console command main function.

Param argc number of arguments
Param argv array with argc entries, each pointing to a zero-terminated string argument
Return console command return code, 0 indicates ”success”

typedef int (***esp_console_cmd_func_with_context_t**)(void *context, int argc, char **argv)

Console command main function, with context.

Param context a user context given at invocation
Param argc number of arguments
Param argv array with argc entries, each pointing to a zero-terminated string argument
Return console command return code, 0 indicates ”success”

typedef struct *esp_console_repl_s* **esp_console_repl_t**

Type defined for console REPL.

2.9.7 eFuse Manager

Introduction

The eFuse Manager library is designed to structure access to eFuse bits and make using these easy. This library operates eFuse bits by a structure name which is assigned in eFuse table. This sections introduces some concepts used by eFuse Manager.

Hardware Description

The ESP32-P4 has a number of eFuses which can store system and user parameters. Each eFuse is a one-bit field which can be programmed to 1 after which it cannot be reverted back to 0. Some of system parameters are using these eFuse bits directly by hardware modules and have special place (for example EFUSE_BLK0).

For more details, see **ESP32-P4 Technical Reference Manual > eFuse Controller (eFuse)** [PDF]. Some eFuse bits are available for user applications.

ESP32-P4 has 11 eFuse blocks each of the size of 256 bits (not all bits are available):

- EFUSE_BLK0 is used entirely for system purposes;
- EFUSE_BLK1 is used entirely for system purposes;
- EFUSE_BLK2 is used entirely for system purposes;
- EFUSE_BLK3 (also named EFUSE_BLK_USER_DATA) can be used for user purposes;
- EFUSE_BLK4 (also named EFUSE_BLK_KEY0) can be used as key (for secure_boot or flash_encryption) or for user purposes;

- EFUSE_BLK5 (also named EFUSE_BLK_KEY1) can be used as key (for secure_boot or flash_encryption) or for user purposes;
- EFUSE_BLK6 (also named EFUSE_BLK_KEY2) can be used as key (for secure_boot or flash_encryption) or for user purposes;
- EFUSE_BLK7 (also named EFUSE_BLK_KEY3) can be used as key (for secure_boot or flash_encryption) or for user purposes;
- EFUSE_BLK8 (also named EFUSE_BLK_KEY4) can be used as key (for secure_boot or flash_encryption) or for user purposes;
- EFUSE_BLK9 (also named EFUSE_BLK_KEY5) can be used as key (for secure_boot or flash_encryption) or for user purposes;
- EFUSE_BLK10 (also named EFUSE_BLK_SYS_DATA_PART2) is reserved for system purposes.

Each block is divided into 8 32-bits registers.

eFuse Manager Component

The component has API functions for reading and writing fields. Access to the fields is carried out through the structures that describe the location of the eFuse bits in the blocks. The component provides the ability to form fields of any length and from any number of individual bits. The description of the fields is made in a CSV file in a table form. To generate from a tabular form (CSV file) in the C-source uses the tool `efuse_table_gen.py`. The tool checks the CSV file for uniqueness of field names and bit intersection, in case of using a *custom* file from the user's project directory, the utility checks with the *common* CSV file.

CSV files:

- common (*esp_efuse_table.csv*) - contains eFuse fields which are used inside the ESP-IDF. C-source generation should be done manually when changing this file (run command `idf.py efuse-common-table`). Note that changes in this file can lead to incorrect operation.
- custom - (optional and can be enabled by *CONFIG_EFUSE_CUSTOM_TABLE*) contains eFuse fields that are used by the user in their application. C-source generation should be done manually when changing this file and running `idf.py efuse-custom-table`.

Description CSV File

The CSV file contains a description of the eFuse fields. In the simple case, one field has one line of description. Table header:

```
# field_name,  efuse_block(EFUSE_BLK0..EFUSE_BLK10), bit_start(0..255),    bit_
↪count(1..256),         comment
```

Individual params in CSV file the following meanings:

field_name

> Name of field. The prefix *ESP_EFUSE_* is added to the name, and this field name is available in the code. This name is used to access the fields. The name must be unique for all fields. If the line has an empty name, then this line is combined with the previous field. This allows you to set an arbitrary order of bits in the field, and expand the field as well (see `MAC_FACTORY` field in the common table). The field_name supports structured format using . to show that the field belongs to another field (see `WR_DIS` and `RD_DIS` in the common table).

efuse_block

> Block number. It determines where the eFuse bits are placed for this field. Available EFUSE_BLK0..EFUSE_BLK10.

bit_start

> Start bit number (0..255). The bit_start field can be omitted. In this case, it is set to bit_start + bit_count from the previous record, if it has the same efuse_block. Otherwise (if efuse_block is different, or this is the first entry), an error will be generated.

bit_count

The number of bits to use in this field (1..-). This parameter cannot be omitted. This field also may be MAX_BLK_LEN in this case, the field length has the maximum block length.

comment

This param is using for comment field, it also move to C-header file. The comment field can be omitted.

If a non-sequential bit order is required to describe a field, then the field description in the following lines should be continued without specifying a name, indicating that it belongs to one field. For example two fields MAC_FACTORY and MAC_FACTORY_CRC:

```
# Factory MAC address #
#######################
MAC_FACTORY,            EFUSE_BLK0,     72,     8,    Factory MAC addr [0]
,                       EFUSE_BLK0,     64,     8,    Factory MAC addr [1]
,                       EFUSE_BLK0,     56,     8,    Factory MAC addr [2]
,                       EFUSE_BLK0,     48,     8,    Factory MAC addr [3]
,                       EFUSE_BLK0,     40,     8,    Factory MAC addr [4]
,                       EFUSE_BLK0,     32,     8,    Factory MAC addr [5]
MAC_FACTORY_CRC,        EFUSE_BLK0,     80,     8,    CRC8 for factory MAC address
```

This field is available in code as ESP_EFUSE_MAC_FACTORY and ESP_EFUSE_MAC_FACTORY_CRC.

Structured eFuse Fields

```
WR_DIS,                             EFUSE_BLK0,    0,     32,     Write protection
WR_DIS.RD_DIS,                      EFUSE_BLK0,    0,     1,      Write protection for␣
↪RD_DIS
WR_DIS.FIELD_1,                     EFUSE_BLK0,    1,     1,      Write protection for␣
↪FIELD_1
WR_DIS.FIELD_2,                     EFUSE_BLK0,    2,     4,      Write protection for␣
↪FIELD_2 (includes B1 and B2)
WR_DIS.FIELD_2.B1,                  EFUSE_BLK0,    2,     2,      Write protection for␣
↪FIELD_2.B1
WR_DIS.FIELD_2.B2,                  EFUSE_BLK0,    4,     2,      Write protection for␣
↪FIELD_2.B2
WR_DIS.FIELD_3,                     EFUSE_BLK0,    5,     1,      Write protection for␣
↪FIELD_3
WR_DIS.FIELD_3.ALIAS,               EFUSE_BLK0,    5,     1,      Write protection for␣
↪FIELD_3 (just a alias for WR_DIS.FIELD_3)
WR_DIS.FIELD_4,                     EFUSE_BLK0,    7,     1,      Write protection for␣
↪FIELD_4
```

The structured eFuse field looks like WR_DIS.RD_DIS where the dot points that this field belongs to the parent field - WR_DIS and cannot be out of the parent's range.

It is possible to use some levels of structured fields as WR_DIS.FIELD_2.B1 and B2. These fields should not be crossed each other and should be in the range of two fields: WR_DIS and WR_DIS.FIELD_2.

It is possible to create aliases for fields with the same range, see WR_DIS.FIELD_3 and WR_DIS.FIELD_3.ALIAS.

The ESP-IDF names for structured eFuse fields should be unique. The efuse_table_gen tool generates the final names where the dot is replaced by _. The names for using in ESP-IDF are ESP_EFUSE_WR_DIS, ESP_EFUSE_WR_DIS_RD_DIS, ESP_EFUSE_WR_DIS_FIELD_2_B1, etc.

The efuse_table_gen tool checks that the fields do not overlap each other and must be within the range of a field if there is a violation, then throws the following error:

```
Field at USER_DATA, EFUSE_BLK3, 0, 256  intersected with  SERIAL_NUMBER, EFUSE_
↪BLK3, 0, 32
```

Solution: Describe SERIAL_NUMBER to be included in USER_DATA. (USER_DATA.SERIAL_NUMBER).

```
Field at FIELD, EFUSE_BLK3, 0, 50  out of range  FIELD.MAJOR_NUMBER, EFUSE_BLK3,␣
↪60, 32
```

Solution: Change bit_start for FIELD.MAJOR_NUMBER from 60 to 0, so MAJOR_NUMBER is in the FIELD range.

efuse_table_gen.py Tool

The tool is designed to generate C-source files from CSV file and validate fields. First of all, the check is carried out on the uniqueness of the names and overlaps of the field bits. If an additional *custom* file is used, it will be checked with the existing *common* file (esp_efuse_table.csv). In case of errors, a message will be displayed and the string that caused the error. C-source files contain structures of type *esp_efuse_desc_t*.

To generate a *common* files, use the following command idf.py efuse-common-table or:

```
cd $IDF_PATH/components/efuse/
./efuse_table_gen.py --idf_target esp32p4 esp32p4/esp_efuse_table.csv
```

After generation in the folder $IDF_PATH/components/efuse/*esp32p4* create:

- *esp_efuse_table.c* file.
- In *include* folder *esp_efuse_table.c* file.

To generate a *custom* files, use the following command idf.py efuse-custom-table or:

```
cd $IDF_PATH/components/efuse/
./efuse_table_gen.py --idf_target esp32p4 esp32p4/esp_efuse_table.csv PROJECT_PATH/
↪main/esp_efuse_custom_table.csv
```

After generation in the folder PROJECT_PATH/main create:

- *esp_efuse_custom_table.c* file.
- In *include* folder *esp_efuse_custom_table.c* file.

To use the generated fields, you need to include two files:

```
#include "esp_efuse.h"
#include "esp_efuse_table.h" // or "esp_efuse_custom_table.h"
```

Supported Coding Scheme

Coding schemes are used to protect against data corruption. ESP32-P4 supports two coding schemes:

- None. EFUSE_BLK0 is stored with four backups, meaning each bit is stored four times. This backup scheme is automatically applied by the hardware and is not visible to software. EFUSE_BLK0 can be written many times.
- RS. EFUSE_BLK1 - EFUSE_BLK10 use Reed-Solomon coding scheme that supports up to 5 bytes of automatic error correction. Software encodes the 32-byte EFUSE_BLKx using RS (44, 32) to generate a 12-byte check code, and then burn the EFUSE_BLKx and the check code into eFuse at the same time. The eFuse Controller automatically decodes the RS encoding and applies error correction when reading back the eFuse block. Because the RS check codes are generated across the entire 256-bit eFuse block, each block can only be written to one time.

To write some fields into one block, or different blocks in one time, you need to use the batch writing mode. Firstly set this mode through *esp_efuse_batch_write_begin()* function then write some fields as usual using the esp_efuse_write_... functions. At the end to burn them, call the *esp_efuse_batch_write_commit()* function. It burns prepared data to the eFuse blocks and disables the batch recording mode.

备注： If there is already pre-written data in the eFuse block using the `Reed-Solomon` encoding scheme, then it is not possible to write anything extra (even if the required bits are empty) without breaking the previous encoding data. This encoding data will be overwritten with new encoding data and completely destroyed (however, the payload eFuses are not damaged). It can be related to: CUSTOM_MAC, SPI_PAD_CONFIG_HD, SPI_PAD_CONFIG_CS, etc. Please contact Espressif to order the required pre-burnt eFuses.

FOR TESTING ONLY (NOT RECOMMENDED): You can ignore or suppress errors that violate encoding scheme data in order to burn the necessary bits in the eFuse block.

eFuse API

Access to the fields is via a pointer to the description structure. API functions have some basic operation:

- *esp_efuse_read_field_blob()* - returns an array of read eFuse bits.
- *esp_efuse_read_field_cnt()* - returns the number of bits programmed as "1".
- *esp_efuse_write_field_blob()* - writes an array.
- *esp_efuse_write_field_cnt()* - writes a required count of bits as "1".
- *esp_efuse_get_field_size()* - returns the number of bits by the field name.
- *esp_efuse_read_reg()* - returns value of eFuse register.
- *esp_efuse_write_reg()* - writes value to eFuse register.
- *esp_efuse_get_coding_scheme()* - returns eFuse coding scheme for blocks.
- *esp_efuse_read_block()* - reads key to eFuse block starting at the offset and the required size.
- *esp_efuse_write_block()* - writes key to eFuse block starting at the offset and the required size.
- *esp_efuse_batch_write_begin()* - set the batch mode of writing fields.
- *esp_efuse_batch_write_commit()* - writes all prepared data for batch writing mode and reset the batch writing mode.
- *esp_efuse_batch_write_cancel()* - reset the batch writing mode and prepared data.
- *esp_efuse_get_key_dis_read()* - Returns a read protection for the key block.
- *esp_efuse_set_key_dis_read()* - Sets a read protection for the key block.
- *esp_efuse_get_key_dis_write()* - Returns a write protection for the key block.
- *esp_efuse_set_key_dis_write()* - Sets a write protection for the key block.
- *esp_efuse_get_key_purpose()* - Returns the current purpose set for an eFuse key block.
- *esp_efuse_write_key()* - Programs a block of key data to an eFuse block
- *esp_efuse_write_keys()* - Programs keys to unused eFuse blocks
- *esp_efuse_find_purpose()* - Finds a key block with the particular purpose set.
- *esp_efuse_get_keypurpose_dis_write()* - Returns a write protection of the key purpose field for an eFuse key block (for esp32 always true).
- *esp_efuse_key_block_unused()* - Returns true if the key block is unused, false otherwise.
- *esp_efuse_destroy_block()* - Destroys the data in this eFuse block. There are two things to do (1) if write protection is not set, then the remaining unset bits are burned, (2) set read protection for this block if it is not locked.

For frequently used fields, special functions are made, like this *esp_efuse_get_pkg_ver()*.

eFuse API for Keys

EFUSE_BLK_KEY0 - EFUSE_BLK_KEY5 are intended to keep up to 6 keys with a length of 256-bits. Each key has an `ESP_EFUSE_KEY_PURPOSE_x` field which defines the purpose of these keys. The purpose field is described in *esp_efuse_purpose_t*.

The purposes like `ESP_EFUSE_KEY_PURPOSE_XTS_AES_...` are used for flash encryption.

The purposes like `ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_DIGEST...` are used for secure boot.

There are some eFuse APIs useful to work with states of keys.

- *esp_efuse_get_purpose_field()* - Returns a pointer to a key purpose for an eFuse key block.
- *esp_efuse_get_key()* - Returns a pointer to a key block.

- *esp_efuse_set_key_purpose()* - Sets a key purpose for an eFuse key block.
- *esp_efuse_set_keypurpose_dis_write()* - Sets a write protection of the key purpose field for an eFuse key block.
- *esp_efuse_find_unused_key_block()* - Search for an unused key block and return the first one found.
- *esp_efuse_count_unused_key_blocks()* - Returns the number of unused eFuse key blocks in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
- *esp_efuse_get_digest_revoke()* - Returns the status of the Secure Boot public key digest revocation bit.
- *esp_efuse_set_digest_revoke()* - Sets the Secure Boot public key digest revocation bit.
- *esp_efuse_get_write_protect_of_digest_revoke()* - Returns a write protection of the Secure Boot public key digest revocation bit.
- *esp_efuse_set_write_protect_of_digest_revoke()* - Sets a write protection of the Secure Boot public key digest revocation bit.

How to Add a New Field

1. Find a free bits for field. Show *esp_efuse_table.csv* file or run `idf.py show-efuse-table` or the next command:

```
$ ./efuse_table_gen.py -t IDF_TARGET_PATH_NAME esp32p4/esp_efuse_table.csv --info

Max number of bits in BLK 256
Parsing efuse CSV input file ../esp32p4/esp_efuse_table.csv ...
Verifying efuse table...
Sorted efuse table:
#       field_name                      efuse_block     bit_start       bit_count
1       WR_DIS                          EFUSE_BLK0          0               32
2       WR_DIS.RD_DIS                   EFUSE_BLK0          0               1
3       WR_DIS.SPI_BOOT_CRYPT_CNT       EFUSE_BLK0          4               1
4       WR_DIS.SECURE_BOOT_KEY_REVOKE0  EFUSE_BLK0          5               1
5       WR_DIS.SECURE_BOOT_KEY_REVOKE1  EFUSE_BLK0          6               1
6       WR_DIS.SECURE_BOOT_KEY_REVOKE2  EFUSE_BLK0          7               1
7       WR_DIS.KEY_PURPOSE_0            EFUSE_BLK0          8               1
8       WR_DIS.KEY_PURPOSE_1            EFUSE_BLK0          9               1
9       WR_DIS.KEY_PURPOSE_2            EFUSE_BLK0          10              1
10      WR_DIS.KEY_PURPOSE_3            EFUSE_BLK0          11              1
11      WR_DIS.KEY_PURPOSE_4            EFUSE_BLK0          12              1
12      WR_DIS.KEY_PURPOSE_5            EFUSE_BLK0          13              1
13      WR_DIS.SECURE_BOOT_EN           EFUSE_BLK0          15              1
14      WR_DIS.BLK1                     EFUSE_BLK0          20              1
15      WR_DIS.MAC                      EFUSE_BLK0          20              1
16      WR_DIS.WAFER_VERSION_MINOR      EFUSE_BLK0          20              1
17      WR_DIS.WAFER_VERSION_MAJOR      EFUSE_BLK0          20              1
18      WR_DIS.DISABLE_WAFER_VERSION_MAJOR      EFUSE_BLK0          20              ␣
↪1
19      WR_DIS.DISABLE_BLK_VERSION_MAJOR        EFUSE_BLK0          20              ␣
↪1
20      WR_DIS.BLK_VERSION_MINOR        EFUSE_BLK0          20              1
21      WR_DIS.BLK_VERSION_MAJOR        EFUSE_BLK0          20              1
22      WR_DIS.FLASH_CAP                EFUSE_BLK0          20              1
23      WR_DIS.FLASH_TEMP               EFUSE_BLK0          20              1
24      WR_DIS.FLASH_VENDOR             EFUSE_BLK0          20              1
25      WR_DIS.PSRAM_CAP                EFUSE_BLK0          20              1
26      WR_DIS.PSRAM_TEMP               EFUSE_BLK0          20              1
27      WR_DIS.PSRAM_VENDOR             EFUSE_BLK0          20              1
28      WR_DIS.PKG_VERSION              EFUSE_BLK0          20              1
29      WR_DIS.OPTIONAL_UNIQUE_ID       EFUSE_BLK0          21              1
30      WR_DIS.BLOCK_USR_DATA           EFUSE_BLK0          22              1
31      WR_DIS.CUSTOM_MAC               EFUSE_BLK0          22              1
```

(下页继续)

(续上页)

```
32      WR_DIS.BLOCK_KEY0                EFUSE_BLK0          23                1
33      WR_DIS.BLOCK_KEY1                EFUSE_BLK0          24                1
34      WR_DIS.BLOCK_KEY2                EFUSE_BLK0          25                1
35      WR_DIS.BLOCK_KEY3                EFUSE_BLK0          26                1
36      WR_DIS.BLOCK_KEY4                EFUSE_BLK0          27                1
37      WR_DIS.BLOCK_KEY5                EFUSE_BLK0          28                1
38      WR_DIS.BLOCK_SYS_DATA2           EFUSE_BLK0          29                1
39      RD_DIS                           EFUSE_BLK0          32                7
40      RD_DIS.BLOCK_KEY0                EFUSE_BLK0          32                1
41      RD_DIS.BLOCK_KEY1                EFUSE_BLK0          33                1
42      RD_DIS.BLOCK_KEY2                EFUSE_BLK0          34                1
43      RD_DIS.BLOCK_KEY3                EFUSE_BLK0          35                1
44      RD_DIS.BLOCK_KEY4                EFUSE_BLK0          36                1
45      RD_DIS.BLOCK_KEY5                EFUSE_BLK0          37                1
46      RD_DIS.BLOCK_SYS_DATA2           EFUSE_BLK0          38                1
47      USB_DEVICE_EXCHG_PINS            EFUSE_BLK0          39                1
48      USB_OTG11_EXCHG_PINS             EFUSE_BLK0          40                1
49      DIS_USB_JTAG                     EFUSE_BLK0          41                1
50      POWERGLITCH_EN                   EFUSE_BLK0          42                1
51      DIS_FORCE_DOWNLOAD               EFUSE_BLK0          44                1
52      SPI_DOWNLOAD_MSPI_DIS            EFUSE_BLK0          45                1
53      DIS_TWAI                         EFUSE_BLK0          46                1
54      JTAG_SEL_ENABLE                  EFUSE_BLK0          47                1
55      SOFT_DIS_JTAG                    EFUSE_BLK0          48                3
56      DIS_PAD_JTAG                     EFUSE_BLK0          51                1
57      DIS_DOWNLOAD_MANUAL_ENCRYPT      EFUSE_BLK0          52                1
58      USB_PHY_SEL                      EFUSE_BLK0          57                1
59      KM_HUK_GEN_STATE                 EFUSE_BLK0          58                9
60      KM_RND_SWITCH_CYCLE              EFUSE_BLK0          67                2
61      KM_DEPLOY_ONLY_ONCE              EFUSE_BLK0          69                4
62      FORCE_USE_KEY_MANAGER_KEY        EFUSE_BLK0          73                4
63      FORCE_DISABLE_SW_INIT_KEY        EFUSE_BLK0          77                1
64      XTS_KEY_LENGTH_256               EFUSE_BLK0          78                1
65      WDT_DELAY_SEL                    EFUSE_BLK0          80                2
66      SPI_BOOT_CRYPT_CNT               EFUSE_BLK0          82                3
67      SECURE_BOOT_KEY_REVOKE0          EFUSE_BLK0          85                1
68      SECURE_BOOT_KEY_REVOKE1          EFUSE_BLK0          86                1
69      SECURE_BOOT_KEY_REVOKE2          EFUSE_BLK0          87                1
70      KEY_PURPOSE_0                    EFUSE_BLK0          88                4
71      KEY_PURPOSE_1                    EFUSE_BLK0          92                4
72      KEY_PURPOSE_2                    EFUSE_BLK0          96                4
73      KEY_PURPOSE_3                    EFUSE_BLK0         100                4
74      KEY_PURPOSE_4                    EFUSE_BLK0         104                4
75      KEY_PURPOSE_5                    EFUSE_BLK0         108                4
76      SEC_DPA_LEVEL                    EFUSE_BLK0         112                2
77      ECDSA_ENABLE_SOFT_K              EFUSE_BLK0         114                1
78      CRYPT_DPA_ENABLE                 EFUSE_BLK0         115                1
79      SECURE_BOOT_EN                   EFUSE_BLK0         116                1
80      SECURE_BOOT_AGGRESSIVE_REVOKE    EFUSE_BLK0         117                1
81      FLASH_TYPE                       EFUSE_BLK0         119                1
82      FLASH_PAGE_SIZE                  EFUSE_BLK0         120                2
83      FLASH_ECC_EN                     EFUSE_BLK0         122                1
84      DIS_USB_OTG_DOWNLOAD_MODE        EFUSE_BLK0         123                1
85      FLASH_TPUW                       EFUSE_BLK0         124                4
86      DIS_DOWNLOAD_MODE                EFUSE_BLK0         128                1
87      DIS_DIRECT_BOOT                  EFUSE_BLK0         129                1
88      DIS_USB_SERIAL_JTAG_ROM_PRINT    EFUSE_BLK0         130                1
89      LOCK_KM_KEY                      EFUSE_BLK0         131                1
90      DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE       EFUSE_BLK0         132              ␣
↪1
91      ENABLE_SECURITY_DOWNLOAD         EFUSE_BLK0         133                1
```

(下页继续)

(续上页)

```
92      UART_PRINT_CONTROL              EFUSE_BLK0        134                 2
93      FORCE_SEND_RESUME               EFUSE_BLK0        136                 1
94      SECURE_VERSION                  EFUSE_BLK0        137                16
95      SECURE_BOOT_DISABLE_FAST_WAKE   EFUSE_BLK0        153                 1
96      HYS_EN_PAD                      EFUSE_BLK0        154                 1
97      DCDC_VSET                       EFUSE_BLK0        155                 5
98      PXA0_TIEH_SEL_0                 EFUSE_BLK0        160                 2
99      PXA0_TIEH_SEL_1                 EFUSE_BLK0        162                 2
100     PXA0_TIEH_SEL_2                 EFUSE_BLK0        164                 2
101     PXA0_TIEH_SEL_3                 EFUSE_BLK0        166                 2
102     KM_DISABLE_DEPLOY_MODE          EFUSE_BLK0        168                 4
103     HP_PWR_SRC_SEL                  EFUSE_BLK0        178                 1
104     DCDC_VSET_EN                    EFUSE_BLK0        179                 1
105     DIS_WDT                         EFUSE_BLK0        180                 1
106     DIS_SWD                         EFUSE_BLK0        181                 1
107     MAC                             EFUSE_BLK1          0                 8
108     MAC                             EFUSE_BLK1          8                 8
109     MAC                             EFUSE_BLK1         16                 8
110     MAC                             EFUSE_BLK1         24                 8
111     MAC                             EFUSE_BLK1         32                 8
112     MAC                             EFUSE_BLK1         40                 8
113     WAFER_VERSION_MINOR             EFUSE_BLK1         64                 4
114     WAFER_VERSION_MAJOR             EFUSE_BLK1         68                 2
115     DISABLE_WAFER_VERSION_MAJOR     EFUSE_BLK1         70                 1
116     DISABLE_BLK_VERSION_MAJOR       EFUSE_BLK1         71                 1
117     BLK_VERSION_MINOR               EFUSE_BLK1         72                 3
118     BLK_VERSION_MAJOR               EFUSE_BLK1         75                 2
119     FLASH_CAP                       EFUSE_BLK1         77                 3
120     FLASH_TEMP                      EFUSE_BLK1         80                 2
121     FLASH_VENDOR                    EFUSE_BLK1         82                 3
122     PSRAM_CAP                       EFUSE_BLK1         85                 2
123     PSRAM_TEMP                      EFUSE_BLK1         87                 2
124     PSRAM_VENDOR                    EFUSE_BLK1         89                 2
125     PKG_VERSION                     EFUSE_BLK1         91                 3
126     SYS_DATA_PART2                  EFUSE_BLK10         0               256
127     OPTIONAL_UNIQUE_ID              EFUSE_BLK2          0               128
128     USER_DATA                       EFUSE_BLK3          0               256
129     USER_DATA.MAC_CUSTOM            EFUSE_BLK3        200                48
130     KEY0                            EFUSE_BLK4          0               256
131     KEY1                            EFUSE_BLK5          0               256
132     KEY2                            EFUSE_BLK6          0               256
133     KEY3                            EFUSE_BLK7          0               256
134     KEY4                            EFUSE_BLK8          0               256
135     KEY5                            EFUSE_BLK9          0               256

Used bits in efuse table:
EFUSE_BLK0
[0 31] [0 0] [4 13] [15 15] [20 20] [20 20] [20 29] [32 38] [32 42] [44 52] [57␣
↪78] [80 117] [119 171] [178 181]

EFUSE_BLK1
[0 47] [64 93]

EFUSE_BLK10
[0 255]

EFUSE_BLK2
[0 127]

EFUSE_BLK3
[0 255] [200 247]
```

(下页继续)

(续上页)

```
EFUSE_BLK4
[0 255]

EFUSE_BLK5
[0 255]

EFUSE_BLK6
[0 255]

EFUSE_BLK7
[0 255]

EFUSE_BLK8
[0 255]

EFUSE_BLK9
[0 255]
Note: Not printed ranges are free for using. (bits in EFUSE_BLK0 are reserved for␣
↪Espressif)
```

The number of bits not included in square brackets is free (some bits are reserved for Espressif). All fields are checked for overlapping.

To add fields to an existing field, use the *Structured efuse fields* technique. For example, adding the fields: SERIAL_NUMBER, MODEL_NUMBER and HARDWARE REV to an existing `USER_DATA` field. Use `.` (dot) to show an attachment in a field.

```
USER_DATA.SERIAL_NUMBER,                    EFUSE_BLK3,     0,  32,
USER_DATA.MODEL_NUMBER,                     EFUSE_BLK3,     32, 10,
USER_DATA.HARDWARE_REV,                     EFUSE_BLK3,     42, 10,
```

2. Fill a line for field: field_name, efuse_block, bit_start, bit_count, comment.
3. Run a `show_efuse_table` command to check eFuse table. To generate source files run `efuse_common_table` or `efuse_custom_table` command.

You may get errors such as `intersects with` or `out of range`. Please see how to solve them in the *Structured efuse fields* article.

Bit Order

The eFuses bit order is little endian (see the example below), it means that eFuse bits are read and written from LSB to MSB:

```
$ espefuse.py dump

USER_DATA       (BLOCK3          ) [3 ] read_regs: 03020100 07060504 0B0A0908␣
↪0F0E0D0C 13121111 17161514 1B1A1918 1F1E1D1C
BLOCK4          (BLOCK4          ) [4 ] read_regs: 03020100 07060504 0B0A0908␣
↪0F0E0D0C 13121111 17161514 1B1A1918 1F1E1D1C

where is the register representation:

EFUSE_RD_USR_DATA0_REG = 0x03020100
EFUSE_RD_USR_DATA1_REG = 0x07060504
EFUSE_RD_USR_DATA2_REG = 0x0B0A0908
EFUSE_RD_USR_DATA3_REG = 0x0F0E0D0C
EFUSE_RD_USR_DATA4_REG = 0x13121111
EFUSE_RD_USR_DATA5_REG = 0x17161514
EFUSE_RD_USR_DATA6_REG = 0x1B1A1918
```

(下页继续)

(续上页)

```
EFUSE_RD_USR_DATA7_REG = 0x1F1E1D1C

where is the byte representation:

byte[0] = 0x00, byte[1] = 0x01, ... byte[3] = 0x03, byte[4] = 0x04, ..., byte[31]␣
↪= 0x1F
```

For example, csv file describes the `USER_DATA` field, which occupies all 256 bits (a whole block).

```
USER_DATA,           EFUSE_BLK3,     0,  256,     User data
USER_DATA.FIELD1,    EFUSE_BLK3,     16,  16,     Field1

ID,                  EFUSE_BLK4,     8,  3,       ID bit[0..2]
,                    EFUSE_BLK4,     16, 2,       ID bit[3..4]
,                    EFUSE_BLK4,     32, 3,       ID bit[5..7]
```

Thus, reading the eFuse `USER_DATA` block written as above gives the following results:

```
uint8_t buf[32] = { 0 };
esp_efuse_read_field_blob(ESP_EFUSE_USER_DATA, &buf, sizeof(buf) * 8);
// buf[0] = 0x00, buf[1] = 0x01, ... buf[31] = 0x1F

uint32_t field1 = 0;
size_t field1_size = ESP_EFUSE_USER_DATA[0]->bit_count; // can be used for this␣
↪case because it only consists of one entry
esp_efuse_read_field_blob(ESP_EFUSE_USER_DATA, &field1, field1_size);
// field1 = 0x0302

uint32_t field1_1 = 0;
esp_efuse_read_field_blob(ESP_EFUSE_USER_DATA, &field1_1, 2); // reads only first␣
↪2 bits
// field1 = 0x0002

uint8_t id = 0;
size_t id_size = esp_efuse_get_field_size(ESP_EFUSE_ID); // returns 6
// size_t id_size = ESP_EFUSE_USER_DATA[0]->bit_count; // cannot be used because␣
↪it consists of 3 entries. It returns 3 not 6.
esp_efuse_read_field_blob(ESP_EFUSE_ID, &id, id_size);
// id = 0x91
// b'100 10  001
//   [3] [2] [3]

uint8_t id_1 = 0;
esp_efuse_read_field_blob(ESP_EFUSE_ID, &id_1, 3);
// id = 0x01
// b'001
```

Get eFuses During Build

There is a way to get the state of eFuses at the build stage of the project. There are two cmake functions for this:

- `espefuse_get_json_summary()` - It calls the `espefuse.py summary --format json` command and returns a json string (it is not stored in a file).
- `espefuse_get_efuse()` - It finds a given eFuse name in the json string and returns its property.

The json string has the following properties:

```
{
    "MAC": {
        "bit_len": 48,
```

(下页继续)

(续上页)

```
        "block": 0,
        "category": "identity",
        "description": "Factory MAC Address",
        "efuse_type": "bytes:6",
        "name": "MAC",
        "pos": 0,
        "readable": true,
        "value": "94:b9:7e:5a:6e:58 (CRC 0xe2 OK)",
        "word": 1,
        "writeable": true
    },
}
```

These functions can be used from a top-level project `CMakeLists.txt` (get-started/hello_world/CMakeLists.txt):

```
# ...
project(hello_world)

espefuse_get_json_summary(efuse_json)
espefuse_get_efuse(ret_data ${efuse_json} "MAC" "value")
message("MAC:" ${ret_data})
```

The format of the `value` property is the same as shown in `espefuse.py summary`.

```
MAC:94:b9:7e:5a:6e:58 (CRC 0xe2 OK)
```

There is an example test system/efuse/CMakeLists.txt which adds a custom target `efuse-summary`. This allows you to run the `idf.py efuse-summary` command to read the required eFuses (specified in the `efuse_names` list) at any time, not just at project build time.

Debug eFuse & Unit Tests

Virtual eFuses The Kconfig option *CONFIG_EFUSE_VIRTUAL* virtualizes eFuse values inside the eFuse Manager, so writes are emulated and no eFuse values are permanently changed. This can be useful for debugging app and unit tests. During startup, the eFuses are copied to RAM. All eFuse operations (read and write) are performed with RAM instead of the real eFuse registers.

In addition to the *CONFIG_EFUSE_VIRTUAL* option there is *CONFIG_EFUSE_VIRTUAL_KEEP_IN_FLASH* option that adds a feature to keep eFuses in flash memory. To use this mode the partition_table should have the *efuse* partition. partition.csv: `"efuse_em, data, efuse, , 0x2000,"`. During startup, the eFuses are copied from flash or, in case if flash is empty, from real eFuse to RAM and then update flash. This option allows keeping eFuses after reboots (possible to test secure_boot and flash_encryption features with this option).

Flash Encryption Testing Flash Encryption (FE) is a hardware feature that requires the physical burning of eFuses: key and FLASH_CRYPT_CNT. If FE is not actually enabled then enabling the *CONFIG_EFUSE_VIRTUAL_KEEP_IN_FLASH* option just gives testing possibilities and does not encrypt anything in the flash, even though the logs say encryption happens. The `bootloader_flash_write()` is adapted for this purpose. But if FE is already enabled on the chip and you run an application or bootloader created with the *CONFIG_EFUSE_VIRTUAL_KEEP_IN_FLASH* option then the flash encryption/decryption operations will work properly (data are encrypted as it is written into an encrypted flash partition and decrypted when they are read from an encrypted partition).

`espefuse.py` esptool includes a useful tool for reading/writing ESP32-P4 eFuse bits - espefuse.py.

```
espefuse.py -p PORT summary

espefuse.py v4.7.0
```

(下页继续)

(续上页)

```
Connecting....
Detecting chip type... ESP32-P4

=== Run "summary" command ===
EFUSE_NAME (Block) Description  = [Meaningful Value] [Readable/Writeable] (Hex␣
↪Value)
----------------------------------------------------------------------------------
↪-----
Config fuses:
WR_DIS (BLOCK0)                                          Disable programming of␣
↪individual eFuses           = 0 R/W (0x00000000)
RD_DIS (BLOCK0)                                          Disable reading from BlOCK4-10 ␣
↪                    = 0 R/W (0b0000000)
POWERGLITCH_EN (BLOCK0)                                  Represents whether power glitch␣
↪function is enable = False R/W (0b0)
                                                         d. 1: enabled. 0: disabled
DIS_TWAI (BLOCK0)                                        Represents whether TWAI␣
↪function is disabled or en = False R/W (0b0)
                                                         abled. 1: disabled. 0: enabled
KM_HUK_GEN_STATE (BLOCK0)                                Set this bit to control␣
↪validation of HUK generate = 0 R/W (0b000000000)
                                                          mode. Odd of 1 is invalid;␣
↪even of 1 is valid
KM_RND_SWITCH_CYCLE (BLOCK0)                             Set bits to control key manager␣
↪random number swit = 0 R/W (0b00)
                                                         ch cycle. 0: control by␣
↪register. 1: 8 km clk cycl
                                                         es. 2: 16 km cycles. 3: 32 km␣
↪cycles
KM_DEPLOY_ONLY_ONCE (BLOCK0)                             Set each bit to control whether␣
↪corresponding key  = 0 R/W (0x0)
                                                         can only be deployed once. 1 is␣
↪true; 0 is false.
                                                         Bit0: ecdsa. Bit1: xts. Bit2:␣
↪hmac. Bit3: ds
DIS_DIRECT_BOOT (BLOCK0)                                 Represents whether direct boot␣
↪mode is disabled or = False R/W (0b0)
                                                          enabled. 1: disabled. 0:␣
↪enabled
UART_PRINT_CONTROL (BLOCK0)                              Represents the type of UART␣
↪printing. 00: force en = 0 R/W (0b00)
                                                         able printing. 01: enable␣
↪printing when GPIO8 is r
                                                         eset at low level. 10: enable␣
↪printing when GPIO8
                                                         is reset at high level. 11:␣
↪force disable printing
HYS_EN_PAD (BLOCK0)                                      Represents whether the␣
↪hysteresis function of corr = False R/W (0b0)
                                                         corresponding PAD is enabled.␣
↪1: enabled. 0:disabled
DCDC_VSET (BLOCK0)                                       Set the dcdc voltage default   ␣
↪                    = 0 R/W (0b00000)
PXA0_TIEH_SEL_0 (BLOCK0)                                 TBD                            ␣
↪                    = 0 R/W (0b00)
PXA0_TIEH_SEL_1 (BLOCK0)                                 TBD                            ␣
↪                    = 0 R/W (0b00)
PXA0_TIEH_SEL_2 (BLOCK0)                                 TBD                            ␣
↪                    = 0 R/W (0b00)
PXA0_TIEH_SEL_3 (BLOCK0)                                 TBD                            ␣
↪                    = 0 R/W (0b00)
```

(下页继续)

(续上页)

```
KM_DISABLE_DEPLOY_MODE (BLOCK0)                    TBD
↪                    = 0 R/W (0x0)
HP_PWR_SRC_SEL (BLOCK0)                            HP system power source select.␣
↪0:LDO. 1: DCDC      = False R/W (0b0)
DCDC_VSET_EN (BLOCK0)                              Select dcdc vset use efuse_dcdc_
↪vset                = False R/W (0b0)
DIS_SWD (BLOCK0)                                   Set this bit to disable super-
↪watchdog              = False R/W (0b0)
PSRAM_CAP (BLOCK1)                                 PSRAM capacity                 ␣
↪                    = 0 R/W (0b00)
PSRAM_TEMP (BLOCK1)                                PSRAM temperature              ␣
↪                    = 0 R/W (0b00)
PSRAM_VENDOR (BLOCK1)                              PSRAM vendor                   ␣
↪                    = 0 R/W (0b00)
BLOCK_USR_DATA (BLOCK3)                            User data
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00 00 00 00 00 00 R/W
BLOCK_SYS_DATA2 (BLOCK10)                          System data part 2 (reserved)
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00 00 00 00 00 00 R/W

Flash fuses:
FLASH_TYPE (BLOCK0)                                The type of interfaced flash.␣
↪0: four data lines;  = False R/W (0b0)
                                                   1: eight data lines
FLASH_PAGE_SIZE (BLOCK0)                           Set flash page size            ␣
↪                    = 0 R/W (0b00)
FLASH_ECC_EN (BLOCK0)                              Set this bit to enable ecc for␣
↪flash boot            = False R/W (0b0)
FLASH_TPUW (BLOCK0)                                Represents the flash waiting␣
↪time after power-up;  = 0 R/W (0x0)
                                                   in unit of ms. When the value␣
↪less than 15; the wa
                                                   iting time is the programmed␣
↪value. Otherwise; the
                                                    waiting time is 2 times the␣
↪programmed value
FORCE_SEND_RESUME (BLOCK0)                         Represents whether ROM code is␣
↪forced to send a re = False R/W (0b0)
                                                   sume command during SPI boot.␣
↪1: forced. 0:not for
                                                   ced
FLASH_CAP (BLOCK1)                                 Flash capacity                 ␣
↪                    = 0 R/W (0b000)
FLASH_TEMP (BLOCK1)                                Flash temperature              ␣
↪                    = 0 R/W (0b00)
FLASH_VENDOR (BLOCK1)                              Flash vendor                   ␣
↪                    = 0 R/W (0b000)

Identity fuses:
WAFER_VERSION_MINOR (BLOCK1)                       Minor chip version             ␣
↪                    = 0 R/W (0x0)
WAFER_VERSION_MAJOR (BLOCK1)                       Major chip version             ␣
↪                    = 0 R/W (0b00)
DISABLE_WAFER_VERSION_MAJOR (BLOCK1)               Disables check of wafer version␣
↪major               = False R/W (0b0)
DISABLE_BLK_VERSION_MAJOR (BLOCK1)                 Disables check of blk version␣
↪major                 = False R/W (0b0)
BLK_VERSION_MINOR (BLOCK1)                         BLK_VERSION_MINOR of BLOCK2    ␣
↪                    = 0 R/W (0b000)
BLK_VERSION_MAJOR (BLOCK1)                         BLK_VERSION_MAJOR of BLOCK2    ␣
↪                    = 0 R/W (0b00)
```

(下页继续)

(续上页)

```
PKG_VERSION (BLOCK1)                                    Package version
↪                    = 0 R/W (0b000)
OPTIONAL_UNIQUE_ID (BLOCK2)                             Optional unique 128-bit ID
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W

Jtag fuses:
JTAG_SEL_ENABLE (BLOCK0)                                Represents whether the
↪selection between usb_to_jt = False R/W (0b0)
                                                        ag and pad_to_jtag through
↪strapping gpio15 when b
                                                        oth EFUSE_DIS_PAD_JTAG and
↪EFUSE_DIS_USB_JTAG are
                                                        equal to 0 is enabled or
↪disabled. 1: enabled. 0:
                                                        disabled
SOFT_DIS_JTAG (BLOCK0)                                  Represents whether JTAG is
↪disabled in soft way. O = 0 R/W (0b000)
                                                        dd number: disabled. Even
↪number: enabled
DIS_PAD_JTAG (BLOCK0)                                   Represents whether JTAG is
↪disabled in the hard wa = False R/W (0b0)
                                                        y(permanently). 1: disabled. 0:
↪enabled

Mac fuses:
MAC (BLOCK1)                                            MAC address
   = 60:55:f9:f8:80:40 (OK) R/W
CUSTOM_MAC (BLOCK3)                                     Custom MAC
   = 00:00:00:00:00:00 (OK) R/W

Security fuses:
DIS_FORCE_DOWNLOAD (BLOCK0)                             Represents whether the function
↪that forces chip i = False R/W (0b0)
                                                        nto download mode is disabled
↪or enabled. 1: disab
                                                        led. 0: enabled
SPI_DOWNLOAD_MSPI_DIS (BLOCK0)                          Set this bit to disable
↪accessing MSPI flash/MSPI  = False R/W (0b0)
                                                        ram by SYS AXI matrix during
↪boot_mode_download
DIS_DOWNLOAD_MANUAL_ENCRYPT (BLOCK0)                    Represents whether flash
↪encrypt function is disab = False R/W (0b0)
                                                        led or enabled(except in SPI
↪boot mode). 1: disable
                                                        ed. 0: enabled
FORCE_USE_KEY_MANAGER_KEY (BLOCK0)                      Set each bit to control whether
↪corresponding key  = 0 R/W (0x0)
                                                        must come from key manager.. 1
↪is true; 0 is false
                                                        . Bit0: ecdsa. Bit1: xts. Bit2:
↪hmac. Bit3: ds
FORCE_DISABLE_SW_INIT_KEY (BLOCK0)                      Set this bit to disable
↪software written init key; = False R/W (0b0)
                                                         and force use efuse_init_key
XTS_KEY_LENGTH_256 (BLOCK0)                             Set this bit to configure flash
↪encryption use xts = False R/W (0b0)
                                                        -128 key; else use xts-256 key
SPI_BOOT_CRYPT_CNT (BLOCK0)                             Enables flash encryption when 1
↪or 3 bits are set  = Disable R/W (0b000)
                                                        and disables otherwise
SECURE_BOOT_KEY_REVOKE0 (BLOCK0)                        Revoke 1st secure boot key
↪                    = False R/W (0b0)
```

(下页继续)

(续上页)

```
SECURE_BOOT_KEY_REVOKE1 (BLOCK0)                    Revoke 2nd secure boot key
 ↪                    = False R/W (0b0)
SECURE_BOOT_KEY_REVOKE2 (BLOCK0)                    Revoke 3rd secure boot key
 ↪                    = False R/W (0b0)
KEY_PURPOSE_0 (BLOCK0)                              Represents the purpose of Key0
 ↪                    = USER R/W (0x0)
KEY_PURPOSE_1 (BLOCK0)                              Represents the purpose of Key1
 ↪                    = USER R/W (0x0)
KEY_PURPOSE_2 (BLOCK0)                              Represents the purpose of Key2
 ↪                    = USER R/W (0x0)
KEY_PURPOSE_3 (BLOCK0)                              Represents the purpose of Key3
 ↪                    = USER R/W (0x0)
KEY_PURPOSE_4 (BLOCK0)                              Represents the purpose of Key4
 ↪                    = USER R/W (0x0)
KEY_PURPOSE_5 (BLOCK0)                              Represents the purpose of Key5
 ↪                    = USER R/W (0x0)
SEC_DPA_LEVEL (BLOCK0)                              Represents the spa secure level
 ↪by configuring the = 0 R/W (0b00)
                                                     clock random divide mode
ECDSA_ENABLE_SOFT_K (BLOCK0)                        Represents whether hardware
 ↪random number k is for = False R/W (0b0)
                                                    ced used in ESDCA. 1: force
 ↪used. 0: not force use
                                                    d
CRYPT_DPA_ENABLE (BLOCK0)                           Represents whether anti-dpa
 ↪attack is enabled. 1:e = False R/W (0b0)
                                                    nabled. 0: disabled
SECURE_BOOT_EN (BLOCK0)                             Represents whether secure boot
 ↪is enabled or disab = False R/W (0b0)
                                                    led. 1: enabled. 0: disabled
SECURE_BOOT_AGGRESSIVE_REVOKE (BLOCK0)              Represents whether revoking
 ↪aggressive secure boot = False R/W (0b0)
                                                     is enabled or disabled. 1:
 ↪enabled. 0: disabled
DIS_DOWNLOAD_MODE (BLOCK0)                          Represents whether Download
 ↪mode is disabled or en = False R/W (0b0)
                                                    abled. 1: disabled. 0: enabled
LOCK_KM_KEY (BLOCK0)                                TBD
 ↪                    = False R/W (0b0)
ENABLE_SECURITY_DOWNLOAD (BLOCK0)                   Represents whether security
 ↪download is enabled or = False R/W (0b0)
                                                     disabled. 1: enabled. 0:
 ↪disabled
SECURE_VERSION (BLOCK0)                             Represents the version used by
 ↪ESP-IDF anti-rollba = 0 R/W (0x0000)
                                                    ck feature
SECURE_BOOT_DISABLE_FAST_WAKE (BLOCK0)              Represents whether FAST VERIFY
 ↪ON WAKE is disabled = False R/W (0b0)
                                                     or enabled when Secure Boot is
 ↪enabled. 1: disable
                                                    ed. 0: enabled
BLOCK_KEY0 (BLOCK4)
  Purpose: USER
                Key0 or user data
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 ↪00 00 00 00 00 00 R/W
BLOCK_KEY1 (BLOCK5)
  Purpose: USER
                Key1 or user data
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 ↪00 00 00 00 00 00 R/W
```

(下页继续)

(续上页)

```
BLOCK_KEY2 (BLOCK6)
  Purpose: USER
               Key2 or user data
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00 00 00 00 00 00 R/W
BLOCK_KEY3 (BLOCK7)
  Purpose: USER
               Key3 or user data
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00 00 00 00 00 00 R/W
BLOCK_KEY4 (BLOCK8)
  Purpose: USER
               Key4 or user data
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00 00 00 00 00 00 R/W
BLOCK_KEY5 (BLOCK9)
  Purpose: USER
               Key5 or user data
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00 00 00 00 00 00 R/W

Usb fuses:
USB_DEVICE_EXCHG_PINS (BLOCK0)                          Enable usb device exchange pins␣
↪of D+ and D-       = False R/W (0b0)
USB_OTG11_EXCHG_PINS (BLOCK0)                           Enable usb otg11 exchange pins␣
↪of D+ and D-        = False R/W (0b0)
DIS_USB_JTAG (BLOCK0)                                   Represents whether the function␣
↪of usb switch to j = False R/W (0b0)
                                                        tag is disabled or enabled. 1:␣
↪disabled. 0: enable
                                                        d
USB_PHY_SEL (BLOCK0)                                    TBD                             ␣
↪                   = False R/W (0b0)
DIS_USB_OTG_DOWNLOAD_MODE (BLOCK0)                      Set this bit to disable␣
↪download via USB-OTG       = False R/W (0b0)
DIS_USB_SERIAL_JTAG_ROM_PRINT (BLOCK0)                  Represents whether print from␣
↪USB-Serial-JTAG is d = False R/W (0b0)
                                                        isabled or enabled. 1: disabled.
↪ 0: enabled
DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE (BLOCK0)              Represents whether the USB-
↪Serial-JTAG download fu = False R/W (0b0)
                                                        nction is disabled or enabled.␣
↪1: disabled. 0: ena
                                                        bled


Wdt fuses:
WDT_DELAY_SEL (BLOCK0)                                  Represents whether RTC watchdog␣
↪timeout threshold  = 0 R/W (0b00)
                                                        is selected at startup. 1:␣
↪selected. 0: not select
                                                        ed
DIS_WDT (BLOCK0)                                        Set this bit to disable watch␣
↪dog                  = False R/W (0b0)
```

To get a dump for all eFuse registers.

```
espefuse.py v4.7.dev1
Connecting....
Detecting chip type... ESP32-P4
BLOCK0          (                ) [0 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000
```

(下页继续)

(续上页)

```
MAC_SPI_8M_0     (BLOCK1           ) [1 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000
BLOCK_SYS_DATA   (BLOCK2           ) [2 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK_USR_DATA   (BLOCK3           ) [3 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK_KEY0       (BLOCK4           ) [4 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK_KEY1       (BLOCK5           ) [5 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK_KEY2       (BLOCK6           ) [6 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK_KEY3       (BLOCK7           ) [7 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK_KEY4       (BLOCK8           ) [8 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK_KEY5       (BLOCK9           ) [9 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK_SYS_DATA2  (BLOCK10          ) [10] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000
BLOCK0           (                 ) [0 ] err__regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000
EFUSE_RD_RS_ERR0_REG        0x00000000
EFUSE_RD_RS_ERR1_REG        0x00000000
=== Run "dump" command ===
```

Header File

- components/efuse/esp32p4/include/esp_efuse_chip.h
- This header file can be included with:

```
#include "esp_efuse_chip.h"
```

- This header file is a part of the API provided by the `efuse` component. To declare that your component depends on `efuse`, add the following to your CMakeLists.txt:

```
REQUIRES efuse
```

or

```
PRIV_REQUIRES efuse
```

Enumerations

enum **esp_efuse_block_t**

Type of eFuse blocks ESP32P4.

Values:

enumerator **EFUSE_BLK0**

Number of eFuse BLOCK0. REPEAT_DATA

enumerator **EFUSE_BLK1**

Number of eFuse BLOCK1. MAC_SPI_8M_SYS

enumerator **EFUSE_BLK2**

Number of eFuse BLOCK2. SYS_DATA_PART1

enumerator **EFUSE_BLK_SYS_DATA_PART1**
Number of eFuse BLOCK2. SYS_DATA_PART1

enumerator **EFUSE_BLK3**
Number of eFuse BLOCK3. USER_DATA

enumerator **EFUSE_BLK_USER_DATA**
Number of eFuse BLOCK3. USER_DATA

enumerator **EFUSE_BLK4**
Number of eFuse BLOCK4. KEY0

enumerator **EFUSE_BLK_KEY0**
Number of eFuse BLOCK4. KEY0

enumerator **EFUSE_BLK5**
Number of eFuse BLOCK5. KEY1

enumerator **EFUSE_BLK_KEY1**
Number of eFuse BLOCK5. KEY1

enumerator **EFUSE_BLK6**
Number of eFuse BLOCK6. KEY2

enumerator **EFUSE_BLK_KEY2**
Number of eFuse BLOCK6. KEY2

enumerator **EFUSE_BLK7**
Number of eFuse BLOCK7. KEY3

enumerator **EFUSE_BLK_KEY3**
Number of eFuse BLOCK7. KEY3

enumerator **EFUSE_BLK8**
Number of eFuse BLOCK8. KEY4

enumerator **EFUSE_BLK_KEY4**
Number of eFuse BLOCK8. KEY4

enumerator **EFUSE_BLK9**
Number of eFuse BLOCK9. KEY5

enumerator **EFUSE_BLK_KEY5**
Number of eFuse BLOCK9. KEY5

enumerator **EFUSE_BLK_KEY_MAX**

enumerator **EFUSE_BLK10**
Number of eFuse BLOCK10. SYS_DATA_PART2

enumerator **EFUSE_BLK_SYS_DATA_PART2**

Number of eFuse BLOCK10. SYS_DATA_PART2

enumerator **EFUSE_BLK_MAX**

enum **esp_efuse_coding_scheme_t**

Type of coding scheme.

Values:

enumerator **EFUSE_CODING_SCHEME_NONE**

None

enumerator **EFUSE_CODING_SCHEME_RS**

Reed-Solomon coding

enum **esp_efuse_purpose_t**

Type of key purpose.

Values:

enumerator **ESP_EFUSE_KEY_PURPOSE_USER**

User purposes (software-only use)

enumerator **ESP_EFUSE_KEY_PURPOSE_ECDSA_KEY**

ECDSA private key (Expected in little endian order)

enumerator **ESP_EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_1**

XTS_AES_256_KEY_1 (flash/PSRAM encryption)

enumerator **ESP_EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_2**

XTS_AES_256_KEY_2 (flash/PSRAM encryption)

enumerator **ESP_EFUSE_KEY_PURPOSE_XTS_AES_128_KEY**

XTS_AES_128_KEY (flash/PSRAM encryption)

enumerator **ESP_EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL**

HMAC Downstream mode

enumerator **ESP_EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG**

JTAG soft enable key (uses HMAC Downstream mode)

enumerator **ESP_EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE**

Digital Signature peripheral key (uses HMAC Downstream mode)

enumerator **ESP_EFUSE_KEY_PURPOSE_HMAC_UP**

HMAC Upstream mode

enumerator **ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_DIGEST0**

SECURE_BOOT_DIGEST0 (Secure Boot key digest)

enumerator **ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_DIGEST1**

SECURE_BOOT_DIGEST1 (Secure Boot key digest)

enumerator **ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_DIGEST2**

SECURE_BOOT_DIGEST2 (Secure Boot key digest)

enumerator **ESP_EFUSE_KEY_PURPOSE_KM_INIT_KEY**

KM_INIT_KEY

enumerator **ESP_EFUSE_KEY_PURPOSE_MAX**

MAX PURPOSE

Header File

- components/efuse/include/esp_efuse.h
- This header file can be included with:

```
#include "esp_efuse.h"
```

- This header file is a part of the API provided by the `efuse` component. To declare that your component depends on `efuse`, add the following to your CMakeLists.txt:

```
REQUIRES efuse
```

or

```
PRIV_REQUIRES efuse
```

Functions

esp_err_t **esp_efuse_read_field_blob**(const *esp_efuse_desc_t* *field[], void *dst, size_t dst_size_bits)

Reads bits from EFUSE field and writes it into an array.

The number of read bits will be limited to the minimum value from the description of the bits in "field" structure or "dst_size_bits" required size. Use "esp_efuse_get_field_size()" function to determine the length of the field.

备注: Please note that reading in the batch mode does not show uncommitted changes.

参数

- **field** -- **[in]** A pointer to the structure describing the fields of efuse.
- **dst** -- **[out]** A pointer to array that will contain the result of reading.
- **dst_size_bits** -- **[in]** The number of bits required to read. If the requested number of bits is greater than the field, the number will be limited to the field size.

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.

bool **esp_efuse_read_field_bit**(const *esp_efuse_desc_t* *field[])

Read a single bit eFuse field as a boolean value.

备注: The value must exist and must be a single bit wide. If there is any possibility of an error in the provided arguments, call esp_efuse_read_field_blob() and check the returned value instead.

备注： If assertions are enabled and the parameter is invalid, execution will abort

备注： Please note that reading in the batch mode does not show uncommitted changes.

参数 **`field`** -- **[in]** A pointer to the structure describing the fields of efuse.

返回

- true: The field parameter is valid and the bit is set.
- false: The bit is not set, or the parameter is invalid and assertions are disabled.

esp_err_t **`esp_efuse_read_field_cnt`**(const *esp_efuse_desc_t* *field[], size_t *out_cnt)

Reads bits from EFUSE field and returns number of bits programmed as ”1”.

If the bits are set not sequentially, they will still be counted.

备注： Please note that reading in the batch mode does not show uncommitted changes.

参数

- **`field`** -- **[in]** A pointer to the structure describing the fields of efuse.
- **`out_cnt`** -- **[out]** A pointer that will contain the number of programmed as ”1” bits.

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.

esp_err_t **`esp_efuse_write_field_blob`**(const *esp_efuse_desc_t* *field[], const void *src, size_t src_size_bits)

Writes array to EFUSE field.

The number of write bits will be limited to the minimum value from the description of the bits in ”field” structure or ”src_size_bits” required size. Use ”esp_efuse_get_field_size()” function to determine the length of the field. After the function is completed, the writing registers are cleared.

参数

- **`field`** -- **[in]** A pointer to the structure describing the fields of efuse.
- **`src`** -- **[in]** A pointer to array that contains the data for writing.
- **`src_size_bits`** -- **[in]** The number of bits required to write.

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

esp_err_t **`esp_efuse_write_field_cnt`**(const *esp_efuse_desc_t* *field[], size_t cnt)

Writes a required count of bits as ”1” to EFUSE field.

If there are no free bits in the field to set the required number of bits to ”1”, ESP_ERR_EFUSE_CNT_IS_FULL error is returned, the field will not be partially recorded. After the function is completed, the writing registers are cleared.

参数

- **`field`** -- **[in]** A pointer to the structure describing the fields of efuse.
- **`cnt`** -- **[in]** Required number of programmed as ”1” bits.

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.

- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.

esp_err_t **esp_efuse_write_field_bit** (const *esp_efuse_desc_t* *field[])

Write a single bit eFuse field to 1.

For use with eFuse fields that are a single bit. This function will write the bit to value 1 if it is not already set, or does nothing if the bit is already set.

This is equivalent to calling esp_efuse_write_field_cnt() with the cnt parameter equal to 1, except that it will return ESP_OK if the field is already set to 1.

参数 **field** -- **[in]** Pointer to the structure describing the efuse field.

返回

- ESP_OK: The operation was successfully completed, or the bit was already set to value 1.
- ESP_ERR_INVALID_ARG: Error in the passed arugments, including if the efuse field is not 1 bit wide.

esp_err_t **esp_efuse_set_write_protect** (*esp_efuse_block_t* blk)

Sets a write protection for the whole block.

After that, it is impossible to write to this block. The write protection does not apply to block 0.

参数 **blk** -- **[in]** Block number of eFuse. (EFUSE_BLK1, EFUSE_BLK2 and EFUSE_BLK3)

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.
- ESP_ERR_NOT_SUPPORTED: The block does not support this command.

esp_err_t **esp_efuse_set_read_protect** (*esp_efuse_block_t* blk)

Sets a read protection for the whole block.

After that, it is impossible to read from this block. The read protection does not apply to block 0.

参数 **blk** -- **[in]** Block number of eFuse. (EFUSE_BLK1, EFUSE_BLK2 and EFUSE_BLK3)

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.
- ESP_ERR_NOT_SUPPORTED: The block does not support this command.

int **esp_efuse_get_field_size** (const *esp_efuse_desc_t* *field[])

Returns the number of bits used by field.

参数 **field** -- **[in]** A pointer to the structure describing the fields of efuse.

返回 Returns the number of bits used by field.

uint32_t **esp_efuse_read_reg** (*esp_efuse_block_t* blk, unsigned int num_reg)

Returns value of efuse register.

This is a thread-safe implementation. Example: EFUSE_BLK2_RDATA3_REG where (blk=2, num_reg=3)

备注: Please note that reading in the batch mode does not show uncommitted changes.

参数

- **blk** -- **[in]** Block number of eFuse.
- **num_reg** -- **[in]** The register number in the block.

返回 Value of register

esp_err_t **esp_efuse_write_reg**(*esp_efuse_block_t* blk, unsigned int num_reg, uint32_t val)

Write value to efuse register.

Apply a coding scheme if necessary. This is a thread-safe implementation. Example: EFUSE_BLK3_WDATA0_REG where (blk=3, num_reg=0)

参数

- **blk** -- **[in]** Block number of eFuse.
- **num_reg** -- **[in]** The register number in the block.
- **val** -- **[in]** Value to write.

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.

esp_efuse_coding_scheme_t **esp_efuse_get_coding_scheme**(*esp_efuse_block_t* blk)

Return efuse coding scheme for blocks.

备注: The coding scheme is applicable only to 1, 2 and 3 blocks. For 0 block, the coding scheme is always NONE.

参数 **blk** -- **[in]** Block number of eFuse.
返回 Return efuse coding scheme for blocks

esp_err_t **esp_efuse_read_block**(*esp_efuse_block_t* blk, void *dst_key, size_t offset_in_bits, size_t size_bits)

Read key to efuse block starting at the offset and the required size.

备注: Please note that reading in the batch mode does not show uncommitted changes.

参数

- **blk** -- **[in]** Block number of eFuse.
- **dst_key** -- **[in]** A pointer to array that will contain the result of reading.
- **offset_in_bits** -- **[in]** Start bit in block.
- **size_bits** -- **[in]** The number of bits required to read.

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

esp_err_t **esp_efuse_write_block**(*esp_efuse_block_t* blk, const void *src_key, size_t offset_in_bits, size_t size_bits)

Write key to efuse block starting at the offset and the required size.

参数

- **blk** -- **[in]** Block number of eFuse.
- **src_key** -- **[in]** A pointer to array that contains the key for writing.
- **offset_in_bits** -- **[in]** Start bit in block.
- **size_bits** -- **[in]** The number of bits required to write.

返回

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits

uint32_t **esp_efuse_get_pkg_ver**(void)

Returns chip package from efuse.

返回 chip package

void **esp_efuse_reset**(void)

Reset efuse write registers.

Efuse write registers are written to zero, to negate any changes that have been staged here.

备注: This function is not threadsafe, if calling code updates efuse values from multiple tasks then this is caller's responsibility to serialise.

---

esp_err_t **esp_efuse_disable_rom_download_mode**(void)

Disable ROM Download Mode via eFuse.

Permanently disables the ROM Download Mode feature. Once disabled, if the SoC is booted with strapping pins set for ROM Download Mode then an error is printed instead.

备注: Not all SoCs support this option. An error will be returned if called on an ESP32 with a silicon revision lower than 3, as these revisions do not support this option.

备注: If ROM Download Mode is already disabled, this function does nothing and returns success.

返回

- ESP_OK If the eFuse was successfully burned, or had already been burned.
- ESP_ERR_NOT_SUPPORTED (ESP32 only) This SoC is not capable of disabling UART download mode
- ESP_ERR_INVALID_STATE (ESP32 only) This eFuse is write protected and cannot be written

esp_err_t **esp_efuse_set_rom_log_scheme**(*esp_efuse_rom_log_scheme_t* log_scheme)

Set boot ROM log scheme via eFuse.

备注: By default, the boot ROM will always print to console. This API can be called to set the log scheme only once per chip, once the value is changed from the default it can't be changed again.

参数 **log_scheme** -- Supported ROM log scheme

返回

- ESP_OK If the eFuse was successfully burned, or had already been burned.
- ESP_ERR_NOT_SUPPORTED (ESP32 only) This SoC is not capable of setting ROM log scheme
- ESP_ERR_INVALID_STATE This eFuse is write protected or has been burned already

esp_err_t **esp_efuse_enable_rom_secure_download_mode**(void)

Switch ROM Download Mode to Secure Download mode via eFuse.

Permanently enables Secure Download mode. This mode limits the use of ROM Download Mode functions to simple flash read, write and erase operations, plus a command to return a summary of currently enabled security features.

备注: If Secure Download mode is already enabled, this function does nothing and returns success.

备注： Disabling the ROM Download Mode also disables Secure Download Mode.

返回

- ESP_OK If the eFuse was successfully burned, or had already been burned.
- ESP_ERR_INVALID_STATE ROM Download Mode has been disabled via eFuse, so Secure Download mode is unavailable.

uint32_t **esp_efuse_read_secure_version**(void)

Return secure_version from efuse field.

返回 Secure version from efuse field

bool **esp_efuse_check_secure_version**(uint32_t secure_version)

Check secure_version from app and secure_version and from efuse field.

参数 **secure_version** -- Secure version from app.

返回

- True: If version of app is equal or more then secure_version from efuse.

esp_err_t **esp_efuse_update_secure_version**(uint32_t secure_version)

Write efuse field by secure_version value.

Update the secure_version value is available if the coding scheme is None. Note: Do not use this function in your applications. This function is called as part of the other API.

参数 **secure_version** -- **[in]** Secure version from app.

返回

- ESP_OK: Successful.
- ESP_FAIL: secure version of app cannot be set to efuse field.
- ESP_ERR_NOT_SUPPORTED: Anti rollback is not supported with the 3/4 and Repeat coding scheme.

esp_err_t **esp_efuse_batch_write_begin**(void)

Set the batch mode of writing fields.

This mode allows you to write the fields in the batch mode when need to burn several efuses at one time. To enable batch mode call begin() then perform as usually the necessary operations read and write and at the end call commit() to actually burn all written efuses. The batch mode can be used nested. The commit will be done by the last commit() function. The number of begin() functions should be equal to the number of commit() functions.

Note: If batch mode is enabled by the first task, at this time the second task cannot write/read efuses. The second task will wait for the first task to complete the batch operation.

```
// Example of using the batch writing mode.

// set the batch writing mode
esp_efuse_batch_write_begin();

// use any writing functions as usual
esp_efuse_write_field_blob(ESP_EFUSE_...);
esp_efuse_write_field_cnt(ESP_EFUSE_...);
esp_efuse_set_write_protect(EFUSE_BLKx);
esp_efuse_write_reg(EFUSE_BLKx, ...);
esp_efuse_write_block(EFUSE_BLKx, ...);
esp_efuse_write(ESP_EFUSE_1, 3);  // ESP_EFUSE_1 == 1, here we write a new␣
↪value = 3. The changes will be burn by the commit() function.
esp_efuse_read_...(ESP_EFUSE_1);  // this function returns ESP_EFUSE_1 == 1␣
↪because uncommitted changes are not readable, it will be available only␣
↪after commit.
```

(下页继续)

(续上页)

```
...

// esp_efuse_batch_write APIs can be called recursively.
esp_efuse_batch_write_begin();
esp_efuse_set_write_protect(EFUSE_BLKx);
esp_efuse_batch_write_commit(); // the burn will be skipped here, it will be␣
↪done in the last commit().

...

// Write all of these fields to the efuse registers
esp_efuse_batch_write_commit();
esp_efuse_read_...(ESP_EFUSE_1);  // this function returns ESP_EFUSE_1 == 3.
```

备注： Please note that reading in the batch mode does not show uncommitted changes.

返回

- ESP_OK: Successful.

esp_err_t **esp_efuse_batch_write_cancel**(void)

Reset the batch mode of writing fields.

It will reset the batch writing mode and any written changes.

返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_STATE: Tha batch mode was not set.

esp_err_t **esp_efuse_batch_write_commit**(void)

Writes all prepared data for the batch mode.

Must be called to ensure changes are written to the efuse registers. After this the batch writing mode will be reset.

返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_STATE: The deferred writing mode was not set.

bool **esp_efuse_block_is_empty**(*esp_efuse_block_t* block)

Checks that the given block is empty.

返回

- True: The block is empty.
- False: The block is not empty or was an error.

bool **esp_efuse_get_key_dis_read**(*esp_efuse_block_t* block)

Returns a read protection for the key block.

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回 True: The key block is read protected False: The key block is readable.

esp_err_t **esp_efuse_set_key_dis_read**(*esp_efuse_block_t* block)

Sets a read protection for the key block.

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.

- ESP_ERR_CODING: Error range of data does not match the coding scheme.

bool **esp_efuse_get_key_dis_write**(*esp_efuse_block_t* block)

Returns a write protection for the key block.

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回 True: The key block is write protected False: The key block is writeable.

esp_err_t **esp_efuse_set_key_dis_write**(*esp_efuse_block_t* block)

Sets a write protection for the key block.

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

bool **esp_efuse_key_block_unused**(*esp_efuse_block_t* block)

Returns true if the key block is unused, false otherwise.

An unused key block is all zero content, not read or write protected, and has purpose 0 (ESP_EFUSE_KEY_PURPOSE_USER)

参数 **block** -- key block to check.
返回

- True if key block is unused,
- False if key block is used or the specified block index is not a key block.

bool **esp_efuse_find_purpose**(*esp_efuse_purpose_t* purpose, *esp_efuse_block_t* *block)

Find a key block with the particular purpose set.

参数

- **purpose** -- **[in]** Purpose to search for.
- **block** -- **[out]** Pointer in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX which will be set to the key block if found. Can be NULL, if only need to test the key block exists.

返回

- True: If found,
- False: If not found (value at block pointer is unchanged).

bool **esp_efuse_get_keypurpose_dis_write**(*esp_efuse_block_t* block)

Returns a write protection of the key purpose field for an efuse key block.

备注: For ESP32: no keypurpose, it returns always True.

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回 True: The key purpose is write protected. False: The key purpose is writeable.

esp_efuse_purpose_t **esp_efuse_get_key_purpose**(*esp_efuse_block_t* block)

Returns the current purpose set for an efuse key block.

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回

- Value: If Successful, it returns the value of the purpose related to the given key block.
- ESP_EFUSE_KEY_PURPOSE_MAX: Otherwise.

const *esp_efuse_desc_t* ****esp_efuse_get_purpose_field**(*esp_efuse_block_t* block)

Returns a pointer to a key purpose for an efuse key block.

To get the value of this field use esp_efuse_read_field_blob() or esp_efuse_get_key_purpose().

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回 Pointer: If Successful returns a pointer to the corresponding efuse field otherwise NULL.

const *esp_efuse_desc_t* ****esp_efuse_get_key**(*esp_efuse_block_t* block)

Returns a pointer to a key block.

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回 Pointer: If Successful returns a pointer to the corresponding efuse field otherwise NULL.

esp_err_t **esp_efuse_set_key_purpose**(*esp_efuse_block_t* block, *esp_efuse_purpose_t* purpose)

Sets a key purpose for an efuse key block.

参数

- **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
- **purpose** -- **[in]** Key purpose.

返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

esp_err_t **esp_efuse_set_keypurpose_dis_write**(*esp_efuse_block_t* block)

Sets a write protection of the key purpose field for an efuse key block.

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

esp_efuse_block_t **esp_efuse_find_unused_key_block**(void)

Search for an unused key block and return the first one found.

See esp_efuse_key_block_unused for a description of an unused key block.

返回 First unused key block, or EFUSE_BLK_KEY_MAX if no unused key block is found.

unsigned **esp_efuse_count_unused_key_blocks**(void)

Return the number of unused efuse key blocks in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX.

bool **esp_efuse_get_digest_revoke**(unsigned num_digest)

Returns the status of the Secure Boot public key digest revocation bit.

参数 **num_digest** -- **[in]** The number of digest in range 0..2
返回

- True: If key digest is revoked,
- False; If key digest is not revoked.

esp_err_t **esp_efuse_set_digest_revoke**(unsigned num_digest)

Sets the Secure Boot public key digest revocation bit.

参数 **num_digest** -- **[in]** The number of digest in range 0..2
返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

bool **esp_efuse_get_write_protect_of_digest_revoke**(unsigned num_digest)

Returns a write protection of the Secure Boot public key digest revocation bit.

参数 **num_digest** -- **[in]** The number of digest in range 0..2

返回 True: The revocation bit is write protected. False: The revocation bit is writeable.

esp_err_t **esp_efuse_set_write_protect_of_digest_revoke**(unsigned num_digest)

Sets a write protection of the Secure Boot public key digest revocation bit.

参数 **num_digest** -- **[in]** The number of digest in range 0..2

返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

esp_err_t **esp_efuse_write_key**(*esp_efuse_block_t* block, *esp_efuse_purpose_t* purpose, const void *key, size_t key_size_bytes)

Program a block of key data to an efuse block.

The burn of a key, protection bits, and a purpose happens in batch mode.

备注: This API also enables the read protection efuse bit for certain key blocks like XTS-AES, HMAC, ECDSA etc. This ensures that the key is only accessible to hardware peripheral.

备注: For SoC's with capability `SOC_EFUSE_ECDSA_USE_HARDWARE_K` (e.g., ESP32-H2), this API writes an additional efuse bit for ECDSA key purpose to enforce hardware TRNG generated k mode in the peripheral.

参数

- **block** -- **[in]** Block to read purpose for. Must be in range EFUSE_BLK_KEY0 to EFUSE_BLK_KEY_MAX. Key block must be unused (esp_efuse_key_block_unused).
- **purpose** -- **[in]** Purpose to set for this key. Purpose must be already unset.
- **key** -- **[in]** Pointer to data to write.
- **key_size_bytes** -- **[in]** Bytes length of data to write.

返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_INVALID_STATE: Error in efuses state, unused block not found.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

esp_err_t **esp_efuse_write_keys**(const *esp_efuse_purpose_t* purposes[], uint8_t keys[][32], unsigned number_of_keys)

Program keys to unused efuse blocks.

The burn of keys, protection bits, and purposes happens in batch mode.

备注: This API also enables the read protection efuse bit for certain key blocks like XTS-AES, HMAC, ECDSA etc. This ensures that the key is only accessible to hardware peripheral.

备注: For SoC's with capability `SOC_EFUSE_ECDSA_USE_HARDWARE_K` (e.g., ESP32-H2), this API writes an additional efuse bit for ECDSA key purpose to enforce hardware TRNG generated k mode in the

peripheral.

参数

- **purposes** -- **[in]** Array of purposes (purpose[number_of_keys]).
- **keys** -- **[in]** Array of keys (uint8_t keys[number_of_keys][32]). Each key is 32 bytes long.
- **number_of_keys** -- **[in]** The number of keys to write (up to 6 keys).

返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_INVALID_STATE: Error in efuses state, unused block not found.
- ESP_ERR_NOT_ENOUGH_UNUSED_KEY_BLOCKS: Error not enough unused key blocks available
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

esp_err_t **esp_secure_boot_read_key_digests**(*esp_secure_boot_key_digests_t* *trusted_key_digests)

Read key digests from efuse. Any revoked/missing digests will be marked as NULL.

参数 **trusted_key_digests** -- **[out]** Trusted keys digests, stored in this parameter after successfully completing this function. The number of digests depends on the SOC's capabilities.

返回

- ESP_OK: Successful.
- ESP_FAIL: If trusted_keys is NULL or there is no valid digest.

esp_err_t **esp_efuse_check_errors**(void)

Checks eFuse errors in BLOCK0.

It does a BLOCK0 check if eFuse EFUSE_ERR_RST_ENABLE is set. If BLOCK0 has an error, it prints the error and returns ESP_FAIL, which should be treated as esp_restart.

备注： Refers to ESP32-C3 only.

返回

- ESP_OK: No errors in BLOCK0.
- ESP_FAIL: Error in BLOCK0 requiring reboot.

esp_err_t **esp_efuse_destroy_block**(*esp_efuse_block_t* block)

Destroys the data in the given efuse block, if possible.

Data destruction occurs through the following steps: 1) Destroy data in the block:

- If write protection is inactive for the block, then unset bits are burned.
- If write protection is active, the block remains unaltered. 2) Set read protection for the block if possible (check write-protection for RD_DIS). In this case, data becomes inaccessible, and the software reads it as all zeros. If write protection is enabled and read protection can not be set, data in the block remains readable (returns an error).

Do not use the batch mode with this function as it does the burning itself!

参数 **block** -- **[in]** A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX

返回

- ESP_OK: Successful.
- ESP_FAIL: Data remained readable because the block is write-protected and read protection can not be set.

Structures

struct **esp_efuse_desc_t**

Type definition for an eFuse field.

Public Members

esp_efuse_block_t **efuse_block**

Block of eFuse

uint8_t **bit_start**

Start bit [0..255]

uint16_t **bit_count**

Length of bit field [1..-]

struct **esp_secure_boot_key_digests_t**

Pointers to the trusted key digests.

The number of digests depends on the SOC's capabilities.

Public Members

const void ***key_digests**[3]

Pointers to the key digests

Macros

ESP_ERR_EFUSE

Base error code for efuse api.

ESP_OK_EFUSE_CNT

OK the required number of bits is set.

ESP_ERR_EFUSE_CNT_IS_FULL

Error field is full.

ESP_ERR_EFUSE_REPEATED_PROG

Error repeated programming of programmed bits is strictly forbidden.

ESP_ERR_CODING

Error while a encoding operation.

ESP_ERR_NOT_ENOUGH_UNUSED_KEY_BLOCKS

Error not enough unused key blocks available

ESP_ERR_DAMAGED_READING

Error. Burn or reset was done during a reading operation leads to damage read data. This error is internal to the efuse component and not returned by any public API.

Enumerations

enum **esp_efuse_rom_log_scheme_t**

Type definition for ROM log scheme.

Values:

enumerator **ESP_EFUSE_ROM_LOG_ALWAYS_ON**

Always enable ROM logging

enumerator **ESP_EFUSE_ROM_LOG_ON_GPIO_LOW**

ROM logging is enabled when specific GPIO level is low during start up

enumerator **ESP_EFUSE_ROM_LOG_ON_GPIO_HIGH**

ROM logging is enabled when specific GPIO level is high during start up

enumerator **ESP_EFUSE_ROM_LOG_ALWAYS_OFF**

Disable ROM logging permanently

2.9.8 错误代码和辅助函数

本节列出了 ESP-IDF 中常见错误代码的定义，以及部分与错误处理相关的辅助函数。

有关 ESP-IDF 中错误代码的基本信息，请参阅错误处理。

有关 ESP-IDF 定义的错误代码的完整列表，请参阅错误代码参考。

API 参考

Header File

- components/esp_common/include/esp_check.h
- This header file can be included with:

```
#include "esp_check.h"
```

Macros

ESP_RETURN_ON_ERROR(x, log_tag, format, ...)

Macro which can be used to check the error code. If the code is not ESP_OK, it prints the message and returns. In the future, we want to switch to C++20. We also want to become compatible with clang. Hence, we provide two versions of the following macros. The first one is using the GNU extension ##__VA_ARGS__. The second one is using the C++20 feature **VA_OPT**(,). This allows users to compile their code with standard C++20 enabled instead of the GNU extension. Below C++20, we haven't found any good alternative to using ##__VA_ARGS__. Macro which can be used to check the error code. If the code is not ESP_OK, it prints the message and returns.

ESP_RETURN_ON_ERROR_ISR(x, log_tag, format, ...)

A version of ESP_RETURN_ON_ERROR() macro that can be called from ISR.

ESP_RETURN_VOID_ON_ERROR(x, log_tag, format, ...)

Macro which can be used to check the error code. If the code is not ESP_OK, it prints the message and returns. This macro is used when the function returns void.

ESP_RETURN_VOID_ON_ERROR_ISR(x, log_tag, format, ...)

A version of ESP_RETURN_VOID_ON_ERROR() macro that can be called from ISR.

ESP_GOTO_ON_ERROR(x, goto_tag, log_tag, format, ...)

Macro which can be used to check the error code. If the code is not ESP_OK, it prints the message, sets the local variable 'ret' to the code, and then exits by jumping to 'goto_tag'.

ESP_GOTO_ON_ERROR_ISR(x, goto_tag, log_tag, format, ...)

A version of ESP_GOTO_ON_ERROR() macro that can be called from ISR.

ESP_RETURN_ON_FALSE(a, err_code, log_tag, format, ...)

Macro which can be used to check the condition. If the condition is not 'true', it prints the message and returns with the supplied 'err_code'.

ESP_RETURN_ON_FALSE_ISR(a, err_code, log_tag, format, ...)

A version of ESP_RETURN_ON_FALSE() macro that can be called from ISR.

ESP_RETURN_VOID_ON_FALSE(a, log_tag, format, ...)

Macro which can be used to check the condition. If the condition is not 'true', it prints the message and returns without a value.

ESP_RETURN_VOID_ON_FALSE_ISR(a, log_tag, format, ...)

A version of ESP_RETURN_VOID_ON_FALSE() macro that can be called from ISR.

ESP_GOTO_ON_FALSE(a, err_code, goto_tag, log_tag, format, ...)

Macro which can be used to check the condition. If the condition is not 'true', it prints the message, sets the local variable 'ret' to the supplied 'err_code', and then exits by jumping to 'goto_tag'.

ESP_GOTO_ON_FALSE_ISR(a, err_code, goto_tag, log_tag, format, ...)

A version of ESP_GOTO_ON_FALSE() macro that can be called from ISR.

Header File

- components/esp_common/include/esp_err.h
- This header file can be included with:

```
#include "esp_err.h"
```

Functions

const char ***esp_err_to_name**(*esp_err_t* code)

Returns string for esp_err_t error codes.

This function finds the error code in a pre-generated lookup-table and returns its string representation.

The function is generated by the Python script tools/gen_esp_err_to_name.py which should be run each time an esp_err_t error is modified, created or removed from the IDF project.

参数 **code** -- esp_err_t error code
返回 string error message

const char ***esp_err_to_name_r**(*esp_err_t* code, char *buf, size_t buflen)

Returns string for esp_err_t and system error codes.

This function finds the error code in a pre-generated lookup-table of esp_err_t errors and returns its string representation. If the error code is not found then it is attempted to be found among system errors.

The function is generated by the Python script tools/gen_esp_err_to_name.py which should be run each time an esp_err_t error is modified, created or removed from the IDF project.

参数

- **code** -- esp_err_t error code
- **buf** -- **[out]** buffer where the error message should be written
- **buflen** -- Size of buffer buf. At most buflen bytes are written into the buf buffer (including the terminating null byte).

返回 buf containing the string error message

Macros

ESP_OK

esp_err_t value indicating success (no error)

ESP_FAIL

Generic esp_err_t code indicating failure

ESP_ERR_NO_MEM

Out of memory

ESP_ERR_INVALID_ARG

Invalid argument

ESP_ERR_INVALID_STATE

Invalid state

ESP_ERR_INVALID_SIZE

Invalid size

ESP_ERR_NOT_FOUND

Requested resource not found

ESP_ERR_NOT_SUPPORTED

Operation or feature not supported

ESP_ERR_TIMEOUT

Operation timed out

ESP_ERR_INVALID_RESPONSE

Received response was invalid

ESP_ERR_INVALID_CRC

CRC or checksum was invalid

ESP_ERR_INVALID_VERSION

Version was invalid

ESP_ERR_INVALID_MAC

MAC address was invalid

ESP_ERR_NOT_FINISHED

Operation has not fully completed

ESP_ERR_NOT_ALLOWED

Operation is not allowed

ESP_ERR_WIFI_BASE

Starting number of WiFi error codes

ESP_ERR_MESH_BASE

Starting number of MESH error codes

ESP_ERR_FLASH_BASE

Starting number of flash error codes

ESP_ERR_HW_CRYPTO_BASE

Starting number of HW cryptography module error codes

ESP_ERR_MEMPROT_BASE

Starting number of Memory Protection API error codes

ESP_ERROR_CHECK(x)

Macro which can be used to check the error code, and terminate the program in case the code is not ESP_OK. Prints the error code, error location, and the failed statement to serial output.

Disabled if assertions are disabled.

ESP_ERROR_CHECK_WITHOUT_ABORT(x)

Macro which can be used to check the error code. Prints the error code, error location, and the failed statement to serial output. In comparison with ESP_ERROR_CHECK(), this prints the same error message but isn't terminating the program.

Type Definitions

typedef int **esp_err_t**

2.9.9 ESP HTTPS OTA 升级

概述

esp_https_ota 是现有 OTA（空中升级）API 的抽象层，其中提供了简化的 API，能够通过 HTTPS 升级固件。

应用示例

```
esp_err_t do_firmware_upgrade()
{
    esp_http_client_config_t config = {
        .url = CONFIG_FIRMWARE_UPGRADE_URL,
        .cert_pem = (char *)server_cert_pem_start,
    };
    esp_https_ota_config_t ota_config = {
        .http_config = &config,
    };
    esp_err_t ret = esp_https_ota(&ota_config);
    if (ret == ESP_OK) {
        esp_restart();
    } else {
        return ESP_FAIL;
    }
    return ESP_OK;
}
```

服务器验证

验证服务器时，应将 PEM 格式的根证书提供给*esp_http_client_config_t::cert_pem* 成员。如需了解有关服务器验证的更多信息，请参阅*TLS 服务器验证*。

备 注: 应使用服务器端点的 **根** 证书应用于验证，而不能使用证书链中的任何中间证书，因为根证书有效期最长，且通常长时间维持不变。用户还可以通过*esp_http_client_config_t::crt_bundle_attach* 成员使用 ESP x509 证书包功能进行验证，其中涵盖了大多数受信任的根证书。

通过 HTTPS 下载部分镜像

要使用部分镜像下载功能，请启用 esp_https_ota_config_t 中的 partial_http_download 配置。启用此配置后，固件镜像将通过多个指定大小的 HTTP 请求进行下载。将 max_http_request_size 设置为所需值，即可指定每个请求的最大内容长度。

在从 AWS S3 等服务获取镜像时，这一选项非常有用。在启用该选项时，可以将 mbedTLS Rx 的 buffer 大小（即*CONFIG_MBEDTLS_SSL_IN_CONTENT_LEN*）设置为较小的值。不启用此配置时，无法将其设置为较小值。

mbedTLS Rx buffer 的默认大小为 16 KB，但如果将 partial_http_download 的 max_http_request_size 设置为 4 KB，便能将 mbedTLS Rx 的 buffer 减小到 4 KB。使用这一配置方式预计可以节省约 12 KB 内存。

签名验证

要进一步提升安全性，还可以验证 OTA 固件镜像的签名。更多内容请参考*没有安全启动的安全 OTA 升级*。

高级 API

esp_https_ota 还提供一些高级 API，用于查看 OTA 过程的更多信息并满足其他控制需求。

如需查看使用高级 ESP_HTTPS_OTA API 的示例，请前往 system/ota/advanced_https_ota。

使用预加密固件进行 OTA 升级

如需使用预加密的固件进行 OTA 升级，请在组件的菜单配置中启用*CONFIG_ESP_HTTPS_OTA_DECRYPT_CB* 选项。

如需查看使用预加密固件进行 OTA 升级的示例，请前往 system/ota/pre_encrypted_ota。

OTA 系统事件

ESP HTTPS OTA 过程中可能发生各种系统事件。当特定事件发生时，会由事件循环库 触发处理程序。此处理程序必须使用*esp_event_handler_register()* 注册。这有助于 ESP HTTPS OTA 进行事件处理。

esp_https_ota_event_t 中包含了使用 ESP HTTPS OTA 升级时可能发生的所有事件。

事件处理程序示例

```
/* 用于捕获系统事件的事件处理程序 */
static void event_handler(void* arg, esp_event_base_t event_base,
                          int32_t event_id, void* event_data)
{
    if (event_base == ESP_HTTPS_OTA_EVENT) {
        switch (event_id) {
            case ESP_HTTPS_OTA_START:
                ESP_LOGI(TAG, "OTA started");
                break;
            case ESP_HTTPS_OTA_CONNECTED:
                ESP_LOGI(TAG, "Connected to server");
                break;
            case ESP_HTTPS_OTA_GET_IMG_DESC:
                ESP_LOGI(TAG, "Reading Image Description");
                break;
            case ESP_HTTPS_OTA_VERIFY_CHIP_ID:
                ESP_LOGI(TAG, "Verifying chip id of new image: %d", *(esp_
↪chip_id_t *)event_data);
                break;
            case ESP_HTTPS_OTA_DECRYPT_CB:
                ESP_LOGI(TAG, "Callback to decrypt function");
                break;
            case ESP_HTTPS_OTA_WRITE_FLASH:
                ESP_LOGD(TAG, "Writing to flash: %d written", *(int␣
↪*)event_data);
                break;
            case ESP_HTTPS_OTA_UPDATE_BOOT_PARTITION:
                ESP_LOGI(TAG, "Boot partition updated. Next Partition: %d
↪", *(esp_partition_subtype_t *)event_data);
                break;
            case ESP_HTTPS_OTA_FINISH:
                ESP_LOGI(TAG, "OTA finish");
                break;
            case ESP_HTTPS_OTA_ABORT:
                ESP_LOGI(TAG, "OTA abort");
                break;
        }
    }
}
```

系统事件循环中，不同 ESP HTTPS OTA 事件的预期数据类型如下所示：

- ESP_HTTPS_OTA_START : `NULL`
- ESP_HTTPS_OTA_CONNECTED : `NULL`
- ESP_HTTPS_OTA_GET_IMG_DESC : `NULL`
- ESP_HTTPS_OTA_VERIFY_CHIP_ID : `esp_chip_id_t`
- ESP_HTTPS_OTA_DECRYPT_CB : `NULL`
- ESP_HTTPS_OTA_WRITE_FLASH : `int`
- ESP_HTTPS_OTA_UPDATE_BOOT_PARTITION : `esp_partition_subtype_t`
- ESP_HTTPS_OTA_FINISH : `NULL`
- ESP_HTTPS_OTA_ABORT : `NULL`

API 参考

Header File

- components/esp_https_ota/include/esp_https_ota.h
- This header file can be included with:

```
#include "esp_https_ota.h"
```

- This header file is a part of the API provided by the `esp_https_ota` component. To declare that your component depends on `esp_https_ota`, add the following to your CMakeLists.txt:

```
REQUIRES esp_https_ota
```

or

```
PRIV_REQUIRES esp_https_ota
```

Functions

esp_err_t **esp_https_ota**(const *esp_https_ota_config_t* *ota_config)

HTTPS OTA Firmware upgrade.

This function allocates HTTPS OTA Firmware upgrade context, establishes HTTPS connection, reads image data from HTTP stream and writes it to OTA partition and finishes HTTPS OTA Firmware upgrade operation. This API supports URL redirection, but if CA cert of URLs differ then it should be appended to `cert_pem` member of `ota_config->http_config`.

备注: This API handles the entire OTA operation, so if this API is being used then no other APIs from `esp_https_ota` component should be called. If more information and control is needed during the HTTPS OTA process, then one can use `esp_https_ota_begin` and subsequent APIs. If this API returns successfully, esp_restart() must be called to boot from the new firmware image.

参数 **ota_config** -- **[in]** pointer to *esp_https_ota_config_t* structure.

返回

- ESP_OK: OTA data updated, next reboot will use specified partition.
- ESP_FAIL: For generic failure.
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- For other return codes, refer OTA documentation in esp-idf's app_update component.

esp_err_t **esp_https_ota_begin**(const *esp_https_ota_config_t* *ota_config, *esp_https_ota_handle_t* *handle)

Start HTTPS OTA Firmware upgrade.

This function initializes ESP HTTPS OTA context and establishes HTTPS connection. This function must be invoked first. If this function returns successfully, then `esp_https_ota_perform` should be called to continue with the OTA process and there should be a call to `esp_https_ota_finish` on completion of OTA operation or on failure in subsequent operations. This API supports URL redirection, but if CA cert of URLs differ then it should be appended to `cert_pem` member of `http_config`, which is a part of `ota_config`. In case of error, this API explicitly sets `handle` to NULL.

备注: This API is blocking, so setting `is_async` member of `http_config` structure will result in an error.

参数

- **ota_config** -- **[in]** pointer to *esp_https_ota_config_t* structure
- **handle** -- **[out]** pointer to an allocated data of type `esp_https_ota_handle_t` which will be initialised in this function

返回

- ESP_OK: HTTPS OTA Firmware upgrade context initialised and HTTPS connection established
- ESP_FAIL: For generic failure.
- ESP_ERR_INVALID_ARG: Invalid argument (missing/incorrect config, certificate, etc.)

- For other return codes, refer documentation in app_update component and esp_http_client component in esp-idf.

esp_err_t **esp_https_ota_perform** (*esp_https_ota_handle_t* https_ota_handle)

Read image data from HTTP stream and write it to OTA partition.

This function reads image data from HTTP stream and writes it to OTA partition. This function must be called only if esp_https_ota_begin() returns successfully. This function must be called in a loop since it returns after every HTTP read operation thus giving you the flexibility to stop OTA operation midway.

参数 **https_ota_handle** -- **[in]** pointer to esp_https_ota_handle_t structure

返回

- ESP_ERR_HTTPS_OTA_IN_PROGRESS: OTA update is in progress, call this API again to continue.
- ESP_OK: OTA update was successful
- ESP_FAIL: OTA update failed
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_VERSION: Invalid chip revision in image header
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- For other return codes, refer OTA documentation in esp-idf's app_update component.

bool **esp_https_ota_is_complete_data_received** (*esp_https_ota_handle_t* https_ota_handle)

Checks if complete data was received or not.

备注: This API can be called just before esp_https_ota_finish() to validate if the complete image was indeed received.

参数 **https_ota_handle** -- **[in]** pointer to esp_https_ota_handle_t structure

返回

- false
- true

esp_err_t **esp_https_ota_finish** (*esp_https_ota_handle_t* https_ota_handle)

Clean-up HTTPS OTA Firmware upgrade and close HTTPS connection.

This function closes the HTTP connection and frees the ESP HTTPS OTA context. This function switches the boot partition to the OTA partition containing the new firmware image.

备注: If this API returns successfully, esp_restart() must be called to boot from the new firmware image esp_https_ota_finish should not be called after calling esp_https_ota_abort

---

参数 **https_ota_handle** -- **[in]** pointer to esp_https_ota_handle_t structure

返回

- ESP_OK: Clean-up successful
- ESP_ERR_INVALID_STATE
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image

esp_err_t **esp_https_ota_abort** (*esp_https_ota_handle_t* https_ota_handle)

Clean-up HTTPS OTA Firmware upgrade and close HTTPS connection.

This function closes the HTTP connection and frees the ESP HTTPS OTA context.

备注： esp_https_ota_abort should not be called after calling esp_https_ota_finish

参数 **https_ota_handle** -- **[in]** pointer to esp_https_ota_handle_t structure

返回

- ESP_OK: Clean-up successful
- ESP_ERR_INVALID_STATE: Invalid ESP HTTPS OTA state
- ESP_FAIL: OTA not started
- ESP_ERR_NOT_FOUND: OTA handle not found
- ESP_ERR_INVALID_ARG: Invalid argument

esp_err_t **esp_https_ota_get_img_desc**(*esp_https_ota_handle_t* https_ota_handle, *esp_app_desc_t* *new_app_info)

Reads app description from image header. The app description provides information like the "Firmware version" of the image.

备注： This API can be called only after esp_https_ota_begin() and before esp_https_ota_perform(). Calling this API is not mandatory.

参数

- **https_ota_handle** -- **[in]** pointer to esp_https_ota_handle_t structure
- **new_app_info** -- **[out]** pointer to an allocated *esp_app_desc_t* structure

返回

- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_INVALID_STATE: Invalid state to call this API. esp_https_ota_begin() not called yet.
- ESP_FAIL: Failed to read image descriptor
- ESP_OK: Successfully read image descriptor

int **esp_https_ota_get_image_len_read**(*esp_https_ota_handle_t* https_ota_handle)

This function returns OTA image data read so far.

备注： This API should be called only if `esp_https_ota_perform()` has been called at least once or if `esp_https_ota_get_img_desc` has been called before.

参数 **https_ota_handle** -- **[in]** pointer to esp_https_ota_handle_t structure

返回

- -1 On failure
- total bytes read so far

int **esp_https_ota_get_status_code**(*esp_https_ota_handle_t* https_ota_handle)

This function returns the HTTP status code of the last HTTP response.

备注： This API should be called only after esp_https_ota_begin() has been called. This can be used to check the HTTP status code of the OTA download process.

参数 **https_ota_handle** -- **[in]** pointer to esp_https_ota_handle_t structure

返回

- -1 On failure
- HTTP status code

int **esp_https_ota_get_image_size**(*esp_https_ota_handle_t* https_ota_handle)

This function returns OTA image total size.

备注: This API should be called after esp_https_ota_begin() has been already called. This can be used to create some sort of progress indication (in combination with esp_https_ota_get_image_len_read())

参数 **https_ota_handle** -- **[in]** pointer to esp_https_ota_handle_t structure

返回

- -1 On failure or chunked encoding
- total bytes of image

Structures

struct **esp_https_ota_config_t**

ESP HTTPS OTA configuration.

Public Members

const *esp_http_client_config_t* ***http_config**

ESP HTTP client configuration

http_client_init_cb_t **http_client_init_cb**

Callback after ESP HTTP client is initialised

bool **bulk_flash_erase**

Erase entire flash partition during initialization. By default flash partition is erased during write operation and in chunk of 4K sector size

bool **partial_http_download**

Enable Firmware image to be downloaded over multiple HTTP requests

int **max_http_request_size**

Maximum request size for partial HTTP download

uint32_t **buffer_caps**

The memory capability to use when allocating the buffer for OTA update. Default capability is MALLOC_CAP_DEFAULT

Macros

ESP_ERR_HTTPS_OTA_BASE

ESP_ERR_HTTPS_OTA_IN_PROGRESS

Type Definitions

typedef void ***esp_https_ota_handle_t**

typedef *esp_err_t* (***http_client_init_cb_t**)(*esp_http_client_handle_t*)

Enumerations

enum **esp_https_ota_event_t**

Events generated by OTA process.

Values:

enumerator **ESP_HTTPS_OTA_START**

OTA started

enumerator **ESP_HTTPS_OTA_CONNECTED**

Connected to server

enumerator **ESP_HTTPS_OTA_GET_IMG_DESC**

Read app description from image header

enumerator **ESP_HTTPS_OTA_VERIFY_CHIP_ID**

Verify chip id of new image

enumerator **ESP_HTTPS_OTA_DECRYPT_CB**

Callback to decrypt function

enumerator **ESP_HTTPS_OTA_WRITE_FLASH**

Flash write operation

enumerator **ESP_HTTPS_OTA_UPDATE_BOOT_PARTITION**

Boot partition update after successful ota update

enumerator **ESP_HTTPS_OTA_FINISH**

OTA finished

enumerator **ESP_HTTPS_OTA_ABORT**

OTA aborted

2.9.10 事件循环库

概述

事件循环库使组件能够声明事件，允许其他组件注册处理程序（即在事件发生时执行的代码片段）。此时，无需直接涉及应用程序，松散耦合组件也能够在其他组件状态变化时附加所需的行为。此外，通过将代码执行序列化，在指定的任务中运行事件循环库，可以简化事件处理程序，实现更高效的事件处理。

调用 esp_event API

使用事件循环库时应注意区分“事件”与“事件循环”。

事件表示重要的发生事件，如 Wi-Fi 成功连接到接入点。引用事件时应使用由两部分组成的标识符，详情请参阅事件定义与事件声明。事件循环是连接事件和事件处理程序之间的桥梁，事件源通过使用事件循环库提供的 API 将事件发布到事件循环中，注册到事件循环中的事件处理程序会响应特定类型的事件。

以下为事件循环库的使用流程：

1. 定义一个函数，并在事件发布到事件循环中时运行该函数。此函数被称为事件处理程序，应具有与*esp_event_handler_t* 同类型的签名。
2. 调用*esp_event_loop_create()* 创建事件循环，该函数输出类型为*esp_event_loop_handle_t* 的循环句柄，使用此 API 创建的事件循环称为用户事件循环。另有一种特殊事件循环，请参阅默认事件循环。
3. 调用*esp_event_handler_register_with()* 将事件处理程序注册到循环中。处理程序可以注册到多个循环中，请参阅注册处理程序注意事项。
4. 事件源调用*esp_event_post_to()* 将事件发布到事件循环中。
5. 调用*esp_event_handler_unregister_with()* ，组件可以在事件循环中取消注册事件处理程序。
6. 调用*esp_event_loop_delete()* 删除不再需要的事件循环。

上述流程代码如下：

```
// 1. 定义事件处理程序
void run_on_event(void* handler_arg, esp_event_base_t base, int32_t id, void*␣
↪event_data)
{
    // 事件处理程序逻辑
}

void app_main()
{
    // 2. 用一个类型为 esp_event_loop_args_t␣
↪的配置结构体，指定所创建循环的属性。获取一个类型为 esp_event_loop_handle_t␣
↪的句柄，用于其他 API 引用循环、执行操作。
    esp_event_loop_args_t loop_args = {
        .queue_size = ...,
        .task_name = ...
        .task_priority = ...,
        .task_stack_size = ...,
        .task_core_id = ...
    };

    esp_event_loop_handle_t loop_handle;

    esp_event_loop_create(&loop_args, &loop_handle);

    // 3. 注册在 (1) 中定义的事件处理程序。MY_EVENT_BASE 和 MY_EVENT_ID␣
↪指定了一个假设事件：将处理程序 run_on_event 发布到循环中时，执行该处理程序。
    esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_
↪on_event, ...);

    ...

    // 4.␣
↪将事件发布到循环中。此时，事件排入事件循环队列，在某个时刻，事件循环会执行已注册到发布事件的事件
↪run_on_event。为简化过程，此示例从 app_main 调用 esp_event_post_
↪to，实际应用中可从任何其他任务中发布事件。
    esp_event_post_to(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, ...);

    ...

    // 5. 注销无用的处理程序。
    esp_event_handler_unregister_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_
↪on_event);

    ...

    // 6. 删除无用的事件循环。
    esp_event_loop_delete(loop_handle);
}
```

事件定义与事件声明

如前所述，事件标识符由两部分组成：事件根基和事件 ID。事件根基标识独立的事件组；事件 ID 标识组中的特定事件。可以将事件根基和事件 ID 类比为人的姓和名，姓表示一个家族，名表示家族中的某个人。

事件循环库提供了宏以便声明和定义事件根基。

声明事件根基：

```
ESP_EVENT_DECLARE_BASE(EVENT_BASE);
```

定义事件根基：

```
ESP_EVENT_DEFINE_BASE(EVENT_BASE);
```

备注： 在 ESP-IDF 中，系统事件的根基标识符为大写字母，并以 _EVENT 结尾。例如，Wi-Fi 事件的根基声明为 WIFI_EVENT，以太网的事件根基声明为 ETHERNET_EVENT 等。这样一来，事件根基与常量类似（尽管根据宏 ESP_EVENT_DECLARE_BASE 和 ESP_EVENT_DEFINE_BASE 的定义，它们属于全局变量）。

建议以枚举类型声明事件 ID，它们通常放在公共头文件中。

事件 ID：

```
enum {
    EVENT_ID_1,
    EVENT_ID_2,
    EVENT_ID_3,
    ...
}
```

默认事件循环

默认事件循环是一种特殊循环，用于处理系统事件（如 Wi-Fi 事件）。用户无法使用该循环的句柄，创建、删除、注册/注销处理程序以及事件发布均通过用户事件循环 API 的变体完成，下表列出了这些变体及其对应用户事件循环。

用户事件循环	默认事件循环
esp_event_loop_create()	*esp_event_loop_create_default()*
esp_event_loop_delete()	*esp_event_loop_delete_default()*
esp_event_handler_register_with()	*esp_event_handler_register()*
esp_event_handler_unregister_with()	*esp_event_handler_unregister()*
esp_event_post_to()	*esp_event_post()*

比较二者签名可知，它们大部分是相似的，唯一区别在于默认事件循环的 API 不需要指定循环句柄。

除了 API 的差异和用于系统事件的特殊分类外，默认事件循环和用户事件循环的行为并无差异。实际上，用户甚至可以将自己的事件发布到默认事件循环中，以节省内存而无需创建自己的循环。

注册处理程序注意事项

通过重复调用*esp_event_handler_register_with()*，可以将单个处理程序独立注册到多个事件中，且每次调用均可指定处理程序应执行的具体事件根基和事件 ID。

然而，在某些情况下，你可能希望处理程序在以下情况时执行：

(1) 所有发布到循环的事件

(2) 特定基本标识符的所有事件

为此，可调用特殊的事件根基标识符 ESP_EVENT_ANY_BASE 和特殊的事件 ESP_EVENT_ANY_ID 实现，这些特殊标识符可以作为*esp_event_handler_register_with()* 的事件根基和事件 ID 参数传递。

因此*esp_event_handler_register_with()* 的有效参数为：

1. <event base>, <event ID> - 根基为 <event base> 且 ID 为 <event ID> 的事件发布到循环中时，执行处理程序。
2. <event base>, ESP_EVENT_ANY_ID - 任何根基为 <event base> 的事件发布到循环中时，执行处理程序。
3. ESP_EVENT_ANY_BASE, ESP_EVENT_ANY_ID - 任何事件发布到循环中时，执行处理程序。

例如，如果注册了以下处理程序：

```
esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_on_
↪event_1, ...);
esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, ESP_EVENT_ANY_ID, run_
↪on_event_2, ...);
esp_event_handler_register_with(loop_handle, ESP_EVENT_ANY_BASE, ESP_EVENT_ANY_ID,␣
↪run_on_event_3, ...);
```

如果假设事件由 MY_EVENT_BASE 和 MY_EVENT_ID 组成，则三个处理程序 run_on_event_1、run_on_event_2 和 run_on_event_3 都会执行。

如果假设事件由 MY_EVENT_BASE 和 MY_OTHER_EVENT_ID 组成，则仅执行处理程序 run_on_event_2 和 run_on_event_3。

如果假设事件由 MY_OTHER_EVENT_BASE 和 MY_OTHER_EVENT_ID 组成，则仅执行处理程序 run_on_event_3。

处理程序自行注销 通常情况下，由事件循环运行的事件处理程序 **不允许在该事件循环上执行任何注册/注销活动**，但允许处理程序自行注销。例如，可以执行以下操作：

```
void run_on_event(void* handler_arg, esp_event_base_t base, int32_t id, void*␣
↪event_data)
{
    esp_event_loop_handle_t *loop_handle = (esp_event_loop_handle_t*) handler_arg;
    esp_event_handler_unregister_with(*loop_handle, MY_EVENT_BASE, MY_EVENT_ID,␣
↪run_on_event);
}

void app_main(void)
{
    esp_event_loop_handle_t loop_handle;
    esp_event_loop_create(&loop_args, &loop_handle);
    esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_
↪on_event, &loop_handle);
    // ... 发布事件 MY_EVENT_BASE 和 MY_EVENT_ID，并在某些时候运行循环
}
```

注册处理程序及处理程序调度顺序 一般而言，对于在调度期间与某个已发布事件匹配的处理程序，先注册的也会先执行。在所有注册均使用单个任务执行的情况下，可以通过在其他处理程序注册前注册目标处理程序，控制处理程序的执行顺序。如果计划利用这一规则，在有多个任务注册处理程序的情况下要多加小心。此时，虽然“先注册，先执行”的规则仍然成立，但率先执行的任务也会率先注册其处理程序，而由单个任务连续注册的处理函数仍然按相对顺序调度。但如果该任务在注册期间被另一个任务抢占，而该任务还注册了处理程序，则在调度期间，那些处理程序也将在处理其他任务时执行。

事件循环性能分析

要启动数据收集，统计所有已创建事件循环的数据，请激活配置选项*CONFIG_ESP_EVENT_LOOP_PROFILING*，函数`esp_event_dump()`可将收集的统计数据输出到文件流中。有关转储信息的更多详情，请参阅`esp_event_dump()` API 参考。

应用示例

使用 `esp_event` 库的示例存放在 system/esp_event 中，涵盖事件声明、循环创建、处理程序注册和注销以及事件发布。

其他使用 `esp_event` 库的示例:

- NMEA Parser，该示例将解码从 GPS 接收到的语句。

API 参考

Header File

- components/esp_event/include/esp_event.h
- This header file can be included with:

```
#include "esp_event.h"
```

- This header file is a part of the API provided by the `esp_event` component. To declare that your component depends on `esp_event`, add the following to your CMakeLists.txt:

```
REQUIRES esp_event
```

or

```
PRIV_REQUIRES esp_event
```

Functions

esp_err_t **esp_event_loop_create**(const *esp_event_loop_args_t* *event_loop_args, *esp_event_loop_handle_t* *event_loop)

Create a new event loop.

参数

- **event_loop_args** -- **[in]** configuration structure for the event loop to create
- **event_loop** -- **[out]** handle to the created event loop

返回

- ESP_OK: Success
- ESP_ERR_INVALID_ARG: event_loop_args or event_loop was NULL
- ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
- ESP_FAIL: Failed to create task loop
- Others: Fail

esp_err_t **esp_event_loop_delete**(*esp_event_loop_handle_t* event_loop)

Delete an existing event loop.

参数 **event_loop** -- **[in]** event loop to delete, must not be NULL

返回

- ESP_OK: Success
- Others: Fail

esp_err_t **esp_event_loop_create_default**(void)

Create default event loop.

返回

- ESP_OK: Success

- ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
- ESP_ERR_INVALID_STATE: Default event loop has already been created
- ESP_FAIL: Failed to create task loop
- Others: Fail

esp_err_t **esp_event_loop_delete_default**(void)

Delete the default event loop.

返回

- ESP_OK: Success
- Others: Fail

esp_err_t **esp_event_loop_run**(*esp_event_loop_handle_t* event_loop, TickType_t ticks_to_run)

Dispatch events posted to an event loop.

This function is used to dispatch events posted to a loop with no dedicated task, i.e. task name was set to NULL in event_loop_args argument during loop creation. This function includes an argument to limit the amount of time it runs, returning control to the caller when that time expires (or some time afterwards). There is no guarantee that a call to this function will exit at exactly the time of expiry. There is also no guarantee that events have been dispatched during the call, as the function might have spent all the allotted time waiting on the event queue. Once an event has been dequeued, however, it is guaranteed to be dispatched. This guarantee contributes to not being able to exit exactly at time of expiry as (1) blocking on internal mutexes is necessary for dispatching the dequeued event, and (2) during dispatch of the dequeued event there is no way to control the time occupied by handler code execution. The guaranteed time of exit is therefore the allotted time + amount of time required to dispatch the last dequeued event.

In cases where waiting on the queue times out, ESP_OK is returned and not ESP_ERR_TIMEOUT, since it is normal behavior.

备注： encountering an unknown event that has been posted to the loop will only generate a warning, not an error.

参数

- **event_loop** -- **[in]** event loop to dispatch posted events from, must not be NULL
- **ticks_to_run** -- **[in]** number of ticks to run the loop

返回

- ESP_OK: Success
- Others: Fail

esp_err_t **esp_event_handler_register**(esp_event_base_t event_base, int32_t event_id, *esp_event_handler_t* event_handler, void *event_handler_arg)

Register an event handler to the system event loop (legacy).

This function can be used to register a handler for either: (1) specific events, (2) all events of a certain event base, or (3) all events known by the system event loop.

- specific events: specify exact event_base and event_id
- all events of a certain base: specify exact event_base and use ESP_EVENT_ANY_ID as the event_id
- all events known by the loop: use ESP_EVENT_ANY_BASE for event_base and ESP_EVENT_ANY_ID as the event_id

Registering multiple handlers to events is possible. Registering a single handler to multiple events is also possible. However, registering the same handler to the same event multiple times would cause the previous registrations to be overwritten.

备注： the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

---

参数

- **event_base** -- **[in]** the base ID of the event to register the handler for
- **event_id** -- **[in]** the ID of the event to register the handler for
- **event_handler** -- **[in]** the handler function which gets called when the event is dispatched
- **event_handler_arg** -- **[in]** data, aside from event data, that is passed to the handler when it is called

返回

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

esp_err_t **esp_event_handler_register_with**(*esp_event_loop_handle_t* event_loop, esp_event_base_t event_base, int32_t event_id, *esp_event_handler_t* event_handler, void *event_handler_arg)

Register an event handler to a specific loop (legacy).

This function behaves in the same manner as esp_event_handler_register, except the additional specification of the event loop to register the handler to.

备注： the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

---

参数

- **event_loop** -- **[in]** the event loop to register this handler function to, must not be NULL
- **event_base** -- **[in]** the base ID of the event to register the handler for
- **event_id** -- **[in]** the ID of the event to register the handler for
- **event_handler** -- **[in]** the handler function which gets called when the event is dispatched
- **event_handler_arg** -- **[in]** data, aside from event data, that is passed to the handler when it is called

返回

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

esp_err_t **esp_event_handler_instance_register_with**(*esp_event_loop_handle_t* event_loop, esp_event_base_t event_base, int32_t event_id, *esp_event_handler_t* event_handler, void *event_handler_arg, *esp_event_handler_instance_t* *instance)

Register an instance of event handler to a specific loop.

This function can be used to register a handler for either: (1) specific events, (2) all events of a certain event base, or (3) all events known by the system event loop.

- specific events: specify exact event_base and event_id
- all events of a certain base: specify exact event_base and use ESP_EVENT_ANY_ID as the event_id
- all events known by the loop: use ESP_EVENT_ANY_BASE for event_base and ESP_EVENT_ANY_ID as the event_id

Besides the error, the function returns an instance object as output parameter to identify each registration. This is necessary to remove (unregister) the registration before the event loop is deleted.

Registering multiple handlers to events, registering a single handler to multiple events as well as registering the same handler to the same event multiple times is possible. Each registration yields a distinct instance object which identifies it over the registration lifetime.

备注： the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

---

备 注： Calling this function with instance set to NULL is equivalent to calling esp_event_handler_register_with.

参数

- **`event_loop`** -- **[in]** the event loop to register this handler function to, must not be NULL
- **`event_base`** -- **[in]** the base ID of the event to register the handler for
- **`event_id`** -- **[in]** the ID of the event to register the handler for
- **`event_handler`** -- **[in]** the handler function which gets called when the event is dispatched
- **`event_handler_arg`** -- **[in]** data, aside from event data, that is passed to the handler when it is called
- **`instance`** -- **[out]** An event handler instance object related to the registered event handler and data, can be NULL. This needs to be kept if the specific callback instance should be unregistered before deleting the whole event loop. Registering the same event handler multiple times is possible and yields distinct instance objects. The data can be the same for all registrations. If no unregistration is needed, but the handler should be deleted when the event loop is deleted, instance can be NULL.

返回

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID or instance is NULL
- Others: Fail

esp_err_t **`esp_event_handler_instance_register`**(esp_event_base_t event_base, int32_t event_id, *esp_event_handler_t* event_handler, void *event_handler_arg, *esp_event_handler_instance_t* *instance)

Register an instance of event handler to the default loop.

This function does the same as esp_event_handler_instance_register_with, except that it registers the handler to the default event loop.

备注： the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

---

备注： Calling this function with instance set to NULL is equivalent to calling esp_event_handler_register.

参数

- **`event_base`** -- **[in]** the base ID of the event to register the handler for
- **`event_id`** -- **[in]** the ID of the event to register the handler for
- **`event_handler`** -- **[in]** the handler function which gets called when the event is dispatched
- **`event_handler_arg`** -- **[in]** data, aside from event data, that is passed to the handler when it is called

- **instance** -- **[out]** An event handler instance object related to the registered event handler and data, can be NULL. This needs to be kept if the specific callback instance should be unregistered before deleting the whole event loop. Registering the same event handler multiple times is possible and yields distinct instance objects. The data can be the same for all registrations. If no unregistration is needed, but the handler should be deleted when the event loop is deleted, instance can be NULL.

返回

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID or instance is NULL
- Others: Fail

esp_err_t **esp_event_handler_unregister**(esp_event_base_t event_base, int32_t event_id, *esp_event_handler_t* event_handler)

Unregister a handler with the system event loop (legacy).

Unregisters a handler, so it will no longer be called during dispatch. Handlers can be unregistered for any combination of event_base and event_id which were previously registered. To unregister a handler, the event_base and event_id arguments must match exactly the arguments passed to esp_event_handler_register() when that handler was registered. Passing ESP_EVENT_ANY_BASE and/or ESP_EVENT_ANY_ID will only unregister handlers that were registered with the same wildcard arguments.

备注: When using ESP_EVENT_ANY_ID, handlers registered to specific event IDs using the same base will not be unregistered. When using ESP_EVENT_ANY_BASE, events registered to specific bases will also not be unregistered. This avoids accidental unregistration of handlers registered by other users or components.

参数

- **event_base** -- **[in]** the base of the event with which to unregister the handler
- **event_id** -- **[in]** the ID of the event with which to unregister the handler
- **event_handler** -- **[in]** the handler to unregister

返回 ESP_OK success

返回 ESP_ERR_INVALID_ARG invalid combination of event base and event ID

返回 others fail

esp_err_t **esp_event_handler_unregister_with**(*esp_event_loop_handle_t* event_loop, esp_event_base_t event_base, int32_t event_id, *esp_event_handler_t* event_handler)

Unregister a handler from a specific event loop (legacy).

This function behaves in the same manner as esp_event_handler_unregister, except the additional specification of the event loop to unregister the handler with.

参数

- **event_loop** -- **[in]** the event loop with which to unregister this handler function, must not be NULL
- **event_base** -- **[in]** the base of the event with which to unregister the handler
- **event_id** -- **[in]** the ID of the event with which to unregister the handler
- **event_handler** -- **[in]** the handler to unregister

返回

- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

esp_err_t **esp_event_handler_instance_unregister_with**(*esp_event_loop_handle_t* event_loop, esp_event_base_t event_base, int32_t event_id, *esp_event_handler_instance_t* instance)

Unregister a handler instance from a specific event loop.

Unregisters a handler instance, so it will no longer be called during dispatch. Handler instances can be unregistered for any combination of event_base and event_id which were previously registered. To unregister a handler instance, the event_base and event_id arguments must match exactly the arguments passed to esp_event_handler_instance_register() when that handler instance was registered. Passing ESP_EVENT_ANY_BASE and/or ESP_EVENT_ANY_ID will only unregister handler instances that were registered with the same wildcard arguments.

备注： When using ESP_EVENT_ANY_ID, handlers registered to specific event IDs using the same base will not be unregistered. When using ESP_EVENT_ANY_BASE, events registered to specific bases will also not be unregistered. This avoids accidental unregistration of handlers registered by other users or components.

参数

- **`event_loop`** -- **[in]** the event loop with which to unregister this handler function, must not be NULL
- **`event_base`** -- **[in]** the base of the event with which to unregister the handler
- **`event_id`** -- **[in]** the ID of the event with which to unregister the handler
- **`instance`** -- **[in]** the instance object of the registration to be unregistered

返回

- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

esp_err_t **`esp_event_handler_instance_unregister`**(esp_event_base_t event_base, int32_t event_id, *esp_event_handler_instance_t* instance)

Unregister a handler from the system event loop.

This function does the same as esp_event_handler_instance_unregister_with, except that it unregisters the handler instance from the default event loop.

参数

- **`event_base`** -- **[in]** the base of the event with which to unregister the handler
- **`event_id`** -- **[in]** the ID of the event with which to unregister the handler
- **`instance`** -- **[in]** the instance object of the registration to be unregistered

返回

- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

esp_err_t **`esp_event_post`**(esp_event_base_t event_base, int32_t event_id, const void *event_data, size_t event_data_size, TickType_t ticks_to_wait)

Posts an event to the system default event loop. The event loop library keeps a copy of event_data and manages the copy's lifetime automatically (allocation + deletion); this ensures that the data the handler receives is always valid.

参数

- **`event_base`** -- **[in]** the event base that identifies the event
- **`event_id`** -- **[in]** the event ID that identifies the event
- **`event_data`** -- **[in]** the data, specific to the event occurrence, that gets passed to the handler
- **`event_data_size`** -- **[in]** the size of the event data
- **`ticks_to_wait`** -- **[in]** number of ticks to block on a full event queue

返回

- ESP_OK: Success
- ESP_ERR_TIMEOUT: Time to wait for event queue to unblock expired, queue full when posting from ISR

- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

esp_err_t **esp_event_post_to**(*esp_event_loop_handle_t* event_loop, esp_event_base_t event_base, int32_t event_id, const void *event_data, size_t event_data_size, TickType_t ticks_to_wait)

Posts an event to the specified event loop. The event loop library keeps a copy of event_data and manages the copy's lifetime automatically (allocation + deletion); this ensures that the data the handler receives is always valid.

This function behaves in the same manner as esp_event_post, except the additional specification of the event loop to post the event to.

参数

- **event_loop** -- **[in]** the event loop to post to, must not be NULL
- **event_base** -- **[in]** the event base that identifies the event
- **event_id** -- **[in]** the event ID that identifies the event
- **event_data** -- **[in]** the data, specific to the event occurrence, that gets passed to the handler
- **event_data_size** -- **[in]** the size of the event data
- **ticks_to_wait** -- **[in]** number of ticks to block on a full event queue

返回

- ESP_OK: Success
- ESP_ERR_TIMEOUT: Time to wait for event queue to unblock expired, queue full when posting from ISR
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

esp_err_t **esp_event_isr_post**(esp_event_base_t event_base, int32_t event_id, const void *event_data, size_t event_data_size, BaseType_t *task_unblocked)

Special variant of esp_event_post for posting events from interrupt handlers.

备注： this function is only available when CONFIG_ESP_EVENT_POST_FROM_ISR is enabled

---

备注： when this function is called from an interrupt handler placed in IRAM, this function should be placed in IRAM as well by enabling CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR

参数

- **event_base** -- **[in]** the event base that identifies the event
- **event_id** -- **[in]** the event ID that identifies the event
- **event_data** -- **[in]** the data, specific to the event occurrence, that gets passed to the handler
- **event_data_size** -- **[in]** the size of the event data; max is 4 bytes
- **task_unblocked** -- **[out]** an optional parameter (can be NULL) which indicates that an event task with higher priority than currently running task has been unblocked by the posted event; a context switch should be requested before the interrupt is existed.

返回

- ESP_OK: Success
- ESP_FAIL: Event queue for the default event loop full
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID, data size of more than 4 bytes
- Others: Fail

esp_err_t **esp_event_isr_post_to**(*esp_event_loop_handle_t* event_loop, esp_event_base_t event_base, int32_t event_id, const void *event_data, size_t event_data_size, BaseType_t *task_unblocked)

Special variant of esp_event_post_to for posting events from interrupt handlers.

备注： this function is only available when CONFIG_ESP_EVENT_POST_FROM_ISR is enabled

备注： when this function is called from an interrupt handler placed in IRAM, this function should be placed in IRAM as well by enabling CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR

参数

- **event_loop** -- **[in]** the event loop to post to, must not be NULL
- **event_base** -- **[in]** the event base that identifies the event
- **event_id** -- **[in]** the event ID that identifies the event
- **event_data** -- **[in]** the data, specific to the event occurrence, that gets passed to the handler
- **event_data_size** -- **[in]** the size of the event data
- **task_unblocked** -- **[out]** an optional parameter (can be NULL) which indicates that an event task with higher priority than currently running task has been unblocked by the posted event; a context switch should be requested before the interrupt is existed.

返回

- ESP_OK: Success
- ESP_FAIL: Event queue for the loop full
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID, data size of more than 4 bytes
- Others: Fail

esp_err_t **esp_event_dump**(FILE *file)

Dumps statistics of all event loops.

Dumps event loop info in the format:

```
      event loop
          handler
          handler
          ...
      event loop
          handler
          handler
          ...

 where:

  event loop
      format: address,name rx:total_received dr:total_dropped
      where:
          address - memory address of the event loop
          name - name of the event loop, 'none' if no dedicated task
          total_received - number of successfully posted events
          total_dropped - number of events unsuccessfully posted due to queue␣
↪being full

  handler
      format: address ev:base,id inv:total_invoked run:total_runtime
      where:
          address - address of the handler function
          base,id - the event specified by event base and ID this handler␣
↪executes
          total_invoked - number of times this handler has been invoked
```

(下页继续)

(续上页)

```
        total_runtime - total amount of time used for invoking this handler
```

备注: this function is a noop when CONFIG_ESP_EVENT_LOOP_PROFILING is disabled

参数 **file** -- **[in]** the file stream to output to

返回

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
- Others: Fail

Structures

struct **esp_event_loop_args_t**

Configuration for creating event loops.

Public Members

int32_t **queue_size**

size of the event loop queue

const char ***task_name**

name of the event loop task; if NULL, a dedicated task is not created for event loop

UBaseType_t **task_priority**

priority of the event loop task, ignored if task name is NULL

uint32_t **task_stack_size**

stack size of the event loop task, ignored if task name is NULL

BaseType_t **task_core_id**

core to which the event loop task is pinned to, ignored if task name is NULL

Header File

- components/esp_event/include/esp_event_base.h
- This header file can be included with:

```
#include "esp_event_base.h"
```

- This header file is a part of the API provided by the `esp_event` component. To declare that your component depends on `esp_event`, add the following to your CMakeLists.txt:

```
REQUIRES esp_event
```

or

```
PRIV_REQUIRES esp_event
```

Macros

ESP_EVENT_DECLARE_BASE(id)

ESP_EVENT_DEFINE_BASE(id)

ESP_EVENT_ANY_BASE

register handler for any event base

ESP_EVENT_ANY_ID

register handler for any event id

Type Definitions

typedef void ***esp_event_loop_handle_t**

a number that identifies an event with respect to a base

typedef void (***esp_event_handler_t**)(void *event_handler_arg, esp_event_base_t event_base, int32_t event_id, void *event_data)

function called when an event is posted to the queue

typedef void ***esp_event_handler_instance_t**

context identifying an instance of a registered event handler

相关文档

2.9.11 FreeRTOS 概述

概述

FreeRTOS 是一个开源的 RTOS（实时操作系统）内核，它以组件的形式集成到 ESP-IDF 中。因此，所有的 ESP-IDF 应用程序及多种 ESP-IDF 组件都基于 FreeRTOS 编写。FreeRTOS 内核已移植到 ESP 芯片的所有 CPU 架构（即 Xtensa 和 RISC-V）中。

此外，ESP-IDF 还提供了不同的 FreeRTOS 实现，支持在多核 ESP 目标芯片上的 SMP（对称多处理）。本文档主要介绍了 FreeRTOS 组件、ESP-IDF 提供的 FreeRTOS 实现，并简要介绍了这些实现的共同之处。

实现

官方 FreeRTOS（下文称为原生 FreeRTOS）是一个单核 RTOS。为了支持各种多核 ESP 目标芯片，ESP-IDF 支持下述不同的 FreeRTOS 实现。

ESP-IDF FreeRTOS

ESP-IDF FreeRTOS 是基于原生 FreeRTOS v10.5.1 的 FreeRTOS 实现，其中包含支持 SMP 的大量更新。ESP-IDF FreeRTOS 最多支持两个核（即双核 SMP），但在设计上对这种场景进行了优化。关于 ESP-IDF FreeRTOS 及具体更新内容，请参考*FreeRTOS (IDF)* 文档。

备注： ESP-IDF FreeRTOS 是目前 ESP-IDF 默认的 FreeRTOS 实现。

配置

内核配置 原生 FreeRTOS 要求移植工具和应用程序通过在 `FreeRTOSConfig.h` 头文件中添加各种 `#define config...` 宏定义来配置内核。原生 FreeRTOS 支持一系列内核配置选项，允许启用或禁用各种内核行为和功能。

然而，对于 ESP-IDF 中的所有 FreeRTOS 移植，FreeRTOSConfig.h 头文件被视为私有文件，用户不得修改。由于该选项在 ESP-IDF 中是必选项或不被支持，`FreeRTOSConfig.h` 中的大量内核配置选项均为硬编码。所有用户可配置的内核配置选项都在 `Component Config/FreeRTOS/Kernel` 下的 menuconfig 中。

关于用户可配置内核选项的完整列表，参见项目配置。下列为常用的内核配置选项：

- *CONFIG_FREERTOS_UNICORE*：仅在核 0 上运行 FreeRTOS。注意，这 **不等同于运行原生 FreeRTOS**。另外，此选项还可能影响除 freertos 外其他组件的行为。关于在单核上运行 FreeRTOS 的更多内容，请参考单核模式（使用 ESP-IDF FreeRTOS 时）或参考 Amazon SMP FreeRTOS 的官方文档，还可以在 ESP-IDF 组件中搜索 `CONFIG_FREERTOS_UNICORE`。
- *CONFIG_FREERTOS_ENABLE_BACKWARD_COMPATIBILITY* 可以向后兼容某些 FreeRTOS 宏、类型或函数，这些宏、类型或函数已在 v8.0 及以上版本中弃用。

端口配置 其他不属于内核配置的 FreeRTOS 相关配置选项都在 `Component Config/FreeRTOS/Port` 下的 menuconfig 中。这些选项可以配置以下内容：

- FreeRTOS 端口本身（如 tick 定时器选择，ISR 堆栈大小）
- 其他添加到 FreeRTOS 实现或端口的功能

使用 FreeRTOS

应用程序入口点 与原生 FreeRTOS 不同，在 ESP-IDF 中使用 FreeRTOS 的用户 **永远不应调用** `vTaskStartScheduler()` 和 `vTaskEndScheduler()`。相反，ESP-IDF 会自动启动 FreeRTOS。用户必须定义一个 `void app_main(void)` 函数作为用户应用程序的入口点，并在 ESP-IDF 启动时被自动调用。

- 通常，用户会从 `app_main` 中启动应用程序的其他任务。
- `app_main` 函数可以在任何时候返回（应用终止前）。
- `app_main` 函数由 `main` 任务调用。

后台任务 在启动过程中，ESP-IDF 和 FreeRTOS 内核会自动创建多个在后台运行的任务，如下表所示。

表 10: 启动过程创建任务列表

任务名称	描述	堆栈大小	亲和性	优先级
空闲任务 (IDLEx)	为每个 CPU 核创建并分配一个空闲任务 (IDLEx)，其中 x 是 CPU 核的编号。当启用单核配置时，x 将被删除。	*CONFIG_FREERTOS_IDLE_TASK_STACKSIZE*	核 x	0
FreeRTOS 定时器任务 (Tmr Svc)	如果应用程序调用了任何 FreeRTOS 定时器 API，FreeRTOS 会创建定时器服务或守护任务	*CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH*	核 0	*CONFIG_FREERTOS_TIMER_TASK_PRIORITY*
主任务 (main)	简单调用 app_main 的任务在 app_main 返回时会自我删除	*CONFIG_ESP_MAIN_TASK_STACK_SIZE*	*CONFIG_ESP_MAIN_TASK_AFFINITY*	1
IPC 任务 (ipcx)	当*CONFIG_FREERTOS_UNICORE* 为假时，为每个 CPU 核创建并分配一个 IPC 任务 (ipcx)。IPC 任务用于实现处理器间调用 (IPC) 功能	*CONFIG_ESP_IPC_TASK_STACK_SIZE*	核 x	24
ESP 定时器任务 (esp_timer)	ESP-IDF 创建 ESP 定时器任务用于处理 ESP 定时器回调	*CONFIG_ESP_TIMER_TASK_STACK_SIZE*	核 0	22

备注： 注意，如果应用程序使用了其他 ESP-IDF 功能（如 Wi-Fi 或蓝牙），那么这些功能可能会在上表的任务之外创建自己的后台任务。

FreeRTOS 附加功能

ESP-IDF 还为 FreeRTOS 提供了一些补充功能，如环形 buffer、ESP-IDF 风格的 tick 钩子和 idle 钩子、以及 TLSP 删除回调。要了解更多信息，请参见*FreeRTOS（附加功能）*。

FreeRTOS 堆

原生 FreeRTOS 自带 堆实现选择，然而 ESP-IDF 已经实现了自己的堆（参见堆内存分配），因此不使用原生 FreeRTOS 的堆实现。ESP-IDF 中的所有 FreeRTOS 端口都将 FreeRTOS 内存分配或释放调用（例如 pvPortMalloc() 和 pvPortFree()）映射到 ESP-IDF 堆 API（即*heap_caps_malloc()* 和*heap_caps_free()*）。然而 FreeRTOS 端口可以确保 FreeRTOS 分配的所有动态内存都放在内部内存中。

备注： 如果希望将 FreeRTOS 任务或对象放在外部内存中，可以使用以下方法：

- 使用一个 ...CreateWithCaps() API，如*xTaskCreateWithCaps()* 和*xQueueCreateWithCaps()* 来分配任务或对象（参见*IDF 附加 API* 获取更多详细信息）。
- 使用*heap_caps_malloc()* 为这些对象手动分配外部内存，然后使用 FreeRTOS 的一个 ...CreateStatic() 函数从分配的内存中创建对象。

2.9.12 FreeRTOS (IDF)

本文档介绍了 ESP-IDF 框架内的 FreeRTOS 双核 SMP 实现，包含以下小节：

目录

- *FreeRTOS (IDF)*
  - 概述
  - 对称多处理
  - 任务
  - *SMP* 调度器
  - 临界区
  - 其他事项
  - *API* 参考

概述

原始 FreeRTOS（下文称 Vanilla FreeRTOS）是一款小巧高效的实时操作系统，适用于许多单核 MCU 和 SoC。但为了支持双核 ESP 芯片，如 ESP32、ESP32-S3、ESP32-P4，ESP-IDF 特别提供了支持双核对称多处理 (SMP) 的 FreeRTOS 实现（下文称 IDF FreeRTOS）。

IDF FreeRTOS 源代码基于 Vanilla FreeRTOS v10.5.1，但内核行为和 API 都有重大修改，以支持双核 SMP。不过用户也可以启用*CONFIG_FREERTOS_UNICORE* 选项，将 IDF FreeRTOS 配置为支持单核，详情请参阅单核模式。

备注： 本文档假定读者已具备 Vanilla FreeRTOS 的必要背景知识，即了解其特性、行为和 API 用法。如需了解背景知识，请参阅 Vanilla FreeRTOS 文档。

对称多处理

基本概念 对称多处理是一种计算架构，其中，两个及以上相同的 CPU 核连接到单个共享的主内存，并由单个操作系统控制。SMP 系统通常具有以下特点：

- 多个核独立运行。每个核都有自己的寄存器文件、中断和中断处理。
- 对每个核呈现相同的内存视图。因此，无论在哪个核上运行，访问特定内存地址的代码都会产生相同的效果。

与单核或非对称多处理系统相比，SMP 系统的主要优势在于：

- 存在多个核，支持多个硬件线程，从而提高整体处理吞吐量。
- 对称内存支持线程在执行期间切换核，从而提高 CPU 利用率。

尽管 SMP 系统支持线程切换核，但在某些情况下，线程必须或应该仅在特定核上运行。因此，在 SMP 系统中，线程也具备核亲和性，指定线程在哪个特定核上运行。

- 分配给特定核的线程只能在该核上运行。
- 未分配给特定核的线程支持在执行期间切换核。

ESP 芯片上的 SMP ESP32、ESP32-S3、ESP32-P4 等 ESP 芯片是双核 SMP SoC，具有以下硬件特性以支持 SMP：

- 具有两个完全相同的核，分别称为核 0 和核 1。代码段无论在哪个核上运行，都有相同的执行效果。
- 具有对称内存（除了少数例外情况）。
  - 如果多个核同时访问相同的内存地址，它们的访问会被内存总线串行化。
  - 通过 ISA 提供的原子比较和交换指令，可以实现对同一内存地址的真正原子访问。
- 跨核中断支持由一个核触发另一个核上的中断，这使得核间可以互相发送信号，如请求在另一个核上进行上下文切换。

备注： 在 ESP-IDF 中，核 0 和核 1 有时分别又被称为 PRO_CPU 和 APP_CPU。别名 PRO_CPU 和 APP_CPU 反映了典型 ESP-IDF 应用程序使用这两个 CPU 的方式。负责处理 Wi-Fi 或蓝牙等协议相关处理程序的任务通常会分配给核 0，因此称核 0 为 PRO_CPU；而处理应用程序其余部分的任务会分配给核 1，因此称核 1 为 APP_CPU。

任务

创建任务 Vanilla FreeRTOS 提供以下用于创建任务的函数：

- 使用 *xTaskCreate()* 创建任务时，任务内存动态分配。
- 使用 *xTaskCreateStatic()* 创建任务时，任务内存静态分配，即由用户提供。

然而，在 SMP 系统中，任务需要分配到特定核。因此，ESP-IDF 提供了 Vanilla FreeRTOS 任务创建函数的 ...PinnedToCore() 版本：

- 使用 *xTaskCreatePinnedToCore()* 可以创建具有特定核亲和性的任务，任务内存动态分配。
- 使用 *xTaskCreateStaticPinnedToCore()* 可以创建具有特定核亲和性的任务，任务内存静态分配，即由用户提供。

不同于普通的任务创建函数 API，...PinnedToCore() 版本的任务创建函数 API 有额外的 xCoreID 参数，用于指定所创建任务的核亲和性。核亲和性的有效值包括：

- 0：将创建的任务分配给核 0
- 1：将创建的任务分配给核 1
- tskNO_AFFINITY：支持任务在两个核上运行

注意，IDF FreeRTOS 仍支持普通的任务创建函数，但这些标准函数已经过调整，会内部调用其 ...PinnedToCore() 版本，同时将核亲和性设置为 tskNO_AFFINITY。

备注： IDF FreeRTOS 还更改了任务创建函数中的 ulStackDepth 参数。在 Vanilla FreeRTOS 中，任务堆栈的大小以字为单位指定，而在 IDF FreeRTOS 中，任务堆栈的大小以字节为单位指定。

执行任务 IDF FreeRTOS 中任务的结构与 Vanilla FreeRTOS 相同。具体而言，IDF FreeRTOS 任务：

- 只能处于以下任一状态：运行中、就绪、阻塞或挂起。
- 任务函数通常为无限循环。
- 任务函数不应返回。

删除任务 调用 *vTaskDelete()* 可以在 Vanilla FreeRTOS 中删除任务。该函数可用于删除其他任务，若任务句柄为 NULL 则删除当前运行任务。如果删除的任务是当前正在运行的任务时，任务的内存释放有时会委托给空闲任务执行。

IDF FreeRTOS 提供了同样的 *vTaskDelete()* 函数。然而，IDF FreeRTOS 是一个双核系统，因此调用 *vTaskDelete()* 时，行为上会与 Vanilla FreeRTOS 有以下差异：

- 删除另一个核上运行的任务时，会在另一个核上触发一次让步，任务内存由其中一个空闲任务释放。
- 如果删除的任务没有在任一核上运行，则会立即释放其内存。

请避免删除正在另一个核上运行的任务，否则由于无法确定该任务正在执行的操作，可能会导致难以预料的行为，例如：

- 删除持有互斥锁的任务。
- 删除尚未释放其先前分配的内存的任务。

请尽可能自己设计应用程序，确保在调用 *vTaskDelete()* 时，删除的任务处于已知状态。例如：

- 当任务完成执行操作并清理了任务内使用的所有资源时，任务调用 vTaskDelete(NULL) 自行删除。

- 在被另一个任务删除前，任务调用 *vTaskSuspend()* 将自己置于挂起状态。

SMP 调度器

对 Vanilla FreeRTOS 调度器最确切的描述是 **具有时间分片和固定优先级的抢占式调度器**，这意味着：

- 每个任务在创建时都赋予了固定优先级，调度器会执行具有最高优先级且处于就绪状态的任务。
- 调度器可以在不需要当前运行任务的协作下切换执行到另一个任务。
- 调度器会采用轮转方式，定期在具有相同优先级的就绪状态任务间切换执行，时间分片由时钟中断控制。

IDF FreeRTOS 调度器支持相同的调度特性，即固定优先级、抢占和时间分片，但也存在细微的行为差异。

固定优先级 在 Vanilla FreeRTOS 中，当调度器选择要运行的新任务时，往往会选择当前优先级最高的就绪任务。而在 IDF FreeRTOS 中，每个核都独立地调度要运行的任务。当特定核选择一个任务时，该核会选择优先级最高且可以在该核上运行的就绪状态任务。满足以下条件时，任务可以在核上运行：

- 任务亲和性兼容，即已分配或未分配给当前核。
- 该任务当前没有在其他核上运行。

但是，两个具有最高优先级的就绪任务不一定始终由调度器运行，因为还需考虑到任务的核亲和性。例如，给定以下任务：

- 优先级为 10 的任务 A，分配给核 0
- 优先级为 9 的任务 B，分配给核 0
- 优先级为 8 的任务 C，分配给核 1

经过调度后，任务 A 将在核 0 上运行，任务 C 将在核 1 上运行。即使任务 B 是第二优先级任务，也不会被执行。

抢占 在 Vanilla FreeRTOS 中，如果优先级更高的任务已准备好执行，调度器可以抢占当前正在运行的任务。同样，在 IDF FreeRTOS 任务中，如果调度器确定一个优先级更高的任务可以在某个核上运行，那么调度器可以单独抢占各个核。

但在某些情况下，一个优先级更高的就绪任务可以在多个核上运行。此时，调度器只会抢占一个核。即便当前有多个核可以抢占，调度器总是优先选择当前核。换句话说，如果优先级更高的就绪任务未分配，并且其优先级高于两个核的当前优先级，调度器将始终选择抢占当前核。例如，给定以下任务：

- 优先级为 8 的任务 A 当前在核 0 上运行
- 优先级为 9 的任务 B 当前在核 1 上运行
- 优先级为 10 的任务 C 未分配，并由任务 B 解除了阻塞

经过调度后，任务 A 将在核 0 上运行，任务 C 将抢占任务 B，因为调度器总是优先选择当前核。

时间分片 Vanilla FreeRTOS 实现了时间分片，这意味着如果当前优先级最高的就绪任务包含多个就绪任务，调度器会在这些任务间轮转定期切换。

然而，在 IDF FreeRTOS 中，由于以下原因，特定任务可能无法在特定核上运行，因此无法实现完美的轮转时间分片：

- 任务分配给了另一个核。
- 任务未分配，但已经由其他核运行。

因此，当核在所有就绪状态任务中搜索寻找要运行的任务时，可能需要跳过同一优先级列表中的一些任务，或者降低优先级，以找到可以运行的就绪状态任务。

IDF FreeRTOS 调度器会确保已选择运行的任务置于列表末尾，为同一优先级的就绪状态任务实现最佳轮转时间分片。这样，在下一次调度迭代（即，下一个滴答中断或让步）中，未经选择的任务优先级会更高。

以下示例展示了最佳轮转时间分片的实操。假设：

- 有四个相同优先级的就绪状态任务 `AX`、`B0`、`C1` 和 `D1`，其中：
  - 优先级是当前具有就绪状态任务的最高优先级。

  - 第一个字符代表任务名称，即 A、B、C、D。
  - 第二个字符表示任务核的分配情况，X 表示未分配。
- 任务列表始终从头开始搜索

1. 起始状态，尚未选择要运行的就绪状态任务。

```
Head [ AX , B0 , C1 , D0 ] Tail
```

2. 核 0 有一个滴答中断，搜索要运行的任务。选择任务 A，并将其移至列表末尾。

```
Core 0 ─┐
        ▼
Head [ AX , B0 , C1 , D0 ] Tail

                          [0]
Head [ B0 , C1 , D0 , AX ] Tail
```

3. 核 1 有一个滴答中断，搜索要运行的任务。由于亲和性不兼容，任务 B 无法运行，因此核 1 跳到任务 C。选择任务 C，并将其移至列表末尾。

```
Core 1 ──────┐
             ▼         [0]
Head [ B0 , C1 , D0 , AX ] Tail

                  [0]  [1]
Head [ B0 , D0 , AX , C1 ] Tail
```

4. 核 0 有另一个滴答中断，搜索要运行的任务。选择任务 B，并将其移至列表末尾。

```
Core 0 ─┐
        ▼              [1]
Head [ B0 , D0 , AX , C1 ] Tail

                  [1]  [0]
Head [ D0 , AX , C1 , B0 ] Tail
```

5. 核 1 有另一个滴答中断，搜索要运行的任务。由于亲和性不兼容，任务 D 无法运行，因此核 1 跳到任务 A。选择任务 A，并将其移至列表末尾。

```
Core 1 ──────┐
             ▼         [0]
Head [ D0 , AX , C1 , B0 ] Tail

                  [0]  [1]
Head [ D0 , C1 , B0 , AX ] Tail
```

在使用最佳轮转时间分片时需注意：

- 相同优先级的多个就绪状态任务不一定可以像在 Vanilla FreeRTOS 中一样按顺序运行。如以上示例所示，核可能会跳过任务。
- 然而，经过足够的滴答次数，任务最终将获得一些处理时间。
- 如果核找不到优先级最高的可运行就绪任务，它将降低优先级来搜索任务。
- 为了实现理想的轮转时间分片，应确保特定优先级的所有任务都分配给同一个核。

时钟中断 Vanilla FreeRTOS 要求定期发生滴答中断，滴答中断有以下作用：

- 增加调度器的滴答计数
- 为超时的阻塞任务解除阻塞
- 检查是否需要进行时间分片，即触发上下文切换
- 执行应用程序滴答函数

在 IDF FreeRTOS 中，每个核都会接收到定期中断，并独立运行滴答中断。每个核上的滴答中断周期相同，但可能不同步。然而，上述滴答中断任务不会由所有核同时执行，具体而言：

- 核 0 执行上述所有滴答中断任务

- 核 1 仅检查是否需要时间分片并执行应用程序滴答函数

备注： 在 IDF FreeRTOS 中，核 0 是负责时间计数的唯一核。因此，任何阻止核 0 增加滴答计数的情况，例如暂停核 0 上的调度器，都会导致整个调度器的时间计数滞后。

空闲任务 启动调度器时，Vanilla FreeRTOS 会隐式创建一个优先级为 0 的空闲任务。当没有其他任务准备运行时，空闲任务运行并有以下作用：

- 释放已删除任务的内存
- 执行应用程序的空闲函数

而 IDF FreeRTOS 为每个核单独创建了一个固定的空闲任务。每个核上的空闲任务起到与其 Vanilla FreeRTOS 对应任务相同的作用。

调度器挂起 Vanilla FreeRTOS 支持调用 *vTaskSuspendAll()* 挂起调度器，调用 *xTaskResumeAll()* 恢复调度器。调度器挂起时：

- 禁用任务切换，但仍启用中断。
- 禁止调用任何阻塞或让出函数，禁用时间分片。
- 时钟计数冻结，但仍会发生时钟中断以执行应用程序时钟函数。

调度器恢复时，*xTaskResumeAll()* 会补上所有丢失的时钟计数，并解除超时任务的阻塞。

在 IDF FreeRTOS 中，无法在多个核上同时挂起调度器。因此，在特定核上（如核 A）调用 *vTaskSuspendAll()* 时：

- 只在核 A 上禁用任务切换，但仍启用核 A 的中断。
- 禁止在核 A 上调用任何阻塞或让出函数，在核 A 上禁用时间分片。
- 核 A 上的中断解除任务阻塞时，对核 A 亲和的任务会进入核 A 的待执行任务列表。未分配的任务或对其他核亲和的任务可以在运行调度器的核上调度。
- 所有核上的调度器均挂起时，由中断解除阻塞的任务将进入它们分配的核的待执行任务列表。如果任务未分配，则进入调用中断的核的待执行任务列表。
- 如果核 A 是核 0，则时钟计数将被冻结，挂起的时钟计数将递增，但仍会发生时钟中断以执行应用程序时钟函数。

在特定核（如核 A）上调用 *xTaskResumeAll()* 时：

- 任何添加到核 A 的待执行任务列表中的任务将恢复执行。
- 如果核 A 是核 0，则挂起的时钟计数将回退，补上丢失的时钟计数。

警告： IDF FreeRTOS 上的调度器挂起仅暂停特定核上的调度，因此调度器挂起 **不能**确保访问共享数据时任务互斥。如果需要互斥，请使用适当的锁定机制，如互斥锁或自旋锁。

临界区

禁用中断 Vanilla FreeRTOS 支持通过调用 *taskDISABLE_INTERRUPTS* 和 *taskENABLE_INTERRUPTS* 分别禁用和启用中断。IDF FreeRTOS 提供了相同的 API，但中断只能在当前核上禁用或启用。

在 Vanilla FreeRTOS 以及其他普通单核系统中，禁用中断可以有效实现互斥，**但在 SMP 系统中，禁用中断并不能确保实现互斥**，而应使用有自旋锁的临界区以实现互斥。

API 变更 Vanilla FreeRTOS 通过禁用中断实现临界区 (Critical Section)，以防止在临界区内发生抢占式上下文切换和中断服务，确保进入临界区的任务或 ISR 是访问共享资源的唯一实体。Vanilla FreeRTOS 中的临界区提供以下 API：

- `taskENTER_CRITICAL()` 通过禁用中断进入临界区

- taskEXIT_CRITICAL() 通过重新启用中断退出临界区
- taskENTER_CRITICAL_FROM_ISR() 通过禁用中断嵌套从 ISR 进入临界区
- taskEXIT_CRITICAL_FROM_ISR() 通过重新启用中断嵌套从 ISR 退出临界区

然而，在 SMP 系统中，仅禁用中断并不能构成临界区，因为存在其他核意味着共享资源仍可以同时访问。因此，IDF FreeRTOS 中的临界区是使用自旋锁实现的。为适应自旋锁，IDF FreeRTOS 中的临界区 API 包含一个额外的自旋锁参数，具体如下：

- 自旋锁为 portMUX_TYPE (**请勿与 FreeRTOS 互斥混淆**)
- taskENTER_CRITICAL(&spinlock) 从任务上下文进入临界区
- taskEXIT_CRITICAL(&spinlock) 从任务上下文退出临界区
- taskENTER_CRITICAL_ISR(&spinlock) 从中断上下文进入临界区
- taskEXIT_CRITICAL_ISR(&spinlock) 从中断上下文退出临界区

备注： 临界区 API 可以递归调用，即可以嵌套使用临界区。只要退出临界区的次数与进入的次数相同，多次递归进入临界区就是有效的。但是，由于临界区可以针对不同的自旋锁，因此在递归进入临界区时，应注意避免死锁。

自旋锁可以静态或动态分配。因此，提供了静态和动态初始化自旋锁的宏，如以下代码片段所示。

- 静态分配自旋锁并使用 portMUX_INITIALIZER_UNLOCKED 初始化：

```
// 静态分配并初始化自旋锁
static portMUX_TYPE my_spinlock = portMUX_INITIALIZER_UNLOCKED;

void some_function(void)
{
    taskENTER_CRITICAL(&my_spinlock);
    // 此时已处于临界区
    taskEXIT_CRITICAL(&my_spinlock);
}
```

- 动态分配自旋锁并使用 portMUX_INITIALIZE() 初始化：

```
// 动态分配自旋锁
portMUX_TYPE *my_spinlock = malloc(sizeof(portMUX_TYPE));
// 动态初始化自旋锁
portMUX_INITIALIZE(my_spinlock);

...

taskENTER_CRITICAL(my_spinlock);
// 访问资源
taskEXIT_CRITICAL(my_spinlock);
```

实现 IDF FreeRTOS 中，特定核进入和退出临界区的过程如下：

- 对于 taskENTER_CRITICAL(&spinlock) 或 taskENTER_CRITICAL_ISR(&spinlock)
  1. 核禁用其中断或中断嵌套，直到达到 configMAX_SYSCALL_INTERRUPT_PRIORITY。
  2. 接着，核使用原子比较和设置指令在自旋锁上自旋，直到获取锁。当核能够将锁的所有者值设置为核的 ID 时，就获得了锁。
  3. 一旦获取了自旋锁，函数返回。剩余的临界区部分将在禁用中断或中断嵌套的情况下运行。
- 对于 taskEXIT_CRITICAL(&spinlock) 或 taskEXIT_CRITICAL_ISR(&spinlock)
  1. 核通过清除自旋锁的所有者值释放自旋锁。
  2. 核重新启用中断或中断嵌套。

限制与注意事项 由于在临界区内禁用了中断或中断嵌套，产生了多个关于在临界区内可执行操作的限制，请牢记以下操作限制和注意事项：

- 临界区应尽可能短

  - 临界区持续时间越长，越可能延迟挂起中断的执行。
  - 临界区通常应仅涉及少量数据结构和/或硬件寄存器。
  - 如果可以，尽可能将执行操作和/或事件处理程序推迟到临界区之外。
- 不应在临界区内调用 FreeRTOS API
- 不应在临界区内调用任何阻塞或让出函数

其他事项

使用浮点 通常情况下，当发生上下文切换时：

- 核寄存器的当前状态保存到要切出的任务栈中
- 核寄存器的先前保存状态从要切入的任务栈中加载

然而，IDF FreeRTOS 为核的浮点运算单元 (FPU) 寄存器实现了延迟上下文切换。换句话说，当在特定核上（如核 0）发生上下文切换时，核的 FPU 寄存器状态不会立即保存到要被切出的任务的堆栈中（如任务 A）。FPU 的寄存器在发生以下情况前将保持不变：

- 另一个任务（如任务 B）在同一核上运行并使用 FPU，这将触发异常，将 FPU 寄存器保存到任务 A 的堆栈中。
- 任务 A 重新调度到同一核并继续执行。在这种情况下，不需要保存和恢复 FPU 的寄存器。

然而，由于任务并未分配给某一核，可以随意调度（如任务 A 切换到核 1），因此很难实现跨核复制和恢复 FPU 寄存器状态。因此，当任务在其执行流程中用 `float` 类型使用 FPU 时，IDF FreeRTOS 会自动将任务分配给当前正在运行的核，确保所有使用 FPU 的任务始终在特定核上运行。

此外，请注意，由于 FPU 寄存器状态与特定任务相关联，IDF FreeRTOS 默认不支持在中断上下文中使用 FPU。

备注： 具有 FPU 的 ESP 芯片不支持双精度浮点运算 `double` 的硬件加速。`double` 通过软件实现，因此比起 `float` 类型，`double` 操作可能消耗更多 CPU 时间。

单核模式 尽管 IDF FreeRTOS 是为双核 SMP 专门设计的，但也可通过启用*CONFIG_FREERTOS_UNICORE* 选项，将 IDF FreeRTOS 配置为支持单核。

对于 ESP32-S2 和 ESP32-C3 等单核芯片，*CONFIG_FREERTOS_UNICORE* 选项始终启用。对于 ESP32 和 ESP32-S3 等多核芯片也可以设置*CONFIG_FREERTOS_UNICORE*，对于多核目标（如 ESP32 和 ESP32-S3），也可以设置*CONFIG_FREERTOS_UNICORE*，但启用该选项后应用仅在核 0 上运行。

在单核模式下，IDF FreeRTOS 与 Vanilla FreeRTOS 完全相同，因此无需考虑前文提到的对内核行为的 SMP 更改。因此，在单核模式下构建 IDF FreeRTOS 具有以下特点：

- 内核在临界区内执行的所有操作都是确定的（即在临界区内不走链表）。
- 恢复了 Vanilla FreeRTOS 调度算法（包括完美的轮转时间分片）。
- 移除了所有单核构建中的 SMP 专用数据。

在单核模式下仍可调用 SMP API，这些 API 仍然保持公开，以便为单核和多核构建源代码，而无需调用不同的 API 集。不过，SMP API 在单核模式下不会展示任何 SMP 行为，因此实际上等同于其对应的单核模式 API。例如：

- 任何 `...ForCore(..., BaseType_t xCoreID)` SMP API 将只接受 `0` 为 `xCoreID` 的有效值。
- `...PinnedToCore()` 任务创建 API 将直接忽略 `xCoreID` 核亲和参数。
- 临界区 API 仍需要自旋锁参数，但不会使用自旋锁，临界区将恢复为仅用于禁用/启用中断。

API 参考

本节介绍了 FreeRTOS 类型、函数和宏，均从 FreeRTOS 头文件自动生成。

任务 API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/task.h
- This header file can be included with:

```
#include "freertos/task.h"
```

Functions

static inline BaseType_t **xTaskCreate**(TaskFunction_t pxTaskCode, const char *const pcName, const configSTACK_DEPTH_TYPE usStackDepth, void *const pvParameters, UBaseType_t uxPriority, *TaskHandle_t* *const pxCreatedTask)

Create a new task and add it to the list of tasks that are ready to run.

Internally, within the FreeRTOS implementation, tasks use two blocks of memory. The first block is used to hold the task's data structures. The second block is used by the task as its stack. If a task is created using xTaskCreate() then both blocks of memory are automatically dynamically allocated inside the xTaskCreate() function. (see https://www.FreeRTOS.org/a00111.html). If a task is created using xTaskCreateStatic() then the application writer must provide the required memory. xTaskCreateStatic() therefore allows a task to be created without using any dynamic memory allocation.

See xTaskCreateStatic() for a version that does not use any dynamic memory allocation.

xTaskCreate() can only be used to create a task that has unrestricted access to the entire microcontroller memory map. Systems that include MPU support can alternatively create an MPU constrained task using xTaskCreateRestricted().

Example usage:

```
// Task to be created.
void vTaskCode( void * pvParameters )
{
for( ;; )
  {
// Task code goes here.
  }
}

// Function that creates a task.
void vOtherFunction( void )
{
static uint8_t ucParameterToPass;
TaskHandle_t xHandle = NULL;

// Create the task, storing the handle.  Note that the passed parameter␣
↪ucParameterToPass
// must exist for the lifetime of the task, so in this case is declared static.
↪  If it was just an
// an automatic stack variable it might no longer exist, or at least have been␣
↪corrupted, by the time
// the new task attempts to access it.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, &ucParameterToPass, tskIDLE_
↪PRIORITY, &xHandle );
  configASSERT( xHandle );

// Use the handle to delete the task.
if( xHandle != NULL )
  {
     vTaskDelete( xHandle );
```

(下页继续)

(续上页)

```
    }
}
```

备 注: If configNUMBER_OF_CORES > 1, this function will create an unpinned task (see tskNO_AFFINITY for more details).

备注: If program uses thread local variables (ones specified with "__thread" keyword) then storage for them will be allocated on the task's stack.

参数

- **pxTaskCode** -- Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- **pcName** -- A descriptive name for the task. This is mainly used to facilitate debugging. Max length defined by configMAX_TASK_NAME_LEN - default is 16.
- **usStackDepth** -- The size of the task stack specified as the NUMBER OF BYTES. Note that this differs from vanilla FreeRTOS.
- **pvParameters** -- Pointer that will be used as the parameter for the task being created.
- **uxPriority** -- The priority at which the task should run. Systems that include MPU support can optionally create tasks in a privileged (system) mode by setting bit portPRIVILEGE_BIT of the priority parameter. For example, to create a privileged task at priority 2 the uxPriority parameter should be set to (2 | portPRIVILEGE_BIT).
- **pxCreatedTask** -- Used to pass back a handle by which the created task can be referenced.

返回 pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h

static inline *TaskHandle_t* **xTaskCreateStatic**(TaskFunction_t pxTaskCode, const char *const pcName, const uint32_t ulStackDepth, void *const pvParameters, UBaseType_t uxPriority, StackType_t *const puxStackBuffer, StaticTask_t *const pxTaskBuffer)

Create a new task and add it to the list of tasks that are ready to run.

Internally, within the FreeRTOS implementation, tasks use two blocks of memory. The first block is used to hold the task's data structures. The second block is used by the task as its stack. If a task is created using xTaskCreate() then both blocks of memory are automatically dynamically allocated inside the xTaskCreate() function. (see https://www.FreeRTOS.org/a00111.html). If a task is created using xTaskCreateStatic() then the application writer must provide the required memory. xTaskCreateStatic() therefore allows a task to be created without using any dynamic memory allocation.

Example usage:

```
// Dimensions the buffer that the task being created will use as its stack.
// NOTE:  This is the number of bytes the stack will hold, not the number of
// words as found in vanilla FreeRTOS.
#define STACK_SIZE 200

// Structure that will hold the TCB of the task being created.
   StaticTask_t xTaskBuffer;

// Buffer that the task being created will use as its stack.  Note this is
// an array of StackType_t variables.  The size of StackType_t is dependent on
// the RTOS port.
   StackType_t xStack[ STACK_SIZE ];
```

(下页继续)

(续上页)

```
// Function that implements the task being created.
void vTaskCode( void * pvParameters )
    {
// The parameter value is expected to be 1 as 1 is passed in the
// pvParameters value in the call to xTaskCreateStatic().
        configASSERT( ( uint32_t ) pvParameters == 1UL );

for( ;; )
        {
// Task code goes here.
        }
    }

// Function that creates a task.
void vOtherFunction( void )
    {
        TaskHandle_t xHandle = NULL;

// Create the task without using any dynamic memory allocation.
        xHandle = xTaskCreateStatic(
                      vTaskCode,       // Function that implements the task.
"NAME",          // Text name for the task.
                      STACK_SIZE,      // Stack size in bytes.
                      ( void * ) 1,    // Parameter passed into the task.
                      tskIDLE_PRIORITY,// Priority at which the task is created.
                      xStack,          // Array to use as the task's stack.
                      &xTaskBuffer );  // Variable to hold the task's data␣
↪structure.

// puxStackBuffer and pxTaskBuffer were not NULL, so the task will have
// been created, and xHandle will be the task's handle.  Use the handle
// to suspend the task.
        vTaskSuspend( xHandle );
    }
```

备 注: If configNUMBER_OF_CORES > 1, this function will create an unpinned task (see tskNO_AFFINITY for more details).

备注: If program uses thread local variables (ones specified with "__thread" keyword) then storage for them will be allocated on the task's stack.

参数

- **pxTaskCode** -- Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- **pcName** -- A descriptive name for the task. This is mainly used to facilitate debugging. The maximum length of the string is defined by configMAX_TASK_NAME_LEN in FreeRTOSConfig.h.
- **ulStackDepth** -- The size of the task stack specified as the NUMBER OF BYTES. Note that this differs from vanilla FreeRTOS.
- **pvParameters** -- Pointer that will be used as the parameter for the task being created.
- **uxPriority** -- The priority at which the task will run.
- **puxStackBuffer** -- Must point to a StackType_t array that has at least ulStackDepth indexes - the array will then be used as the task's stack, removing the need for the stack to be allocated dynamically.
- **pxTaskBuffer** -- Must point to a variable of type StaticTask_t, which will then be used to hold the task's data structures, removing the need for the memory to be allocated

dynamically.

返回 If neither puxStackBuffer nor pxTaskBuffer are NULL, then the task will be created and a handle to the created task is returned. If either puxStackBuffer or pxTaskBuffer are NULL then the task will not be created and NULL is returned.

void **vTaskAllocateMPURegions**(*TaskHandle_t* xTask, const MemoryRegion_t *const pxRegions)

Memory regions are assigned to a restricted task when the task is created by a call to xTaskCreateRestricted(). These regions can be redefined using vTaskAllocateMPURegions().

Example usage:

```
// Define an array of MemoryRegion_t structures that configures an MPU region
// allowing read/write access for 1024 bytes starting at the beginning of the
// ucOneKByte array.  The other two of the maximum 3 definable regions are
// unused so set to zero.
static const MemoryRegion_t xAltRegions[ portNUM_CONFIGURABLE_REGIONS ] =
{
 // Base address      Length      Parameters
 { ucOneKByte,       1024,       portMPU_REGION_READ_WRITE },
 { 0,                0,          0 },
 { 0,                0,          0 }
};

void vATask( void *pvParameters )
{
 // This task was created such that it has access to certain regions of
 // memory as defined by the MPU configuration.  At some point it is
 // desired that these MPU regions are replaced with that defined in the
 // xAltRegions const struct above.  Use a call to vTaskAllocateMPURegions()
 // for this purpose.  NULL is used as the task handle to indicate that this
 // function should modify the MPU regions of the calling task.
 vTaskAllocateMPURegions( NULL, xAltRegions );

 // Now the task can continue its function, but from this point on can only
 // access its stack and the ucOneKByte array (unless any other statically
 // defined or shared regions have been declared elsewhere).
}
```

参数

- **xTask** -- The handle of the task being updated.
- **pxRegions** -- A pointer to a MemoryRegion_t structure that contains the new memory region definitions.

void **vTaskDelete**(*TaskHandle_t* xTaskToDelete)

INCLUDE_vTaskDelete must be defined as 1 for this function to be available. See the configuration section for more information.

Remove a task from the RTOS real time kernel's management. The task being deleted will be removed from all ready, blocked, suspended and event lists.

NOTE: The idle task is responsible for freeing the kernel allocated memory from tasks that have been deleted. It is therefore important that the idle task is not starved of microcontroller processing time if your application makes any calls to vTaskDelete (). Memory allocated by the task code is not automatically freed, and should be freed before the task is deleted.

See the demo application file death.c for sample code that utilises vTaskDelete ().

Example usage:

```
void vOtherFunction( void )
{
TaskHandle_t xHandle;

// Create the task, storing the handle.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle
↪);

// Use the handle to delete the task.
  vTaskDelete( xHandle );
}
```

参数 xTaskToDelete -- The handle of the task to be deleted. Passing NULL will cause the calling task to be deleted.

void **vTaskDelay**(const TickType_t xTicksToDelay)

Delay a task for a given number of ticks. The actual time that the task remains blocked depends on the tick rate. The constant portTICK_PERIOD_MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

INCLUDE_vTaskDelay must be defined as 1 for this function to be available. See the configuration section for more information.

vTaskDelay() specifies a time at which the task wishes to unblock relative to the time at which vTaskDelay() is called. For example, specifying a block period of 100 ticks will cause the task to unblock 100 ticks after vTaskDelay() is called. vTaskDelay() does not therefore provide a good method of controlling the frequency of a periodic task as the path taken through the code, as well as other task and interrupt activity, will affect the frequency at which vTaskDelay() gets called and therefore the time at which the task next executes. See xTaskDelayUntil() for an alternative API function designed to facilitate fixed frequency execution. It does this by specifying an absolute time (rather than a relative time) at which the calling task should unblock.

Example usage:

```
void vTaskFunction( void * pvParameters )
{
// Block for 500ms.
const TickType_t xDelay = 500 / portTICK_PERIOD_MS;

for( ;; )
  {
// Simply toggle the LED every 500ms, blocking between each toggle.
      vToggleLED();
      vTaskDelay( xDelay );
  }
}
```

参数 xTicksToDelay -- The amount of time, in tick periods, that the calling task should block.

BaseType_t **xTaskDelayUntil**(TickType_t *const pxPreviousWakeTime, const TickType_t xTimeIncrement)

INCLUDE_xTaskDelayUntil must be defined as 1 for this function to be available. See the configuration section for more information.

Delay a task until a specified time. This function can be used by periodic tasks to ensure a constant execution frequency.

This function differs from vTaskDelay () in one important aspect: vTaskDelay () will cause a task to block for the specified number of ticks from the time vTaskDelay () is called. It is therefore difficult to use vTaskDelay () by itself to generate a fixed execution frequency as the time between a task starting to execute and that task

calling vTaskDelay () may not be fixed [the task may take a different path though the code between calls, or may get interrupted or preempted a different number of times each time it executes].

Whereas vTaskDelay () specifies a wake time relative to the time at which the function is called, xTaskDelayUntil () specifies the absolute (exact) time at which it wishes to unblock.

The macro pdMS_TO_TICKS() can be used to calculate the number of ticks from a time specified in milliseconds with a resolution of one tick period.

Example usage:

```
// Perform an action every 10 ticks.
void vTaskFunction( void * pvParameters )
{
TickType_t xLastWakeTime;
const TickType_t xFrequency = 10;
BaseType_t xWasDelayed;

// Initialise the xLastWakeTime variable with the current time.
    xLastWakeTime = xTaskGetTickCount ();
for( ;; )
    {
// Wait for the next cycle.
        xWasDelayed = xTaskDelayUntil( &xLastWakeTime, xFrequency );

// Perform action here. xWasDelayed value can be used to determine
// whether a deadline was missed if the code here took too long.
    }
}
```

参数

- **pxPreviousWakeTime** -- Pointer to a variable that holds the time at which the task was last unblocked. The variable must be initialised with the current time prior to its first use (see the example below). Following this the variable is automatically updated within xTaskDelayUntil ().
- **xTimeIncrement** -- The cycle time period. The task will be unblocked at time *pxPreviousWakeTime + xTimeIncrement. Calling xTaskDelayUntil with the same xTimeIncrement parameter value will cause the task to execute with a fixed interface period.

返回 Value which can be used to check whether the task was actually delayed. Will be pdTRUE if the task way delayed and pdFALSE otherwise. A task will not be delayed if the next expected wake time is in the past.

BaseType_t **xTaskAbortDelay**(*TaskHandle_t* xTask)

INCLUDE_xTaskAbortDelay must be defined as 1 in FreeRTOSConfig.h for this function to be available.

A task will enter the Blocked state when it is waiting for an event. The event it is waiting for can be a temporal event (waiting for a time), such as when vTaskDelay() is called, or an event on an object, such as when xQueueReceive() or ulTaskNotifyTake() is called. If the handle of a task that is in the Blocked state is used in a call to xTaskAbortDelay() then the task will leave the Blocked state, and return from whichever function call placed the task into the Blocked state.

There is no 'FromISR' version of this function as an interrupt would need to know which object a task was blocked on in order to know which actions to take. For example, if the task was blocked on a queue the interrupt handler would then need to know if the queue was locked.

参数 **xTask** -- The handle of the task to remove from the Blocked state.

返回 If the task referenced by xTask was not in the Blocked state then pdFAIL is returned. Otherwise pdPASS is returned.

UBaseType_t **uxTaskPriorityGet**(const *TaskHandle_t* xTask)

INCLUDE_uxTaskPriorityGet must be defined as 1 for this function to be available. See the configuration section for more information.

Obtain the priority of any task.

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;

// Create a task, storing the handle.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle␣
↪);

// ...

// Use the handle to obtain the priority of the created task.
// It was created with tskIDLE_PRIORITY, but may have changed
// it itself.
if( uxTaskPriorityGet( xHandle ) != tskIDLE_PRIORITY )
  {
// The task has changed it's priority.
  }

// ...

// Is our priority higher than the created task?
if( uxTaskPriorityGet( xHandle ) < uxTaskPriorityGet( NULL ) )
  {
// Our priority (obtained using NULL handle) is higher.
  }
}
```

参数 **xTask** -- Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.

返回 The priority of xTask.

UBaseType_t **uxTaskPriorityGetFromISR**(const *TaskHandle_t* xTask)

A version of uxTaskPriorityGet() that can be used from an ISR.

eTaskState **eTaskGetState**(*TaskHandle_t* xTask)

INCLUDE_eTaskGetState must be defined as 1 for this function to be available. See the configuration section for more information.

Obtain the state of any task. States are encoded by the eTaskState enumerated type.

参数 **xTask** -- Handle of the task to be queried.

返回 The state of xTask at the time the function was called. Note the state of the task might change between the function being called, and the functions return value being tested by the calling task.

void **vTaskGetInfo**(*TaskHandle_t* xTask, *TaskStatus_t* *pxTaskStatus, BaseType_t xGetFreeStackSpace, *eTaskState* eState)

configUSE_TRACE_FACILITY must be defined as 1 for this function to be available. See the configuration section for more information.

Populates a TaskStatus_t structure with information about a task.

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;
TaskStatus_t xTaskDetails;

 // Obtain the handle of a task from its name.
 xHandle = xTaskGetHandle( "Task_Name" );

 // Check the handle is not NULL.
 configASSERT( xHandle );

 // Use the handle to obtain further information about the task.
 vTaskGetInfo( xHandle,
               &xTaskDetails,
               pdTRUE, // Include the high water mark in xTaskDetails.
               eInvalid ); // Include the task state in xTaskDetails.
}
```

参数

- **xTask** -- Handle of the task being queried. If xTask is NULL then information will be returned about the calling task.
- **pxTaskStatus** -- A pointer to the TaskStatus_t structure that will be filled with information about the task referenced by the handle passed using the xTask parameter.
- **xGetFreeStackSpace** -- The TaskStatus_t structure contains a member to report the stack high water mark of the task being queried. Calculating the stack high water mark takes a relatively long time, and can make the system temporarily unresponsive - so the xGetFreeStackSpace parameter is provided to allow the high water mark checking to be skipped. The high watermark value will only be written to the TaskStatus_t structure if xGetFreeStackSpace is not set to pdFALSE;
- **eState** -- The TaskStatus_t structure contains a member to report the state of the task being queried. Obtaining the task state is not as fast as a simple assignment - so the eState parameter is provided to allow the state information to be omitted from the TaskStatus_t structure. To obtain state information then set eState to eInvalid - otherwise the value passed in eState will be reported as the task state in the TaskStatus_t structure.

void **vTaskPrioritySet**(*TaskHandle_t* xTask, UBaseType_t uxNewPriority)

INCLUDE_vTaskPrioritySet must be defined as 1 for this function to be available. See the configuration section for more information.

Set the priority of any task.

A context switch will occur before the function returns if the priority being set is higher than the currently executing task.

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;

// Create a task, storing the handle.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle␣
↪);

// ...

// Use the handle to raise the priority of the created task.
```

(下页继续)

(续上页)

```
  vTaskPrioritySet( xHandle, tskIDLE_PRIORITY + 1 );

// ...

// Use a NULL handle to raise our priority to the same value.
  vTaskPrioritySet( NULL, tskIDLE_PRIORITY + 1 );
}
```

参数

- **xTask** -- Handle to the task for which the priority is being set. Passing a NULL handle results in the priority of the calling task being set.
- **uxNewPriority** -- The priority to which the task will be set.

void **vTaskSuspend**(*TaskHandle_t* xTaskToSuspend)

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

Suspend any task. When suspended a task will never get any microcontroller processing time, no matter what its priority.

Calls to vTaskSuspend are not accumulative - i.e. calling vTaskSuspend () twice on the same task still only requires one call to vTaskResume () to ready the suspended task.

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;

// Create a task, storing the handle.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle␣
↪);

// ...

// Use the handle to suspend the created task.
  vTaskSuspend( xHandle );

// ...

// The created task will not run during this period, unless
// another task calls vTaskResume( xHandle ).

//...


// Suspend ourselves.
  vTaskSuspend( NULL );

// We cannot get here unless another task calls vTaskResume
// with our handle as the parameter.
}
```

参数 **xTaskToSuspend** -- Handle to the task being suspended. Passing a NULL handle will cause the calling task to be suspended.

void **vTaskResume**(*TaskHandle_t* xTaskToResume)

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

Resumes a suspended task.

A task that has been suspended by one or more calls to vTaskSuspend () will be made available for running again by a single call to vTaskResume ().

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;

// Create a task, storing the handle.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle␣
↪);

// ...

// Use the handle to suspend the created task.
  vTaskSuspend( xHandle );

// ...

// The created task will not run during this period, unless
// another task calls vTaskResume( xHandle ).

//...


// Resume the suspended task ourselves.
  vTaskResume( xHandle );

// The created task will once again get microcontroller processing
// time in accordance with its priority within the system.
}
```

参数 **xTaskToResume** -- Handle to the task being readied.

BaseType_t **xTaskResumeFromISR** (*TaskHandle_t* xTaskToResume)

INCLUDE_xTaskResumeFromISR must be defined as 1 for this function to be available. See the configuration section for more information.

An implementation of vTaskResume() that can be called from within an ISR.

A task that has been suspended by one or more calls to vTaskSuspend () will be made available for running again by a single call to xTaskResumeFromISR ().

xTaskResumeFromISR() should not be used to synchronise a task with an interrupt if there is a chance that the interrupt could arrive prior to the task being suspended - as this can lead to interrupts being missed. Use of a semaphore as a synchronisation mechanism would avoid this eventuality.

参数 **xTaskToResume** -- Handle to the task being readied.

返回 pdTRUE if resuming the task should result in a context switch, otherwise pdFALSE. This is used by the ISR to determine if a context switch may be required following the ISR.

void **vTaskSuspendAll** (void)

Suspends the scheduler without disabling interrupts. Context switches will not occur while the scheduler is suspended.

After calling vTaskSuspendAll () the calling task will continue to execute without risk of being swapped out until a call to xTaskResumeAll () has been made.

API functions that have the potential to cause a context switch (for example, xTaskDelayUntil(), xQueueSend(), etc.) must not be called while the scheduler is suspended.

Example usage:

```
void vTask1( void * pvParameters )
{
for( ;; )
  {
// Task code goes here.

// ...

// At some point the task wants to perform a long operation during
// which it does not want to get swapped out.  It cannot use
// taskENTER_CRITICAL ()/taskEXIT_CRITICAL () as the length of the
// operation may cause interrupts to be missed - including the
// ticks.

// Prevent the real time kernel swapping out the task.
      vTaskSuspendAll ();

// Perform the operation here.  There is no need to use critical
// sections as we have all the microcontroller processing time.
// During this time interrupts will still operate and the kernel
// tick count will be maintained.

// ...

// The operation is complete.  Restart the kernel.
      xTaskResumeAll ();
  }
}
```

BaseType_t **xTaskResumeAll** (void)

Resumes scheduler activity after it was suspended by a call to vTaskSuspendAll().

xTaskResumeAll() only resumes the scheduler. It does not unsuspend tasks that were previously suspended by a call to vTaskSuspend().

Example usage:

```
void vTask1( void * pvParameters )
{
for( ;; )
  {
// Task code goes here.

// ...

// At some point the task wants to perform a long operation during
// which it does not want to get swapped out.  It cannot use
// taskENTER_CRITICAL ()/taskEXIT_CRITICAL () as the length of the
// operation may cause interrupts to be missed - including the
// ticks.

// Prevent the real time kernel swapping out the task.
      vTaskSuspendAll ();

// Perform the operation here.  There is no need to use critical
// sections as we have all the microcontroller processing time.
// During this time interrupts will still operate and the real
// time kernel tick count will be maintained.
```

(下页继续)

(续上页)

```
// ...

// The operation is complete.  Restart the kernel.  We want to force
// a context switch - but there is no point if resuming the scheduler
// caused a context switch already.
if( !xTaskResumeAll () )
      {
           taskYIELD ();
      }
  }
}
```

返回 If resuming the scheduler caused a context switch then pdTRUE is returned, otherwise pdFALSE is returned.

TickType_t **xTaskGetTickCount**(void)

返回 The count of ticks since vTaskStartScheduler was called.

TickType_t **xTaskGetTickCountFromISR**(void)

This is a version of xTaskGetTickCount() that is safe to be called from an ISR - provided that TickType_t is the natural word size of the microcontroller being used or interrupt nesting is either not supported or not being used.

返回 The count of ticks since vTaskStartScheduler was called.

UBaseType_t **uxTaskGetNumberOfTasks**(void)

返回 The number of tasks that the real time kernel is currently managing. This includes all ready, blocked and suspended tasks. A task that has been deleted but not yet freed by the idle task will also be included in the count.

char *__pcTaskGetName__(*TaskHandle_t* xTaskToQuery)

返回 The text (human readable) name of the task referenced by the handle xTaskToQuery. A task can query its own name by either passing in its own handle, or by setting xTaskToQuery to NULL.

TaskHandle_t **xTaskGetHandle**(const char *pcNameToQuery)

NOTE: This function takes a relatively long time to complete and should be used sparingly.

返回 The handle of the task that has the human readable name pcNameToQuery. NULL is returned if no matching name is found. INCLUDE_xTaskGetHandle must be set to 1 in FreeRTOSConfig.h for pcTaskGetHandle() to be available.

BaseType_t **xTaskGetStaticBuffers**(*TaskHandle_t* xTask, StackType_t **ppuxStackBuffer, StaticTask_t **ppxTaskBuffer)

Retrieve pointers to a statically created task's data structure buffer and stack buffer. These are the same buffers that are supplied at the time of creation.

参数

- **xTask** -- The task for which to retrieve the buffers.
- **ppuxStackBuffer** -- Used to return a pointer to the task's stack buffer.
- **ppxTaskBuffer** -- Used to return a pointer to the task's data structure buffer.

返回 pdTRUE if buffers were retrieved, pdFALSE otherwise.

UBaseType_t **uxTaskGetStackHighWaterMark**(*TaskHandle_t* xTask)

INCLUDE_uxTaskGetStackHighWaterMark must be set to 1 in FreeRTOSConfig.h for this function to be available.

Returns the high water mark of the stack associated with xTask. That is, the minimum free stack space there has been (in words, so on a 32 bit machine a value of 1 means 4 bytes) since the task started. The smaller the returned number the closer the task has come to overflowing its stack.

uxTaskGetStackHighWaterMark() and uxTaskGetStackHighWaterMark2() are the same except for their return type. Using configSTACK_DEPTH_TYPE allows the user to determine the return type. It gets around the problem of the value overflowing on 8-bit types without breaking backward compatibility for applications that expect an 8-bit return type.

参数 **xTask** -- Handle of the task associated with the stack to be checked. Set xTask to NULL to check the stack of the calling task.

返回 The smallest amount of free stack space there has been (in words, so actual spaces on the stack rather than bytes) since the task referenced by xTask was created.

configSTACK_DEPTH_TYPE **uxTaskGetStackHighWaterMark2**(*TaskHandle_t* xTask)

INCLUDE_uxTaskGetStackHighWaterMark2 must be set to 1 in FreeRTOSConfig.h for this function to be available.

Returns the high water mark of the stack associated with xTask. That is, the minimum free stack space there has been (in words, so on a 32 bit machine a value of 1 means 4 bytes) since the task started. The smaller the returned number the closer the task has come to overflowing its stack.

uxTaskGetStackHighWaterMark() and uxTaskGetStackHighWaterMark2() are the same except for their return type. Using configSTACK_DEPTH_TYPE allows the user to determine the return type. It gets around the problem of the value overflowing on 8-bit types without breaking backward compatibility for applications that expect an 8-bit return type.

参数 **xTask** -- Handle of the task associated with the stack to be checked. Set xTask to NULL to check the stack of the calling task.

返回 The smallest amount of free stack space there has been (in words, so actual spaces on the stack rather than bytes) since the task referenced by xTask was created.

void **vTaskSetApplicationTaskTag**(*TaskHandle_t* xTask, *TaskHookFunction_t* pxHookFunction)

Sets pxHookFunction to be the task hook function used by the task xTask. Passing xTask as NULL has the effect of setting the calling tasks hook function.

TaskHookFunction_t **xTaskGetApplicationTaskTag**(*TaskHandle_t* xTask)

Returns the pxHookFunction value assigned to the task xTask. Do not call from an interrupt service routine - call xTaskGetApplicationTaskTagFromISR() instead.

TaskHookFunction_t **xTaskGetApplicationTaskTagFromISR**(*TaskHandle_t* xTask)

Returns the pxHookFunction value assigned to the task xTask. Can be called from an interrupt service routine.

void **vTaskSetThreadLocalStoragePointer**(*TaskHandle_t* xTaskToSet, BaseType_t xIndex, void *pvValue)

Each task contains an array of pointers that is dimensioned by the configNUM_THREAD_LOCAL_STORAGE_POINTERS setting in FreeRTOSConfig.h. The kernel does not use the pointers itself, so the application writer can use the pointers for any purpose they wish. The following two functions are used to set and query a pointer respectively.

void ***pvTaskGetThreadLocalStoragePointer**(*TaskHandle_t* xTaskToQuery, BaseType_t xIndex)

void **vApplicationGetIdleTaskMemory**(StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize)

This function is used to provide a statically allocated block of memory to FreeRTOS to hold the Idle Task TCB. This function is required when configSUPPORT_STATIC_ALLOCATION is set. For more information see this URI: https://www.FreeRTOS.org/a00110.html#configSUPPORT_STATIC_ALLOCATION

参数

- **ppxIdleTaskTCBBuffer** -- A handle to a statically allocated TCB buffer
- **ppxIdleTaskStackBuffer** -- A handle to a statically allocated Stack buffer for the idle task

- **pulIdleTaskStackSize** -- A pointer to the number of elements that will fit in the allocated stack buffer

BaseType_t **xTaskCallApplicationTaskHook**(*TaskHandle_t* xTask, void *pvParameter)

Calls the hook function associated with xTask. Passing xTask as NULL has the effect of calling the Running tasks (the calling task) hook function.

pvParameter is passed to the hook function for the task to interpret as it wants. The return value is the value returned by the task hook function registered by the user.

TaskHandle_t **xTaskGetIdleTaskHandle**(void)

xTaskGetIdleTaskHandle() is only available if INCLUDE_xTaskGetIdleTaskHandle is set to 1 in FreeRTOSConfig.h.

Simply returns the handle of the idle task of the current core. It is not valid to call xTaskGetIdleTaskHandle() before the scheduler has been started.

UBaseType_t **uxTaskGetSystemState**(*TaskStatus_t* *const pxTaskStatusArray, const UBaseType_t uxArraySize, configRUN_TIME_COUNTER_TYPE *const pulTotalRunTime)

configUSE_TRACE_FACILITY must be defined as 1 in FreeRTOSConfig.h for uxTaskGetSystemState() to be available.

uxTaskGetSystemState() populates an TaskStatus_t structure for each task in the system. TaskStatus_t structures contain, among other things, members for the task handle, task name, task priority, task state, and total amount of run time consumed by the task. See the TaskStatus_t structure definition in this file for the full member list.

NOTE: This function is intended for debugging use only as its use results in the scheduler remaining suspended for an extended period.

Example usage:

```
 // This example demonstrates how a human readable table of run time stats
 // information is generated from raw data provided by uxTaskGetSystemState().
 // The human readable table is written to pcWriteBuffer
 void vTaskGetRunTimeStats( char *pcWriteBuffer )
 {
 TaskStatus_t *pxTaskStatusArray;
 volatile UBaseType_t uxArraySize, x;
 configRUN_TIME_COUNTER_TYPE ulTotalRunTime, ulStatsAsPercentage;

// Make sure the write buffer does not contain a string.
pcWriteBuffer = 0x00;

// Take a snapshot of the number of tasks in case it changes while this
// function is executing.
     uxArraySize = uxTaskGetNumberOfTasks();

// Allocate a TaskStatus_t structure for each task.  An array could be
// allocated statically at compile time.
     pxTaskStatusArray = pvPortMalloc( uxArraySize * sizeof( TaskStatus_t ) );

if( pxTaskStatusArray != NULL )
     {
// Generate raw status information about each task.
         uxArraySize = uxTaskGetSystemState( pxTaskStatusArray, uxArraySize, &
↪ulTotalRunTime );

// For percentage calculations.
         ulTotalRunTime /= 100UL;
```

(下页继续)

(续上页)

```
// Avoid divide by zero errors.
if( ulTotalRunTime > 0 )
        {
// For each populated position in the pxTaskStatusArray array,
// format the raw data as human readable ASCII data
for( x = 0; x < uxArraySize; x++ )
            {
// What percentage of the total run time has the task used?
// This will always be rounded down to the nearest integer.
// ulTotalRunTimeDiv100 has already been divided by 100.
                ulStatsAsPercentage = pxTaskStatusArray[ x ].ulRunTimeCounter␣
↪/ ulTotalRunTime;

if( ulStatsAsPercentage > 0UL )
                {
                    sprintf( pcWriteBuffer, "%s\t\t%lu\t\t%lu%%\r\n",␣
↪pxTaskStatusArray[ x ].pcTaskName, pxTaskStatusArray[ x ].ulRunTimeCounter,␣
↪ulStatsAsPercentage );
                }
else
                {
// If the percentage is zero here then the task has
// consumed less than 1% of the total run time.
                    sprintf( pcWriteBuffer, "%s\t\t%lu\t\t<1%%\r\n",␣
↪pxTaskStatusArray[ x ].pcTaskName, pxTaskStatusArray[ x ].ulRunTimeCounter );
                }

                pcWriteBuffer += strlen( ( char * ) pcWriteBuffer );
            }
        }

// The array is no longer needed, free the memory it consumes.
        vPortFree( pxTaskStatusArray );
    }
}
```

参数

- **pxTaskStatusArray** -- A pointer to an array of TaskStatus_t structures. The array must contain at least one TaskStatus_t structure for each task that is under the control of the RTOS. The number of tasks under the control of the RTOS can be determined using the uxTaskGetNumberOfTasks() API function.
- **uxArraySize** -- The size of the array pointed to by the pxTaskStatusArray parameter. The size is specified as the number of indexes in the array, or the number of TaskStatus_t structures contained in the array, not by the number of bytes in the array.
- **pulTotalRunTime** -- If configGENERATE_RUN_TIME_STATS is set to 1 in FreeRTOSConfig.h then *pulTotalRunTime is set by uxTaskGetSystemState() to the total run time (as defined by the run time stats clock, see https://www.FreeRTOS.org/rtos-run-time-stats.html) since the target booted. pulTotalRunTime can be set to NULL to omit the total run time information.

返回 The number of TaskStatus_t structures that were populated by uxTaskGetSystemState(). This should equal the number returned by the uxTaskGetNumberOfTasks() API function, but will be zero if the value passed in the uxArraySize parameter was too small.

void **vTaskList**(char *pcWriteBuffer)

configUSE_TRACE_FACILITY and configUSE_STATS_FORMATTING_FUNCTIONS must both be defined as 1 for this function to be available. See the configuration section of the FreeRTOS.org website for more information.

NOTE 1: This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

Lists all the current tasks, along with their current state and stack usage high water mark.

Tasks are reported as blocked ('B'), ready ('R'), deleted ('D') or suspended ('S').

PLEASE NOTE:

This function is provided for convenience only, and is used by many of the demo applications. Do not consider it to be part of the scheduler.

vTaskList() calls uxTaskGetSystemState(), then formats part of the uxTaskGetSystemState() output into a human readable table that displays task: names, states, priority, stack usage and task number. Stack usage specified as the number of unused StackType_t words stack can hold on top of stack - not the number of bytes.

vTaskList() has a dependency on the sprintf() C library function that might bloat the code size, use a lot of stack, and provide different results on different platforms. An alternative, tiny, third party, and limited functionality implementation of sprintf() is provided in many of the FreeRTOS/Demo sub-directories in a file called printf-stdarg.c (note printf-stdarg.c does not provide a full snprintf() implementation!).

It is recommended that production systems call uxTaskGetSystemState() directly to get access to raw stats data, rather than indirectly through a call to vTaskList().

参数 `pcWriteBuffer` -- A buffer into which the above mentioned details will be written, in ASCII form. This buffer is assumed to be large enough to contain the generated report. Approximately 40 bytes per task should be sufficient.

void **`vTaskGetRunTimeStats`**(char *pcWriteBuffer)

configGENERATE_RUN_TIME_STATS and configUSE_STATS_FORMATTING_FUNCTIONS must both be defined as 1 for this function to be available. The application must also then provide definitions for portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() and portGET_RUN_TIME_COUNTER_VALUE() to configure a peripheral timer/counter and return the timers current count value respectively. The counter should be at least 10 times the frequency of the tick count.

NOTE 1: This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

Setting configGENERATE_RUN_TIME_STATS to 1 will result in a total accumulated execution time being stored for each task. The resolution of the accumulated time value depends on the frequency of the timer configured by the portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() macro. Calling vTaskGetRunTimeStats() writes the total execution time of each task into a buffer, both as an absolute count value and as a percentage of the total system execution time.

NOTE 2:

This function is provided for convenience only, and is used by many of the demo applications. Do not consider it to be part of the scheduler.

vTaskGetRunTimeStats() calls uxTaskGetSystemState(), then formats part of the uxTaskGetSystemState() output into a human readable table that displays the amount of time each task has spent in the Running state in both absolute and percentage terms.

vTaskGetRunTimeStats() has a dependency on the sprintf() C library function that might bloat the code size, use a lot of stack, and provide different results on different platforms. An alternative, tiny, third party, and limited functionality implementation of sprintf() is provided in many of the FreeRTOS/Demo sub-directories in a file called printf-stdarg.c (note printf-stdarg.c does not provide a full snprintf() implementation!).

It is recommended that production systems call uxTaskGetSystemState() directly to get access to raw stats data, rather than indirectly through a call to vTaskGetRunTimeStats().

参数 `pcWriteBuffer` -- A buffer into which the execution times will be written, in ASCII form. This buffer is assumed to be large enough to contain the generated report. Approximately 40 bytes per task should be sufficient.

configRUN_TIME_COUNTER_TYPE **`ulTaskGetIdleRunTimeCounter`**(void)

configGENERATE_RUN_TIME_STATS, configUSE_STATS_FORMATTING_FUNCTIONS and INCLUDE_xTaskGetIdleTaskHandle must all be defined as 1 for these functions to be available. The application must also then provide definitions for portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()

and portGET_RUN_TIME_COUNTER_VALUE() to configure a peripheral timer/counter and return the timers current count value respectively. The counter should be at least 10 times the frequency of the tick count.

Setting configGENERATE_RUN_TIME_STATS to 1 will result in a total accumulated execution time being stored for each task. The resolution of the accumulated time value depends on the frequency of the timer configured by the portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() macro. While uxTaskGetSystemState() and vTaskGetRunTimeStats() writes the total execution time of each task into a buffer, ulTaskGetIdleRunTimeCounter() returns the total execution time of just the idle task and ulTaskGetIdleRunTimePercent() returns the percentage of the CPU time used by just the idle task.

Note the amount of idle time is only a good measure of the slack time in a system if there are no other tasks executing at the idle priority, tickless idle is not used, and configIDLE_SHOULD_YIELD is set to 0.

备注： If configNUMBER_OF_CORES > 1, calling this function will query the idle task of the current core.

返回 The total run time of the idle task or the percentage of the total run time consumed by the idle task. This is the amount of time the idle task has actually been executing. The unit of time is dependent on the frequency configured using the portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() and portGET_RUN_TIME_COUNTER_VALUE() macros.

configRUN_TIME_COUNTER_TYPE **ulTaskGetIdleRunTimePercent**(void)

BaseType_t **xTaskGenericNotifyWait**(UBaseType_t uxIndexToWaitOn, uint32_t ulBitsToClearOnEntry, uint32_t ulBitsToClearOnExit, uint32_t *pulNotificationValue, TickType_t xTicksToWait)

Waits for a direct to task notification to be pending at a given index within an array of direct to task notifications.

See https://www.FreeRTOS.org/RTOS-task-notifications.html for details.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this function to be available.

Each task has a private array of "notification values" (or 'notifications'), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task's notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWaitIndexed() or ulTaskNotifyTakeIndexed() (or their un-indexed equivalents). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

A task can use xTaskNotifyWaitIndexed() to [optionally] block to wait for a notification to be pending, or ulTaskNotifyTakeIndexed() to [optionally] block to wait for a notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

NOTE Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single "notification value", and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotifyWait() is the original API function, and remains backward compatible by always

operating on the notification value at index 0 in the array. Calling xTaskNotifyWait() is equivalent to calling xTaskNotifyWaitIndexed() with the uxIndexToWaitOn parameter set to 0.

参数

- **uxIndexToWaitOn** -- The index within the calling task's array of notification values on which the calling task will wait for a notification to be received. uxIndexToWaitOn must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotifyWait() does not have this parameter and always waits for notifications on index 0.
- **ulBitsToClearOnEntry** -- Bits that are set in ulBitsToClearOnEntry value will be cleared in the calling task's notification value before the task checks to see if any notifications are pending, and optionally blocks if no notifications are pending. Setting ulBitsToClearOnEntry to ULONG_MAX (if limits.h is included) or 0xffffffffUL (if limits.h is not included) will have the effect of resetting the task's notification value to 0. Setting ulBitsToClearOnEntry to 0 will leave the task's notification value unchanged.
- **ulBitsToClearOnExit** -- If a notification is pending or received before the calling task exits the xTaskNotifyWait() function then the task's notification value (see the xTaskNotify() API function) is passed out using the pulNotificationValue parameter. Then any bits that are set in ulBitsToClearOnExit will be cleared in the task's notification value (note *pulNotificationValue is set before any bits are cleared). Setting ulBitsToClearOnExit to ULONG_MAX (if limits.h is included) or 0xffffffffUL (if limits.h is not included) will have the effect of resetting the task's notification value to 0 before the function exits. Setting ulBitsToClearOnExit to 0 will leave the task's notification value unchanged when the function exits (in which case the value passed out in pulNotificationValue will match the task's notification value).
- **pulNotificationValue** -- Used to pass the task's notification value out of the function. Note the value passed out will not be effected by the clearing of any bits caused by ulBitsToClearOnExit being non-zero.
- **xTicksToWait** -- The maximum amount of time that the task should wait in the Blocked state for a notification to be received, should a notification not already be pending when xTaskNotifyWait() was called. The task will not consume any processing time while it is in the Blocked state. This is specified in kernel ticks, the macro pdMS_TO_TICKS(value_in_ms) can be used to convert a time specified in milliseconds to a time specified in ticks.

返回 If a notification was received (including notifications that were already pending when xTaskNotifyWait was called) then pdPASS is returned. Otherwise pdFAIL is returned.

void **vTaskGenericNotifyGiveFromISR**(*TaskHandle_t* xTaskToNotify, UBaseType_t uxIndexToNotify, BaseType_t *pxHigherPriorityTaskWoken)

A version of xTaskNotifyGiveIndexed() that can be called from an interrupt service routine (ISR).

See https://www.FreeRTOS.org/RTOS-task-notifications.html for more details.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this macro to be available.

Each task has a private array of "notification values" (or 'notifications'), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task's notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

vTaskNotifyGiveIndexedFromISR() is intended for use when task notifications are used as light weight and faster binary or counting semaphore equivalents. Actual FreeRTOS semaphores are given from an ISR using the xSemaphoreGiveFromISR() API function, the equivalent action that instead uses a task notification is vTaskNotifyGiveIndexedFromISR().

When task notifications are being used as a binary or counting semaphore equivalent then the task being notified should wait for the notification using the ulTaskNotifyTakeIndexed() API function rather than the xTaskNotifyWaitIndexed() API function.

NOTE Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single "notification value", and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotifyFromISR() is the original API function, and remains backward compatible by always operating on the notification value at index 0 within the array. Calling xTaskNotifyGiveFromISR() is equivalent to calling xTaskNotifyGiveIndexedFromISR() with the uxIndexToNotify parameter set to 0.

参数

- **`xTaskToNotify`** -- The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- **`uxIndexToNotify`** -- The index within the target task's array of notification values to which the notification is to be sent. uxIndexToNotify must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotifyGiveFromISR() does not have this parameter and always sends notifications to index 0.
- **`pxHigherPriorityTaskWoken`** -- vTaskNotifyGiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending the notification caused the task to which the notification was sent to leave the Blocked state, and the unblocked task has a priority higher than the currently running task. If vTaskNotifyGiveFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited. How a context switch is requested from an ISR is dependent on the port - see the documentation page for the port in use.

BaseType_t **`xTaskGenericNotifyStateClear`**(*TaskHandle_t* xTask, UBaseType_t uxIndexToClear)

See https://www.FreeRTOS.org/RTOS-task-notifications.html for details.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for these functions to be available.

Each task has a private array of "notification values" (or 'notifications'), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

If a notification is sent to an index within the array of notifications then the notification at that index is said to be 'pending' until it is read or explicitly cleared by the receiving task. xTaskNotifyStateClearIndexed() is the function that clears a pending notification without reading the notification value. The notification value at the same array index is not altered. Set xTask to NULL to clear the notification state of the calling task.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single "notification value", and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotifyStateClear() is the original API function, and remains backward compatible by always operating on the notification value at index 0 within the array. Calling xTaskNotifyStateClear() is equivalent to calling xTaskNotifyStateClearIndexed() with the uxIndexToNotify parameter set to 0.

参数

- **`xTask`** -- The handle of the RTOS task that will have a notification state cleared. Set xTask to NULL to clear a notification state in the calling task. To obtain a task's handle create the task using xTaskCreate() and make use of the pxCreatedTask parameter, or create the task using xTaskCreateStatic() and store the returned value, or use the task's name in a call to xTaskGetHandle().
- **`uxIndexToClear`** -- The index within the target task's array of notification values to act upon. For example, setting uxIndexToClear to 1 will clear the state of the notification at index 1 within the array. uxIndexToClear must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. ulTaskNotifyStateClear() does not have

this parameter and always acts on the notification at index 0.

返回 pdTRUE if the task's notification state was set to eNotWaitingNotification, otherwise pdFALSE.

uint32_t **ulTaskGenericNotifyValueClear**(*TaskHandle_t* xTask, UBaseType_t uxIndexToClear, uint32_t ulBitsToClear)

See https://www.FreeRTOS.org/RTOS-task-notifications.html for details.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for these functions to be available.

Each task has a private array of "notification values" (or 'notifications'), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

ulTaskNotifyValueClearIndexed() clears the bits specified by the ulBitsToClear bit mask in the notification value at array index uxIndexToClear of the task referenced by xTask.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single "notification value", and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. ulTaskNotifyValueClear() is the original API function, and remains backward compatible by always operating on the notification value at index 0 within the array. Calling ulTaskNotifyValueClear() is equivalent to calling ulTaskNotifyValueClearIndexed() with the uxIndexToClear parameter set to 0.

参数

- **xTask** -- The handle of the RTOS task that will have bits in one of its notification values cleared. Set xTask to NULL to clear bits in a notification value of the calling task. To obtain a task's handle create the task using xTaskCreate() and make use of the pxCreatedTask parameter, or create the task using xTaskCreateStatic() and store the returned value, or use the task's name in a call to xTaskGetHandle().
- **uxIndexToClear** -- The index within the target task's array of notification values in which to clear the bits. uxIndexToClear must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. ulTaskNotifyValueClear() does not have this parameter and always clears bits in the notification value at index 0.
- **ulBitsToClear** -- Bit mask of the bits to clear in the notification value of xTask. Set a bit to 1 to clear the corresponding bits in the task's notification value. Set ulBitsToClear to 0xffffffff (UINT_MAX on 32-bit architectures) to clear the notification value to 0. Set ulBitsToClear to 0 to query the task's notification value without clearing any bits.

返回 The value of the target task's notification value before the bits specified by ulBitsToClear were cleared.

void **vTaskSetTimeOutState**(TimeOut_t *const pxTimeOut)

Capture the current time for future use with xTaskCheckForTimeOut().

参数 **pxTimeOut** -- Pointer to a timeout object into which the current time is to be captured. The captured time includes the tick count and the number of times the tick count has overflowed since the system first booted.

BaseType_t **xTaskCheckForTimeOut**(TimeOut_t *const pxTimeOut, TickType_t *const pxTicksToWait)

Determines if pxTicksToWait ticks has passed since a time was captured using a call to vTaskSetTimeOutState(). The captured time includes the tick count and the number of times the tick count has overflowed.

Example Usage:

```
// Driver library function used to receive uxWantedBytes from an Rx buffer
// that is filled by a UART interrupt. If there are not enough bytes in the
// Rx buffer then the task enters the Blocked state until it is notified that
// more data has been placed into the buffer. If there is still not enough
// data then the task re-enters the Blocked state, and xTaskCheckForTimeOut()
// is used to re-calculate the Block time to ensure the total amount of time
```

(下页继续)

(续上页)

```
// spent in the Blocked state does not exceed MAX_TIME_TO_WAIT. This
// continues until either the buffer contains at least uxWantedBytes bytes,
// or the total amount of time spent in the Blocked state reaches
// MAX_TIME_TO_WAIT - at which point the task reads however many bytes are
// available up to a maximum of uxWantedBytes.

size_t xUART_Receive( uint8_t *pucBuffer, size_t uxWantedBytes )
{
size_t uxReceived = 0;
TickType_t xTicksToWait = MAX_TIME_TO_WAIT;
TimeOut_t xTimeOut;

// Initialize xTimeOut.  This records the time at which this function
// was entered.
    vTaskSetTimeOutState( &xTimeOut );

// Loop until the buffer contains the wanted number of bytes, or a
// timeout occurs.
while( UART_bytes_in_rx_buffer( pxUARTInstance ) < uxWantedBytes )
    {
// The buffer didn't contain enough data so this task is going to
// enter the Blocked state. Adjusting xTicksToWait to account for
// any time that has been spent in the Blocked state within this
// function so far to ensure the total amount of time spent in the
// Blocked state does not exceed MAX_TIME_TO_WAIT.
if( xTaskCheckForTimeOut( &xTimeOut, &xTicksToWait ) != pdFALSE )
        {
//Timed out before the wanted number of bytes were available,
// exit the loop.
break;
        }

// Wait for a maximum of xTicksToWait ticks to be notified that the
// receive interrupt has placed more data into the buffer.
        ulTaskNotifyTake( pdTRUE, xTicksToWait );
    }

// Attempt to read uxWantedBytes from the receive buffer into pucBuffer.
// The actual number of bytes read (which might be less than
// uxWantedBytes) is returned.
    uxReceived = UART_read_from_receive_buffer( pxUARTInstance,
                                                pucBuffer,
                                                uxWantedBytes );

return uxReceived;
}
```

参见:

https://www.FreeRTOS.org/xTaskCheckForTimeOut.html

参数

- **pxTimeOut** -- The time status as captured previously using vTaskSetTimeOutState. If the timeout has not yet occurred, it is updated to reflect the current time status.
- **pxTicksToWait** -- The number of ticks to check for timeout i.e. if pxTicksToWait ticks have passed since pxTimeOut was last updated (either by vTaskSetTimeOutState() or xTaskCheckForTimeOut()), the timeout has occurred. If the timeout has not occurred, pxTicksToWait is updated to reflect the number of remaining ticks.

返回 If timeout has occurred, pdTRUE is returned. Otherwise pdFALSE is returned and pxTicksToWait is updated to reflect the number of remaining ticks.

BaseType_t **xTaskCatchUpTicks**(TickType_t xTicksToCatchUp)

This function corrects the tick count value after the application code has held interrupts disabled for an extended period resulting in tick interrupts having been missed.

This function is similar to vTaskStepTick(), however, unlike vTaskStepTick(), xTaskCatchUpTicks() may move the tick count forward past a time at which a task should be removed from the blocked state. That means tasks may have to be removed from the blocked state as the tick count is moved.

参数 **xTicksToCatchUp** -- The number of tick interrupts that have been missed due to interrupts being disabled. Its value is not computed automatically, so must be computed by the application writer.

返回 pdTRUE if moving the tick count forward resulted in a task leaving the blocked state and a context switch being performed. Otherwise pdFALSE.

Structures

struct **xTASK_STATUS**

Used with the uxTaskGetSystemState() function to return the state of each task in the system.

Public Members

TaskHandle_t **xHandle**

The handle of the task to which the rest of the information in the structure relates.

const char ***pcTaskName**

A pointer to the task's name. This value will be invalid if the task was deleted since the structure was populated!

UBaseType_t **xTaskNumber**

A number unique to the task.

eTaskState **eCurrentState**

The state in which the task existed when the structure was populated.

UBaseType_t **uxCurrentPriority**

The priority at which the task was running (may be inherited) when the structure was populated.

UBaseType_t **uxBasePriority**

The priority to which the task will return if the task's current priority has been inherited to avoid unbounded priority inversion when obtaining a mutex. Only valid if configUSE_MUTEXES is defined as 1 in FreeRTOSConfig.h.

configRUN_TIME_COUNTER_TYPE **ulRunTimeCounter**

The total run time allocated to the task so far, as defined by the run time stats clock. See https://www.FreeRTOS.org/rtos-run-time-stats.html. Only valid when configGENERATE_RUN_TIME_STATS is defined as 1 in FreeRTOSConfig.h.

StackType_t ***pxStackBase**

Points to the lowest address of the task's stack area.

configSTACK_DEPTH_TYPE **usStackHighWaterMark**

The minimum amount of stack space that has remained for the task since the task was created. The closer this value is to zero the closer the task has come to overflowing its stack.

BaseType_t **xCoreID**

Core this task is pinned to (0, 1, or tskNO_AFFINITY). If configNUMBER_OF_CORES == 1, this will always be 0.

Macros

tskIDLE_PRIORITY

Defines the priority used by the idle task. This must not be modified.

tskNO_AFFINITY

Macro representing and unpinned (i.e., "no affinity") task in xCoreID parameters

taskVALID_CORE_ID(xCoreID)

Macro to check if an xCoreID value is valid

返回 pdTRUE if valid, pdFALSE otherwise.

taskYIELD()

Macro for forcing a context switch.

taskENTER_CRITICAL(x)

Macro to mark the start of a critical code region. Preemptive context switches cannot occur when in a critical region.

NOTE: This may alter the stack (depending on the portable implementation) so must be used with care!

taskENTER_CRITICAL_FROM_ISR()

taskENTER_CRITICAL_ISR(x)

taskEXIT_CRITICAL(x)

Macro to mark the end of a critical code region. Preemptive context switches cannot occur when in a critical region.

NOTE: This may alter the stack (depending on the portable implementation) so must be used with care!

taskEXIT_CRITICAL_FROM_ISR(x)

taskEXIT_CRITICAL_ISR(x)

taskDISABLE_INTERRUPTS()

Macro to disable all maskable interrupts.

taskENABLE_INTERRUPTS()

Macro to enable microcontroller interrupts.

taskSCHEDULER_SUSPENDED

Definitions returned by xTaskGetSchedulerState(). taskSCHEDULER_SUSPENDED is 0 to generate more optimal code when configASSERT() is defined as the constant is used in assert() statements.

taskSCHEDULER_NOT_STARTED

taskSCHEDULER_RUNNING

xTaskNotifyIndexed(xTaskToNotify, uxIndexToNotify, ulValue, eAction)

See https://www.FreeRTOS.org/RTOS-task-notifications.html for details.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for these functions to be available.

Sends a direct to task notification to a task, with an optional value and action.

Each task has a private array of ”notification values” (or ’notifications’), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task’s notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

A task can use xTaskNotifyWaitIndexed() or ulTaskNotifyTakeIndexed() to [optionally] block to wait for a notification to be pending. The task does not consume any CPU time while it is in the Blocked state.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWaitIndexed() or ulTaskNotifyTakeIndexed() (or their un-indexed equivalents). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

NOTE Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single ”notification value”, and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotify() is the original API function, and remains backward compatible by always operating on the notification value at index 0 in the array. Calling xTaskNotify() is equivalent to calling xTaskNotifyIndexed() with the uxIndexToNotify parameter set to 0.

eSetBits - The target notification value is bitwise ORed with ulValue. xTaskNotifyIndexed() always returns pdPASS in this case.

eIncrement - The target notification value is incremented. ulValue is not used and xTaskNotifyIndexed() always returns pdPASS in this case.

eSetValueWithOverwrite - The target notification value is set to the value of ulValue, even if the task being notified had not yet processed the previous notification at the same array index (the task already had a notification pending at that index). xTaskNotifyIndexed() always returns pdPASS in this case.

eSetValueWithoutOverwrite - If the task being notified did not already have a notification pending at the same array index then the target notification value is set to ulValue and xTaskNotifyIndexed() will return pdPASS. If the task being notified already had a notification pending at the same array index then no action is performed and pdFAIL is returned.

eNoAction - The task receives a notification at the specified array index without the notification value at that index being updated. ulValue is not used and xTaskNotifyIndexed() always returns pdPASS in this case.

pulPreviousNotificationValue - Can be used to pass out the subject task’s notification value before any bits are modified by the notify function.

参数

- **`xTaskToNotify`** -- The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- **`uxIndexToNotify`** -- The index within the target task’s array of notification values to which the notification is to be sent. uxIndexToNotify must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotify() does not have this parameter and always sends notifications to index 0.
- **`ulValue`** -- Data that can be sent with the notification. How the data is used depends on the value of the eAction parameter.

- **eAction** -- Specifies how the notification updates the task's notification value, if at all. Valid values for eAction are as follows:

返回 Dependent on the value of eAction. See the description of the eAction parameter.

xTaskNotifyAndQueryIndexed(xTaskToNotify, uxIndexToNotify, ulValue, eAction, pulPreviousNotifyValue)

See https://www.FreeRTOS.org/RTOS-task-notifications.html for details.

xTaskNotifyAndQueryIndexed() performs the same operation as xTaskNotifyIndexed() with the addition that it also returns the subject task's prior notification value (the notification value at the time the function is called rather than when the function returns) in the additional pulPreviousNotifyValue parameter.

xTaskNotifyAndQuery() performs the same operation as xTaskNotify() with the addition that it also returns the subject task's prior notification value (the notification value as it was at the time the function is called, rather than when the function returns) in the additional pulPreviousNotifyValue parameter.

xTaskNotifyIndexedFromISR(xTaskToNotify, uxIndexToNotify, ulValue, eAction, pxHigherPriorityTaskWoken)

See https://www.FreeRTOS.org/RTOS-task-notifications.html for details.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for these functions to be available.

A version of xTaskNotifyIndexed() that can be used from an interrupt service routine (ISR).

Each task has a private array of "notification values" (or 'notifications'), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task's notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

A task can use xTaskNotifyWaitIndexed() to [optionally] block to wait for a notification to be pending, or ulTaskNotifyTakeIndexed() to [optionally] block to wait for a notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWaitIndexed() or ulTaskNotifyTakeIndexed() (or their un-indexed equivalents). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

NOTE Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single "notification value", and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotifyFromISR() is the original API function, and remains backward compatible by always operating on the notification value at index 0 within the array. Calling xTaskNotifyFromISR() is equivalent to calling xTaskNotifyIndexedFromISR() with the uxIndexToNotify parameter set to 0.

eSetBits - The task's notification value is bitwise ORed with ulValue. xTaskNotify() always returns pdPASS in this case.

eIncrement - The task's notification value is incremented. ulValue is not used and xTaskNotify() always returns pdPASS in this case.

eSetValueWithOverwrite - The task's notification value is set to the value of ulValue, even if the task being notified had not yet processed the previous notification (the task already had a notification pending). xTaskNotify() always returns pdPASS in this case.

eSetValueWithoutOverwrite - If the task being notified did not already have a notification pending then the task's notification value is set to ulValue and xTaskNotify() will return pdPASS. If the task being notified already had a notification pending then no action is performed and pdFAIL is returned.

eNoAction - The task receives a notification without its notification value being updated. ulValue is not used and xTaskNotify() always returns pdPASS in this case.

参数

- **uxIndexToNotify** -- The index within the target task's array of notification values to which the notification is to be sent. uxIndexToNotify must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotifyFromISR() does not have this parameter and always sends notifications to index 0.
- **xTaskToNotify** -- The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- **ulValue** -- Data that can be sent with the notification. How the data is used depends on the value of the eAction parameter.
- **eAction** -- Specifies how the notification updates the task's notification value, if at all. Valid values for eAction are as follows:
- **pxHigherPriorityTaskWoken** -- xTaskNotifyFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending the notification caused the task to which the notification was sent to leave the Blocked state, and the unblocked task has a priority higher than the currently running task. If xTaskNotifyFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited. How a context switch is requested from an ISR is dependent on the port - see the documentation page for the port in use.

返回 Dependent on the value of eAction. See the description of the eAction parameter.

xTaskNotifyAndQueryIndexedFromISR(xTaskToNotify, uxIndexToNotify, ulValue, eAction, pulPreviousNotificationValue, pxHigherPriorityTaskWoken)

See https://www.FreeRTOS.org/RTOS-task-notifications.html for details.

xTaskNotifyAndQueryIndexedFromISR() performs the same operation as xTaskNotifyIndexedFromISR() with the addition that it also returns the subject task's prior notification value (the notification value at the time the function is called rather than at the time the function returns) in the additional pulPreviousNotifyValue parameter.

xTaskNotifyAndQueryFromISR() performs the same operation as xTaskNotifyFromISR() with the addition that it also returns the subject task's prior notification value (the notification value at the time the function is called rather than at the time the function returns) in the additional pulPreviousNotifyValue parameter.

xTaskNotifyWait(ulBitsToClearOnEntry, ulBitsToClearOnExit, pulNotificationValue, xTicksToWait)

xTaskNotifyWaitIndexed(uxIndexToWaitOn, ulBitsToClearOnEntry, ulBitsToClearOnExit, pulNotificationValue, xTicksToWait)

xTaskNotifyGiveIndexed(xTaskToNotify, uxIndexToNotify)

Sends a direct to task notification to a particular index in the target task's notification array in a manner similar to giving a counting semaphore.

See https://www.FreeRTOS.org/RTOS-task-notifications.html for more details.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for these macros to be available.

Each task has a private array of "notification values" (or 'notifications'), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task's notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

xTaskNotifyGiveIndexed() is a helper macro intended for use when task notifications are used as light weight and faster binary or counting semaphore equivalents. Actual FreeRTOS semaphores are given using the xSemaphoreGive() API function, the equivalent action that instead uses a task notification is xTaskNotifyGiveIndexed().

When task notifications are being used as a binary or counting semaphore equivalent then the task being notified should wait for the notification using the ulTaskNotifyTakeIndexed() API function rather than the xTaskNotifyWaitIndexed() API function.

NOTE Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single "notification value", and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotifyGive() is the original API function, and remains backward compatible by always operating on the notification value at index 0 in the array. Calling xTaskNotifyGive() is equivalent to calling xTaskNotifyGiveIndexed() with the uxIndexToNotify parameter set to 0.

参数

- **`xTaskToNotify`** -- The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- **`uxIndexToNotify`** -- The index within the target task's array of notification values to which the notification is to be sent. uxIndexToNotify must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotifyGive() does not have this parameter and always sends notifications to index 0.

返回 xTaskNotifyGive() is a macro that calls xTaskNotify() with the eAction parameter set to eIncrement - so pdPASS is always returned.

`vTaskNotifyGiveFromISR`(xTaskToNotify, pxHigherPriorityTaskWoken)

`vTaskNotifyGiveIndexedFromISR`(xTaskToNotify, uxIndexToNotify, pxHigherPriorityTaskWoken)

`ulTaskNotifyTakeIndexed`(uxIndexToWaitOn, xClearCountOnExit, xTicksToWait)

Waits for a direct to task notification on a particular index in the calling task's notification array in a manner similar to taking a counting semaphore.

See https://www.FreeRTOS.org/RTOS-task-notifications.html for details.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this function to be available.

Each task has a private array of "notification values" (or 'notifications'), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task's notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

ulTaskNotifyTakeIndexed() is intended for use when a task notification is used as a faster and lighter weight binary or counting semaphore alternative. Actual FreeRTOS semaphores are taken using the xSemaphoreTake() API function, the equivalent action that instead uses a task notification is ulTaskNotifyTakeIndexed().

When a task is using its notification value as a binary or counting semaphore other tasks should send notifications to it using the xTaskNotifyGiveIndexed() macro, or xTaskNotifyIndex() function with the eAction parameter set to eIncrement.

ulTaskNotifyTakeIndexed() can either clear the task's notification value at the array index specified by the uxIndexToWaitOn parameter to zero on exit, in which case the notification value acts like a binary semaphore, or decrement the notification value on exit, in which case the notification value acts like a counting semaphore.

A task can use ulTaskNotifyTakeIndexed() to [optionally] block to wait for a notification. The task does not consume any CPU time while it is in the Blocked state.

Where as xTaskNotifyWaitIndexed() will return when a notification is pending, ulTaskNotifyTakeIndexed() will return when the task's notification value is not zero.

NOTE Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single "notification value", and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. ulTaskNotifyTake() is the original API function, and remains backward compatible by always operating on the notification value at index 0 in the array. Calling ulTaskNotifyTake() is equivalent to calling ulTaskNotifyTakeIndexed() with the uxIndexToWaitOn parameter set to 0.

参数

- **uxIndexToWaitOn** -- The index within the calling task's array of notification values on which the calling task will wait for a notification to be non-zero. uxIndexToWaitOn must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotifyTake() does not have this parameter and always waits for notifications on index 0.
- **xClearCountOnExit** -- if xClearCountOnExit is pdFALSE then the task's notification value is decremented when the function exits. In this way the notification value acts like a counting semaphore. If xClearCountOnExit is not pdFALSE then the task's notification value is cleared to zero when the function exits. In this way the notification value acts like a binary semaphore.
- **xTicksToWait** -- The maximum amount of time that the task should wait in the Blocked state for the task's notification value to be greater than zero, should the count not already be greater than zero when ulTaskNotifyTake() was called. The task will not consume any processing time while it is in the Blocked state. This is specified in kernel ticks, the macro pdMS_TO_TICKS(value_in_ms) can be used to convert a time specified in milliseconds to a time specified in ticks.

返回 The task's notification count before it is either cleared to zero or decremented (see the xClearCountOnExit parameter).

xTaskNotifyStateClear(xTask)

xTaskNotifyStateClearIndexed(xTask, uxIndexToClear)

ulTaskNotifyValueClear(xTask, ulBitsToClear)

ulTaskNotifyValueClearIndexed(xTask, uxIndexToClear, ulBitsToClear)

Type Definitions

typedef struct tskTaskControlBlock ***TaskHandle_t**

typedef BaseType_t (***TaskHookFunction_t**)(void*)

Defines the prototype to which the application task hook function must conform.

typedef struct *xTASK_STATUS* **TaskStatus_t**

Used with the uxTaskGetSystemState() function to return the state of each task in the system.

Enumerations

enum **eTaskState**

Task states returned by eTaskGetState.

Values:

enumerator **eRunning**

A task is querying the state of itself, so must be running.

enumerator **eReady**

The task being queried is in a ready or pending ready list.

enumerator **eBlocked**

The task being queried is in the Blocked state.

enumerator **eSuspended**

The task being queried is in the Suspended state, or is in the Blocked state with an infinite time out.

enumerator **eDeleted**

The task being queried has been deleted, but its TCB has not yet been freed.

enumerator **eInvalid**

Used as an 'invalid state' value.

enum **eNotifyAction**

Actions that can be performed when vTaskNotify() is called.

Values:

enumerator **eNoAction**

Notify the task without updating its notify value.

enumerator **eSetBits**

Set bits in the task's notification value.

enumerator **eIncrement**

Increment the task's notification value.

enumerator **eSetValueWithOverwrite**

Set the task's notification value to a specific value even if the previous value has not yet been read by the task.

enumerator **eSetValueWithoutOverwrite**

Set the task's notification value if the previous value has been read by the task.

enum **eSleepModeStatus**

Possible return values for eTaskConfirmSleepModeStatus().

Values:

enumerator **eAbortSleep**

A task has been made ready or a context switch pended since portSUPPRESS_TICKS_AND_SLEEP() was called - abort entering a sleep mode.

enumerator **eStandardSleep**

Enter a sleep mode that will not last any longer than the expected idle time.

队列 API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/queue.h
- This header file can be included with:

```
#include "freertos/queue.h"
```

Functions

BaseType_t **xQueueGenericSend**(*QueueHandle_t* xQueue, const void *const pvItemToQueue, TickType_t xTicksToWait, const BaseType_t xCopyPosition)

It is preferred that the macros xQueueSend(), xQueueSendToFront() and xQueueSendToBack() are used in place of calling this function directly.

Post an item on a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in an ISR.

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;

 // Create a queue capable of containing 10 uint32_t values.
 xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

 // Create a queue capable of containing 10 pointers to AMessage structures.
 // These should be passed by pointer as they contain a lot of data.
 xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

 // ...

 if( xQueue1 != 0 )
```

(下页继续)

(续上页)

```
 {
// Send an uint32_t.  Wait for 10 ticks for space to become
// available if necessary.
if( xQueueGenericSend( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10,␣
↪queueSEND_TO_BACK ) != pdPASS )
     {
// Failed to post the message, even after 10 ticks.
     }
 }

 if( xQueue2 != 0 )
 {
// Send a pointer to a struct AMessage object.  Don't block if the
// queue is already full.
     pxMessage = & xMessage;
     xQueueGenericSend( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0,␣
↪queueSEND_TO_BACK );
 }

 // ... Rest of task code.
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **xTicksToWait** -- The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.
- **xCopyPosition** -- Can take the value queueSEND_TO_BACK to place the item at the back of the queue, or queueSEND_TO_FRONT to place the item at the front of the queue (for high priority messages).

返回 pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

BaseType_t **xQueuePeek**(*QueueHandle_t* xQueue, void *const pvBuffer, TickType_t xTicksToWait)

Receive an item from a queue without removing the item from the queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items remain on the queue so will be returned again by the next call, or a call to xQueueReceive().

This macro must not be used in an interrupt service routine. See xQueuePeekFromISR() for an alternative that can be called from an interrupt service routine.

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
} xMessage;

QueueHandle_t xQueue;
```

(下页继续)

(续上页)

```
// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
struct AMessage *pxMessage;

 // Create a queue capable of containing 10 pointers to AMessage structures.
 // These should be passed by pointer as they contain a lot of data.
 xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
 if( xQueue == 0 )
 {
// Failed to create the queue.
 }

 // ...

 // Send a pointer to a struct AMessage object.  Don't block if the
 // queue is already full.
 pxMessage = & xMessage;
 xQueueSend( xQueue, ( void * ) &pxMessage, ( TickType_t ) 0 );

 // ... Rest of task code.
}

// Task to peek the data from the queue.
void vADifferentTask( void *pvParameters )
{
struct AMessage *pxRxedMessage;

 if( xQueue != 0 )
 {
// Peek a message on the created queue.  Block for 10 ticks if a
// message is not immediately available.
if( xQueuePeek( xQueue, &( pxRxedMessage ), ( TickType_t ) 10 ) )
     {
// pcRxedMessage now points to the struct AMessage variable posted
// by vATask, but the item still remains on the queue.
     }
 }

 // ... Rest of task code.
}
```

参数

- **xQueue** -- The handle to the queue from which the item is to be received.
- **pvBuffer** -- Pointer to the buffer into which the received item will be copied.
- **xTicksToWait** -- The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required. xQueuePeek() will return immediately if xTicksToWait is 0 and the queue is empty.

返回 pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

BaseType_t **xQueuePeekFromISR** (*QueueHandle_t* xQueue, void *const pvBuffer)

A version of xQueuePeek() that can be called from an interrupt service routine (ISR).

Receive an item from a queue without removing the item from the queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items remain on the queue so will be returned again by the next call, or a call to xQueueReceive().

参数

- **xQueue** -- The handle to the queue from which the item is to be received.
- **pvBuffer** -- Pointer to the buffer into which the received item will be copied.

返回 pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

BaseType_t **xQueueReceive**(*QueueHandle_t* xQueue, void *const pvBuffer, TickType_t xTicksToWait)

Receive an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items are removed from the queue.

This function must not be used in an interrupt service routine. See xQueueReceiveFromISR for an alternative that can.

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
} xMessage;

QueueHandle_t xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
struct AMessage *pxMessage;

 // Create a queue capable of containing 10 pointers to AMessage structures.
 // These should be passed by pointer as they contain a lot of data.
 xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
 if( xQueue == 0 )
 {
// Failed to create the queue.
 }

 // ...

 // Send a pointer to a struct AMessage object.  Don't block if the
 // queue is already full.
 pxMessage = & xMessage;
 xQueueSend( xQueue, ( void * ) &pxMessage, ( TickType_t ) 0 );

 // ... Rest of task code.
}

// Task to receive from the queue.
void vADifferentTask( void *pvParameters )
{
struct AMessage *pxRxedMessage;

 if( xQueue != 0 )
 {
// Receive a message on the created queue.  Block for 10 ticks if a
// message is not immediately available.
if( xQueueReceive( xQueue, &( pxRxedMessage ), ( TickType_t ) 10 ) )
     {
// pcRxedMessage now points to the struct AMessage variable posted
// by vATask.
     }
 }
```

(下页继续)

(续上页)

```
 // ... Rest of task code.
}
```

参数

- **xQueue** -- The handle to the queue from which the item is to be received.
- **pvBuffer** -- Pointer to the buffer into which the received item will be copied.
- **xTicksToWait** -- The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. xQueueReceive() will return immediately if xTicksToWait is zero and the queue is empty. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

返回 pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

UBaseType_t **uxQueueMessagesWaiting**(const *QueueHandle_t* xQueue)

Return the number of messages stored in a queue.

参数 **xQueue** -- A handle to the queue being queried.
返回 The number of messages available in the queue.

UBaseType_t **uxQueueSpacesAvailable**(const *QueueHandle_t* xQueue)

Return the number of free spaces available in a queue. This is equal to the number of items that can be sent to the queue before the queue becomes full if no items are removed.

参数 **xQueue** -- A handle to the queue being queried.
返回 The number of spaces available in the queue.

void **vQueueDelete**(*QueueHandle_t* xQueue)

Delete a queue - freeing all the memory allocated for storing of items placed on the queue.

参数 **xQueue** -- A handle to the queue to be deleted.

BaseType_t **xQueueGenericSendFromISR**(*QueueHandle_t* xQueue, const void *const pvItemToQueue, BaseType_t *const pxHigherPriorityTaskWoken, const BaseType_t xCopyPosition)

It is preferred that the macros xQueueSendFromISR(), xQueueSendToFrontFromISR() and xQueueSendToBackFromISR() be used in place of calling this function directly. xQueueGiveFromISR() is an equivalent for use by semaphores that don't actually copy any data.

Post an item on a queue. It is safe to use this function from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
char cIn;
BaseType_t xHigherPriorityTaskWokenByPost;

 // We have not woken a task at the start of the ISR.
 xHigherPriorityTaskWokenByPost = pdFALSE;

 // Loop until the buffer is empty.
 do
 {
// Obtain a byte from the buffer.
     cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );
```

(下页继续)

(续上页)

```
// Post each byte.
     xQueueGenericSendFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWokenByPost,
↪ queueSEND_TO_BACK );

 } while( portINPUT_BYTE( BUFFER_COUNT ) );

 // Now the buffer is empty we can switch context if necessary.  Note that the
 // name of the yield function required is port specific.
 if( xHigherPriorityTaskWokenByPost )
 {
     portYIELD_FROM_ISR();
 }
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** -- xQueueGenericSendFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueGenericSendFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.
- **xCopyPosition** -- Can take the value queueSEND_TO_BACK to place the item at the back of the queue, or queueSEND_TO_FRONT to place the item at the front of the queue (for high priority messages).

返回 pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

BaseType_t **xQueueGiveFromISR**(*QueueHandle_t* xQueue, BaseType_t *const pxHigherPriorityTaskWoken)

BaseType_t **xQueueReceiveFromISR**(*QueueHandle_t* xQueue, void *const pvBuffer, BaseType_t *const pxHigherPriorityTaskWoken)

Receive an item from a queue. It is safe to use this function from within an interrupt service routine.

Example usage:

```
QueueHandle_t xQueue;

// Function to create a queue and post some values.
void vAFunction( void *pvParameters )
{
char cValueToPost;
const TickType_t xTicksToWait = ( TickType_t )0xff;

 // Create a queue capable of containing 10 characters.
 xQueue = xQueueCreate( 10, sizeof( char ) );
 if( xQueue == 0 )
 {
// Failed to create the queue.
 }

 // ...

 // Post some characters that will be used within an ISR.  If the queue
 // is full then this task will block for xTicksToWait ticks.
```

(下页继续)

(续上页)

```
 cValueToPost = 'a';
 xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );
 cValueToPost = 'b';
 xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );

 // ... keep posting characters ... this task may block when the queue
 // becomes full.

 cValueToPost = 'c';
 xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );
}

// ISR that outputs all the characters received on the queue.
void vISR_Routine( void )
{
BaseType_t xTaskWokenByReceive = pdFALSE;
char cRxedChar;

 while( xQueueReceiveFromISR( xQueue, ( void * ) &cRxedChar, &
↪xTaskWokenByReceive) )
 {
// A character was received.  Output the character now.
     vOutputCharacter( cRxedChar );

// If removing the character from the queue woke the task that was
// posting onto the queue xTaskWokenByReceive will have been set to
// pdTRUE.  No matter how many times this loop iterates only one
// task will be woken.
 }

 if( xTaskWokenByReceive != ( char ) pdFALSE;
 {
     taskYIELD ();
 }
}
```

参数

- **xQueue** -- The handle to the queue from which the item is to be received.
- **pvBuffer** -- Pointer to the buffer into which the received item will be copied.
- **pxHigherPriorityTaskWoken** -- A task may be blocked waiting for space to become available on the queue. If xQueueReceiveFromISR causes such a task to unblock *pxTaskWoken will get set to pdTRUE, otherwise *pxTaskWoken will remain unchanged.

返回 pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

BaseType_t **xQueueIsQueueEmptyFromISR** (const *QueueHandle_t* xQueue)

Queries a queue to determine if the queue is empty. This function should only be used in an ISR.

参数 **xQueue** -- The handle of the queue being queried

返回 pdFALSE if the queue is not empty, or pdTRUE if the queue is empty.

BaseType_t **xQueueIsQueueFullFromISR** (const *QueueHandle_t* xQueue)

Queries a queue to determine if the queue is full. This function should only be used in an ISR.

参数 **xQueue** -- The handle of the queue being queried

返回 pdFALSE if the queue is not full, or pdTRUE if the queue is full.

UBaseType_t **uxQueueMessagesWaitingFromISR** (const *QueueHandle_t* xQueue)

A version of uxQueueMessagesWaiting() that can be called from an ISR. Return the number of messages stored in a queue.

参数 **xQueue** -- A handle to the queue being queried.

返回 The number of messages available in the queue.

void **vQueueAddToRegistry**(*QueueHandle_t* xQueue, const char *pcQueueName)

The registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call vQueueAddToRegistry() add a queue, semaphore or mutex handle to the registry if you want the handle to be available to a kernel aware debugger. If you are not using a kernel aware debugger then this function can be ignored.

configQUEUE_REGISTRY_SIZE defines the maximum number of handles the registry can hold. configQUEUE_REGISTRY_SIZE must be greater than 0 within FreeRTOSConfig.h for the registry to be available. Its value does not affect the number of queues, semaphores and mutexes that can be created - just the number that the registry can hold.

If vQueueAddToRegistry is called more than once with the same xQueue parameter, the registry will store the pcQueueName parameter from the most recent call to vQueueAddToRegistry.

参数

- **xQueue** -- The handle of the queue being added to the registry. This is the handle returned by a call to xQueueCreate(). Semaphore and mutex handles can also be passed in here.
- **pcQueueName** -- The name to be associated with the handle. This is the name that the kernel aware debugger will display. The queue registry only stores a pointer to the string - so the string must be persistent (global or preferably in ROM/Flash), not on the stack.

void **vQueueUnregisterQueue**(*QueueHandle_t* xQueue)

The registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call vQueueAddToRegistry() add a queue, semaphore or mutex handle to the registry if you want the handle to be available to a kernel aware debugger, and vQueueUnregisterQueue() to remove the queue, semaphore or mutex from the register. If you are not using a kernel aware debugger then this function can be ignored.

参数 **xQueue** -- The handle of the queue being removed from the registry.

const char ***pcQueueGetName**(*QueueHandle_t* xQueue)

The queue registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call pcQueueGetName() to look up and return the name of a queue in the queue registry from the queue's handle.

参数 **xQueue** -- The handle of the queue the name of which will be returned.

返回 If the queue is in the registry then a pointer to the name of the queue is returned. If the queue is not in the registry then NULL is returned.

QueueSetHandle_t **xQueueCreateSet**(const UBaseType_t uxEventQueueLength)

Queue sets provide a mechanism to allow a task to block (pend) on a read operation from multiple queues or semaphores simultaneously.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

A queue set must be explicitly created using a call to xQueueCreateSet() before it can be used. Once created, standard FreeRTOS queues and semaphores can be added to the set using calls to xQueueAddToSet(). xQueueSelectFromSet() is then used to determine which, if any, of the queues or semaphores contained in the set is in a state where a queue read or semaphore take operation would be successful.

Note 1: See the documentation on https://www.FreeRTOS.org/RTOS-queue-sets.html for reasons why queue sets are very rarely needed in practice as there are simpler methods of blocking on multiple objects.

Note 2: Blocking on a queue set that contains a mutex will not cause the mutex holder to inherit the priority of the blocked task.

Note 3: An additional 4 bytes of RAM is required for each space in a every queue added to a queue set. Therefore counting semaphores that have a high maximum count value should not be added to a queue set.

Note 4: A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

参数 **uxEventQueueLength** -- Queue sets store events that occur on the queues and semaphores contained in the set. uxEventQueueLength specifies the maximum number of events that can be queued at once. To be absolutely certain that events are not lost uxEventQueueLength should be set to the total sum of the length of the queues added to the set, where binary semaphores and mutexes have a length of 1, and counting semaphores have a length set by their maximum count value. Examples:

- If a queue set is to hold a queue of length 5, another queue of length 12, and a binary semaphore, then uxEventQueueLength should be set to (5 + 12 + 1), or 18.
- If a queue set is to hold three binary semaphores then uxEventQueueLength should be set to (1 + 1 + 1), or 3.
- If a queue set is to hold a counting semaphore that has a maximum count of 5, and a counting semaphore that has a maximum count of 3, then uxEventQueueLength should be set to (5 + 3), or 8.

返回 If the queue set is created successfully then a handle to the created queue set is returned. Otherwise NULL is returned.

BaseType_t **xQueueAddToSet**(*QueueSetMemberHandle_t* xQueueOrSemaphore, *QueueSetHandle_t* xQueueSet)

Adds a queue or semaphore to a queue set that was previously created by a call to xQueueCreateSet().

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

Note 1: A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

参数

- **xQueueOrSemaphore** -- The handle of the queue or semaphore being added to the queue set (cast to an QueueSetMemberHandle_t type).
- **xQueueSet** -- The handle of the queue set to which the queue or semaphore is being added.

返回 If the queue or semaphore was successfully added to the queue set then pdPASS is returned. If the queue could not be successfully added to the queue set because it is already a member of a different queue set then pdFAIL is returned.

BaseType_t **xQueueRemoveFromSet**(*QueueSetMemberHandle_t* xQueueOrSemaphore, *QueueSetHandle_t* xQueueSet)

Removes a queue or semaphore from a queue set. A queue or semaphore can only be removed from a set if the queue or semaphore is empty.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

参数

- **xQueueOrSemaphore** -- The handle of the queue or semaphore being removed from the queue set (cast to an QueueSetMemberHandle_t type).
- **xQueueSet** -- The handle of the queue set in which the queue or semaphore is included.

返回 If the queue or semaphore was successfully removed from the queue set then pdPASS is returned. If the queue was not in the queue set, or the queue (or semaphore) was not empty, then pdFAIL is returned.

QueueSetMemberHandle_t **xQueueSelectFromSet**(*QueueSetHandle_t* xQueueSet, const TickType_t xTicksToWait)

xQueueSelectFromSet() selects from the members of a queue set a queue or semaphore that either contains data (in the case of a queue) or is available to take (in the case of a semaphore). xQueueSelectFromSet() effectively allows a task to block (pend) on a read operation on all the queues and semaphores in a queue set simultaneously.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

Note 1: See the documentation on https://www.FreeRTOS.org/RTOS-queue-sets.html for reasons why queue sets are very rarely needed in practice as there are simpler methods of blocking on multiple objects.

Note 2: Blocking on a queue set that contains a mutex will not cause the mutex holder to inherit the priority of the blocked task.

Note 3: A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

参数

- **xQueueSet** -- The queue set on which the task will (potentially) block.
- **xTicksToWait** -- The maximum time, in ticks, that the calling task will remain in the Blocked state (with other tasks executing) to wait for a member of the queue set to be ready for a successful queue read or semaphore take operation.

返回 xQueueSelectFromSet() will return the handle of a queue (cast to a QueueSetMemberHandle_t type) contained in the queue set that contains data, or the handle of a semaphore (cast to a QueueSetMemberHandle_t type) contained in the queue set that is available, or NULL if no such queue or semaphore exists before before the specified block time expires.

QueueSetMemberHandle_t **xQueueSelectFromSetFromISR**(*QueueSetHandle_t* xQueueSet)

A version of xQueueSelectFromSet() that can be used from an ISR.

Macros

xQueueCreate(uxQueueLength, uxItemSize)

Creates a new queue instance, and returns a handle by which the new queue can be referenced.

Internally, within the FreeRTOS implementation, queues use two blocks of memory. The first block is used to hold the queue's data structures. The second block is used to hold items placed into the queue. If a queue is created using xQueueCreate() then both blocks of memory are automatically dynamically allocated inside the xQueueCreate() function. (see https://www.FreeRTOS.org/a00111.html). If a queue is created using xQueueCreateStatic() then the application writer must provide the memory that will get used by the queue. xQueueCreateStatic() therefore allows a queue to be created without using any dynamic memory allocation.

https://www.FreeRTOS.org/Embedded-RTOS-Queues.html

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
};

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;

 // Create a queue capable of containing 10 uint32_t values.
 xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );
 if( xQueue1 == 0 )
 {
// Queue was not created and must not be used.
 }

 // Create a queue capable of containing 10 pointers to AMessage structures.
 // These should be passed by pointer as they contain a lot of data.
 xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );
 if( xQueue2 == 0 )
 {
// Queue was not created and must not be used.
 }
```

(下页继续)

(续上页)

```
 // ... Rest of task code.
}
```

参数

- **uxQueueLength** -- The maximum number of items that the queue can contain.
- **uxItemSize** -- The number of bytes each item in the queue will require. Items are queued by copy, not by reference, so this is the number of bytes that will be copied for each posted item. Each item on the queue must be the same size.

返回 If the queue is successfully create then a handle to the newly created queue is returned. If the queue cannot be created then 0 is returned.

xQueueCreateStatic(uxQueueLength, uxItemSize, pucQueueStorage, pxQueueBuffer)

Creates a new queue instance, and returns a handle by which the new queue can be referenced.

Internally, within the FreeRTOS implementation, queues use two blocks of memory. The first block is used to hold the queue's data structures. The second block is used to hold items placed into the queue. If a queue is created using xQueueCreate() then both blocks of memory are automatically dynamically allocated inside the xQueueCreate() function. (see https://www.FreeRTOS.org/a00111.html). If a queue is created using xQueueCreateStatic() then the application writer must provide the memory that will get used by the queue. xQueueCreateStatic() therefore allows a queue to be created without using any dynamic memory allocation.

https://www.FreeRTOS.org/Embedded-RTOS-Queues.html

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
};

#define QUEUE_LENGTH 10
#define ITEM_SIZE sizeof( uint32_t )

// xQueueBuffer will hold the queue structure.
StaticQueue_t xQueueBuffer;

// ucQueueStorage will hold the items posted to the queue.  Must be at least
// [(queue length) * ( queue item size)] bytes long.
uint8_t ucQueueStorage[ QUEUE_LENGTH * ITEM_SIZE ];

void vATask( void *pvParameters )
{
 QueueHandle_t xQueue1;

 // Create a queue capable of containing 10 uint32_t values.
 xQueue1 = xQueueCreate( QUEUE_LENGTH, // The number of items the queue can␣
↪hold.
                         ITEM_SIZE     // The size of each item in the queue
                         &( ucQueueStorage[ 0 ] ), // The buffer that will␣
↪hold the items in the queue.
                         &xQueueBuffer ); // The buffer that will hold the␣
↪queue structure.

 // The queue is guaranteed to be created successfully as no dynamic memory
 // allocation is used.  Therefore xQueue1 is now a handle to a valid queue.

 // ... Rest of task code.
}
```

参数

- **uxQueueLength** -- The maximum number of items that the queue can contain.
- **uxItemSize** -- The number of bytes each item in the queue will require. Items are queued by copy, not by reference, so this is the number of bytes that will be copied for each posted item. Each item on the queue must be the same size.
- **pucQueueStorage** -- If uxItemSize is not zero then pucQueueStorage must point to a uint8_t array that is at least large enough to hold the maximum number of items that can be in the queue at any one time - which is (uxQueueLength * uxItemsSize) bytes. If uxItemSize is zero then pucQueueStorage can be NULL.
- **pxQueueBuffer** -- Must point to a variable of type StaticQueue_t, which will be used to hold the queue's data structure.

返回 If the queue is created then a handle to the created queue is returned. If pxQueueBuffer is NULL then NULL is returned.

xQueueGetStaticBuffers(xQueue, ppucQueueStorage, ppxStaticQueue)

Retrieve pointers to a statically created queue's data structure buffer and storage area buffer. These are the same buffers that are supplied at the time of creation.

参数

- **xQueue** -- The queue for which to retrieve the buffers.
- **ppucQueueStorage** -- Used to return a pointer to the queue's storage area buffer.
- **ppxStaticQueue** -- Used to return a pointer to the queue's data structure buffer.

返回 pdTRUE if buffers were retrieved, pdFALSE otherwise.

xQueueSendToFront(xQueue, pvItemToQueue, xTicksToWait)

Post an item to the front of a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in an ISR.

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;

 // Create a queue capable of containing 10 uint32_t values.
 xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

 // Create a queue capable of containing 10 pointers to AMessage structures.
 // These should be passed by pointer as they contain a lot of data.
 xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

 // ...

 if( xQueue1 != 0 )
 {
// Send an uint32_t.  Wait for 10 ticks for space to become
// available if necessary.
if( xQueueSendToFront( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) !=␣
↪pdPASS )
```

(下页继续)

(续上页)

```
    {
// Failed to post the message, even after 10 ticks.
    }
 }

 if( xQueue2 != 0 )
 {
// Send a pointer to a struct AMessage object.  Don't block if the
// queue is already full.
    pxMessage = & xMessage;
    xQueueSendToFront( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0 );
 }

 // ... Rest of task code.
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **xTicksToWait** -- The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

返回 pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

xQueueSendToBack(xQueue, pvItemToQueue, xTicksToWait)

This is a macro that calls xQueueGenericSend().

Post an item to the back of a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in an ISR.

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;

 // Create a queue capable of containing 10 uint32_t values.
 xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

 // Create a queue capable of containing 10 pointers to AMessage structures.
 // These should be passed by pointer as they contain a lot of data.
 xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

 // ...
```

(下页继续)

(续上页)

```
 if( xQueue1 != 0 )
 {
// Send an uint32_t.  Wait for 10 ticks for space to become
// available if necessary.
if( xQueueSendToBack( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) !=␣
↪pdPASS )
     {
// Failed to post the message, even after 10 ticks.
     }
 }

 if( xQueue2 != 0 )
 {
// Send a pointer to a struct AMessage object.  Don't block if the
// queue is already full.
     pxMessage = & xMessage;
     xQueueSendToBack( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0 );
 }

 // ... Rest of task code.
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **xTicksToWait** -- The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

返回 pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

xQueueSend(xQueue, pvItemToQueue, xTicksToWait)

This is a macro that calls xQueueGenericSend(). It is included for backward compatibility with versions of FreeRTOS.org that did not include the xQueueSendToFront() and xQueueSendToBack() macros. It is equivalent to xQueueSendToBack().

Post an item on a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in an ISR.

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;
```

(下页继续)

(续上页)

```
 // Create a queue capable of containing 10 uint32_t values.
 xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

 // Create a queue capable of containing 10 pointers to AMessage structures.
 // These should be passed by pointer as they contain a lot of data.
 xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

 // ...

 if( xQueue1 != 0 )
 {
// Send an uint32_t.  Wait for 10 ticks for space to become
// available if necessary.
if( xQueueSend( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) != pdPASS )
     {
// Failed to post the message, even after 10 ticks.
     }
 }

 if( xQueue2 != 0 )
 {
// Send a pointer to a struct AMessage object.  Don't block if the
// queue is already full.
     pxMessage = & xMessage;
     xQueueSend( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0 );
 }

 // ... Rest of task code.
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **xTicksToWait** -- The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

返回 pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

xQueueOverwrite(xQueue, pvItemToQueue)

Only for use with queues that have a length of one - so the queue is either empty or full.

Post an item on a queue. If the queue is already full then overwrite the value held in the queue. The item is queued by copy, not by reference.

This function must not be called from an interrupt service routine. See xQueueOverwriteFromISR () for an alternative which may be used in an ISR.

Example usage:

```
void vFunction( void *pvParameters )
{
QueueHandle_t xQueue;
uint32_t ulVarToSend, ulValReceived;
```

(下页继续)

(续上页)

```
 // Create a queue to hold one uint32_t value.  It is strongly
 // recommended *not* to use xQueueOverwrite() on queues that can
 // contain more than one value, and doing so will trigger an assertion
 // if configASSERT() is defined.
 xQueue = xQueueCreate( 1, sizeof( uint32_t ) );

 // Write the value 10 to the queue using xQueueOverwrite().
 ulVarToSend = 10;
 xQueueOverwrite( xQueue, &ulVarToSend );

 // Peeking the queue should now return 10, but leave the value 10 in
 // the queue.  A block time of zero is used as it is known that the
 // queue holds a value.
 ulValReceived = 0;
 xQueuePeek( xQueue, &ulValReceived, 0 );

 if( ulValReceived != 10 )
 {
// Error unless the item was removed by a different task.
 }

 // The queue is still full.  Use xQueueOverwrite() to overwrite the
 // value held in the queue with 100.
 ulVarToSend = 100;
 xQueueOverwrite( xQueue, &ulVarToSend );

 // This time read from the queue, leaving the queue empty once more.
 // A block time of 0 is used again.
 xQueueReceive( xQueue, &ulValReceived, 0 );

 // The value read should be the last value written, even though the
 // queue was already full when the value was written.
 if( ulValReceived != 100 )
 {
// Error!
 }

 // ...
}
```

参数

- **xQueue** -- The handle of the queue to which the data is being sent.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.

返回 xQueueOverwrite() is a macro that calls xQueueGenericSend(), and therefore has the same return values as xQueueSendToFront(). However, pdPASS is the only value that can be returned because xQueueOverwrite() will write to the queue even when the queue is already full.

xQueueSendToFrontFromISR (xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

This is a macro that calls xQueueGenericSendFromISR().

Post an item to the front of a queue. It is safe to use this macro from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
char cIn;
BaseType_t xHigherPriorityTaskWoken;

 // We have not woken a task at the start of the ISR.
 xHigherPriorityTaskWoken = pdFALSE;

 // Loop until the buffer is empty.
 do
 {
// Obtain a byte from the buffer.
     cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

// Post the byte.
     xQueueSendToFrontFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

 } while( portINPUT_BYTE( BUFFER_COUNT ) );

 // Now the buffer is empty we can switch context if necessary.
 if( xHigherPriorityTaskWoken )
 {
     taskYIELD ();
 }
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** -- xQueueSendToFrontFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendToFromFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

返回 pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

xQueueSendToBackFromISR(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

This is a macro that calls xQueueGenericSendFromISR().

Post an item to the back of a queue. It is safe to use this macro from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
char cIn;
BaseType_t xHigherPriorityTaskWoken;

 // We have not woken a task at the start of the ISR.
 xHigherPriorityTaskWoken = pdFALSE;

 // Loop until the buffer is empty.
 do
 {
```

(下页继续)

(续上页)

```
// Obtain a byte from the buffer.
     cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

// Post the byte.
     xQueueSendToBackFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

 } while( portINPUT_BYTE( BUFFER_COUNT ) );

 // Now the buffer is empty we can switch context if necessary.
 if( xHigherPriorityTaskWoken )
 {
     taskYIELD ();
 }
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** -- xQueueSendToBackFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendToBackFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

返回 pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

xQueueOverwriteFromISR(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

A version of xQueueOverwrite() that can be used in an interrupt service routine (ISR).

Only for use with queues that can hold a single item - so the queue is either empty or full.

Post an item on a queue. If the queue is already full then overwrite the value held in the queue. The item is queued by copy, not by reference.

Example usage:

```
QueueHandle_t xQueue;

void vFunction( void *pvParameters )
{
 // Create a queue to hold one uint32_t value.  It is strongly
 // recommended *not* to use xQueueOverwriteFromISR() on queues that can
 // contain more than one value, and doing so will trigger an assertion
 // if configASSERT() is defined.
 xQueue = xQueueCreate( 1, sizeof( uint32_t ) );
}

void vAnInterruptHandler( void )
{
// xHigherPriorityTaskWoken must be set to pdFALSE before it is used.
BaseType_t xHigherPriorityTaskWoken = pdFALSE;
uint32_t ulVarToSend, ulValReceived;

 // Write the value 10 to the queue using xQueueOverwriteFromISR().
 ulVarToSend = 10;
 xQueueOverwriteFromISR( xQueue, &ulVarToSend, &xHigherPriorityTaskWoken );
```

(下页继续)

(续上页)

```
 // The queue is full, but calling xQueueOverwriteFromISR() again will still
 // pass because the value held in the queue will be overwritten with the
 // new value.
 ulVarToSend = 100;
 xQueueOverwriteFromISR( xQueue, &ulVarToSend, &xHigherPriorityTaskWoken );

 // Reading from the queue will now return 100.

 // ...

 if( xHigherPritoritaskWoken == pdTRUE )
 {
// Writing to the queue caused a task to unblock and the unblocked task
// has a priority higher than or equal to the priority of the currently
// executing task (the task this interrupt interrupted).  Perform a context
// switch so this interrupt returns directly to the unblocked task.
     portYIELD_FROM_ISR(); // or portEND_SWITCHING_ISR() depending on the port.
 }
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** -- xQueueOverwriteFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueOverwriteFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

返回 xQueueOverwriteFromISR() is a macro that calls xQueueGenericSendFromISR(), and therefore has the same return values as xQueueSendToFrontFromISR(). However, pdPASS is the only value that can be returned because xQueueOverwriteFromISR() will write to the queue even when the queue is already full.

xQueueSendFromISR(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

This is a macro that calls xQueueGenericSendFromISR(). It is included for backward compatibility with versions of FreeRTOS.org that did not include the xQueueSendToBackFromISR() and xQueueSendToFrontFromISR() macros.

Post an item to the back of a queue. It is safe to use this function from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
char cIn;
BaseType_t xHigherPriorityTaskWoken;

 // We have not woken a task at the start of the ISR.
 xHigherPriorityTaskWoken = pdFALSE;

 // Loop until the buffer is empty.
 do
 {
```

(下页继续)

(续上页)

```
// Obtain a byte from the buffer.
     cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

// Post the byte.
     xQueueSendFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

 } while( portINPUT_BYTE( BUFFER_COUNT ) );

 // Now the buffer is empty we can switch context if necessary.
 if( xHigherPriorityTaskWoken )
 {
// Actual macro used here is port specific.
     portYIELD_FROM_ISR ();
 }
}
```

参数

- **xQueue** -- The handle to the queue on which the item is to be posted.
- **pvItemToQueue** -- A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** -- xQueueSendFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

返回 pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

xQueueReset(xQueue)

Reset a queue back to its original empty state. The return value is now obsolete and is always set to pdPASS.

Type Definitions

typedef struct QueueDefinition ***QueueHandle_t**

typedef struct QueueDefinition ***QueueSetHandle_t**

Type by which queue sets are referenced. For example, a call to xQueueCreateSet() returns an xQueueSet variable that can then be used as a parameter to xQueueSelectFromSet(), xQueueAddToSet(), etc.

typedef struct QueueDefinition ***QueueSetMemberHandle_t**

Queue sets can contain both queues and semaphores, so the QueueSetMemberHandle_t is defined as a type to be used where a parameter or return value can be either an QueueHandle_t or an SemaphoreHandle_t.

信号量 API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/semphr.h
- This header file can be included with:

```
#include "freertos/semphr.h"
```

Macros

semBINARY_SEMAPHORE_QUEUE_LENGTH

semSEMAPHORE_QUEUE_ITEM_LENGTH

semGIVE_BLOCK_TIME

vSemaphoreCreateBinary(xSemaphore)

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a binary semaphore! https://www.FreeRTOS.org/RTOS-task-notifications.html

This old vSemaphoreCreateBinary() macro is now deprecated in favour of the xSemaphoreCreateBinary() function. Note that binary semaphores created using the vSemaphoreCreateBinary() macro are created in a state such that the first call to 'take' the semaphore would pass, whereas binary semaphores created using xSemaphoreCreateBinary() are created in a state such that the the semaphore must first be 'given' before it can be 'taken'.

Macro that implements a semaphore by using the existing queue mechanism. The queue length is 1 as this is a binary semaphore. The data size is 0 as we don't want to actually store any data - we just want to know if the queue is empty or full.

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously 'give' the semaphore while another continuously 'takes' the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see xSemaphoreCreateMutex().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters )
{
 // Semaphore cannot be used before a call to vSemaphoreCreateBinary ().
 // This is a macro so pass the variable in directly.
 vSemaphoreCreateBinary( xSemaphore );

 if( xSemaphore != NULL )
 {
// The semaphore was created successfully.
// The semaphore can now be used.
 }
}
```

参数

- **xSemaphore** -- Handle to the created semaphore. Should be of type SemaphoreHandle_t.

xSemaphoreCreateBinary()

Creates a new binary semaphore instance, and returns a handle by which the new semaphore can be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a binary semaphore! https://www.FreeRTOS.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, binary semaphores use a block of memory, in which the semaphore structure is stored. If a binary semaphore is created using xSemaphoreCreateBinary() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateBinary() function. (see https://www.FreeRTOS.org/a00111.html). If a binary semaphore is created using xSemaphoreCreateBinaryStatic() then the application writer must provide the memory. xSemaphoreCreateBinaryStatic() therefore allows a binary semaphore to be created without using any dynamic memory allocation.

The old vSemaphoreCreateBinary() macro is now deprecated in favour of this xSemaphoreCreateBinary() function. Note that binary semaphores created using the vSemaphoreCreateBinary() macro are created in a state such that the first call to 'take' the semaphore would pass, whereas binary semaphores created using xSemaphoreCreateBinary() are created in a state such that the the semaphore must first be 'given' before it can be 'taken'.

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously 'give' the semaphore while another continuously 'takes' the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see xSemaphoreCreateMutex().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters )
{
 // Semaphore cannot be used before a call to xSemaphoreCreateBinary().
 // This is a macro so pass the variable in directly.
 xSemaphore = xSemaphoreCreateBinary();

 if( xSemaphore != NULL )
 {
// The semaphore was created successfully.
// The semaphore can now be used.
 }
}
```

返回 Handle to the created semaphore, or NULL if the memory required to hold the semaphore's data structures could not be allocated.

xSemaphoreCreateBinaryStatic(pxStaticSemaphore)

Creates a new binary semaphore instance, and returns a handle by which the new semaphore can be referenced.

NOTE: In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a binary semaphore! https://www.FreeRTOS.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, binary semaphores use a block of memory, in which the semaphore structure is stored. If a binary semaphore is created using xSemaphoreCreateBinary() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateBinary() function. (see https://www.FreeRTOS.org/a00111.html). If a binary semaphore is created using xSemaphoreCreateBinaryStatic() then the application writer must provide the memory. xSemaphoreCreateBinaryStatic() therefore allows a binary semaphore to be created without using any dynamic memory allocation.

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously 'give' the semaphore while another continuously 'takes' the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see xSemaphoreCreateMutex().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters )
{
 // Semaphore cannot be used before a call to xSemaphoreCreateBinary().
```

(下页继续)

(续上页)

```
 // The semaphore's data structures will be placed in the xSemaphoreBuffer
 // variable, the address of which is passed into the function.  The
 // function's parameter is not NULL, so the function will not attempt any
 // dynamic memory allocation, and therefore the function will not return
 // return NULL.
 xSemaphore = xSemaphoreCreateBinary( &xSemaphoreBuffer );

 // Rest of task code goes here.
}
```

参数

- **pxStaticSemaphore** -- Must point to a variable of type StaticSemaphore_t, which will then be used to hold the semaphore's data structure, removing the need for the memory to be allocated dynamically.

返回 If the semaphore is created then a handle to the created semaphore is returned. If pxSemaphoreBuffer is NULL then NULL is returned.

xSemaphoreTake(xSemaphore, xBlockTime)

Macro to obtain a semaphore. The semaphore must have previously been created with a call to xSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;

// A task that creates a semaphore.
void vATask( void * pvParameters )
{
 // Create the semaphore to guard a shared resource.
 xSemaphore = xSemaphoreCreateBinary();
}

// A task that uses the semaphore.
void vAnotherTask( void * pvParameters )
{
 // ... Do other things.

 if( xSemaphore != NULL )
 {
// See if we can obtain the semaphore.  If the semaphore is not available
// wait 10 ticks to see if it becomes free.
if( xSemaphoreTake( xSemaphore, ( TickType_t ) 10 ) == pdTRUE )
     {
// We were able to obtain the semaphore and can now access the
// shared resource.

// ...

// We have finished accessing the shared resource.  Release the
// semaphore.
          xSemaphoreGive( xSemaphore );
     }
else
     {
// We could not obtain the semaphore and can therefore not access
// the shared resource safely.
     }
 }
}
```

参数

- **xSemaphore** -- A handle to the semaphore being taken - obtained when the semaphore was created.
- **xBlockTime** -- The time in ticks to wait for the semaphore to become available. The macro portTICK_PERIOD_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. A block time of portMAX_DELAY can be used to block indefinitely (provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h).

返回 pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the semaphore becoming available.

xSemaphoreTakeRecursive(xMutex, xBlockTime)

Macro to recursively obtain, or 'take', a mutex type semaphore. The mutex must have previously been created using a call to xSemaphoreCreateRecursiveMutex();

configUSE_RECURSIVE_MUTEXES must be set to 1 in FreeRTOSConfig.h for this macro to be available.

This macro must not be used on mutexes created using xSemaphoreCreateMutex().

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called xSemaphoreGiveRecursive() for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

Example usage:

```
SemaphoreHandle_t xMutex = NULL;

// A task that creates a mutex.
void vATask( void * pvParameters )
{
 // Create the mutex to guard a shared resource.
 xMutex = xSemaphoreCreateRecursiveMutex();
}

// A task that uses the mutex.
void vAnotherTask( void * pvParameters )
{
 // ... Do other things.

 if( xMutex != NULL )
 {
// See if we can obtain the mutex.  If the mutex is not available
// wait 10 ticks to see if it becomes free.
if( xSemaphoreTakeRecursive( xSemaphore, ( TickType_t ) 10 ) == pdTRUE )
     {
// We were able to obtain the mutex and can now access the
// shared resource.

// ...
// For some reason due to the nature of the code further calls to
// xSemaphoreTakeRecursive() are made on the same mutex.  In real
// code these would not be just sequential calls as this would make
// no sense.  Instead the calls are likely to be buried inside
// a more complex call structure.
         xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );
         xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );

// The mutex has now been 'taken' three times, so will not be
// available to another task until it has also been given back
// three times.  Again it is unlikely that real code would have
// these calls sequentially, but instead buried in a more complex
```

(下页继续)

(续上页)

```
// call structure.  This is just for illustrative purposes.
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );

// Now the mutex can be taken by other tasks.
    }
else
    {
// We could not obtain the mutex and can therefore not access
// the shared resource safely.
    }
 }
}
```

参数

- **xMutex** -- A handle to the mutex being obtained. This is the handle returned by xSemaphoreCreateRecursiveMutex();
- **xBlockTime** -- The time in ticks to wait for the semaphore to become available. The macro portTICK_PERIOD_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. If the task already owns the semaphore then xSemaphoreTakeRecursive() will return immediately no matter what the value of xBlockTime.

返回 pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the semaphore becoming available.

xSemaphoreGive(xSemaphore)

Macro to release a semaphore. The semaphore must have previously been created with a call to xSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting(). and obtained using sSemaphoreTake().

This macro must not be used from an ISR. See xSemaphoreGiveFromISR () for an alternative which can be used from an ISR.

This macro must also not be used on semaphores created using xSemaphoreCreateRecursiveMutex().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters )
{
 // Create the semaphore to guard a shared resource.
 xSemaphore = vSemaphoreCreateBinary();

 if( xSemaphore != NULL )
 {
if( xSemaphoreGive( xSemaphore ) != pdTRUE )
    {
// We would expect this call to fail because we cannot give
// a semaphore without first "taking" it!
    }

// Obtain the semaphore - don't block if the semaphore is not
// immediately available.
if( xSemaphoreTake( xSemaphore, ( TickType_t ) 0 ) )
    {
// We now have the semaphore and can access the shared resource.
```

(下页继续)

(续上页)

```
// ...

// We have finished accessing the shared resource so can free the
// semaphore.
if( xSemaphoreGive( xSemaphore ) != pdTRUE )
        {
// We would not expect this call to fail because we must have
// obtained the semaphore to get here.
        }
    }
 }
}
```

参数

- **xSemaphore** -- A handle to the semaphore being released. This is the handle returned when the semaphore was created.

返回 pdTRUE if the semaphore was released. pdFALSE if an error occurred. Semaphores are implemented using queues. An error can occur if there is no space on the queue to post a message - indicating that the semaphore was not first obtained correctly.

xSemaphoreGiveRecursive(xMutex)

Macro to recursively release, or 'give', a mutex type semaphore. The mutex must have previously been created using a call to xSemaphoreCreateRecursiveMutex();

configUSE_RECURSIVE_MUTEXES must be set to 1 in FreeRTOSConfig.h for this macro to be available.

This macro must not be used on mutexes created using xSemaphoreCreateMutex().

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called xSemaphoreGiveRecursive() for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

Example usage:

```
SemaphoreHandle_t xMutex = NULL;

// A task that creates a mutex.
void vATask( void * pvParameters )
{
 // Create the mutex to guard a shared resource.
 xMutex = xSemaphoreCreateRecursiveMutex();
}

// A task that uses the mutex.
void vAnotherTask( void * pvParameters )
{
 // ... Do other things.

 if( xMutex != NULL )
 {
// See if we can obtain the mutex.  If the mutex is not available
// wait 10 ticks to see if it becomes free.
if( xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 ) == pdTRUE )
     {
// We were able to obtain the mutex and can now access the
// shared resource.
```

(下页继续)

(续上页)

```
// ...
// For some reason due to the nature of the code further calls to
// xSemaphoreTakeRecursive() are made on the same mutex.  In real
// code these would not be just sequential calls as this would make
// no sense.  Instead the calls are likely to be buried inside
// a more complex call structure.
        xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );
        xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );

// The mutex has now been 'taken' three times, so will not be
// available to another task until it has also been given back
// three times.  Again it is unlikely that real code would have
// these calls sequentially, it would be more likely that the calls
// to xSemaphoreGiveRecursive() would be called as a call stack
// unwound.  This is just for demonstrative purposes.
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );

// Now the mutex can be taken by other tasks.
    }
else
    {
// We could not obtain the mutex and can therefore not access
// the shared resource safely.
    }
 }
}
```

参数

- **xMutex** -- A handle to the mutex being released, or 'given'. This is the handle returned by xSemaphoreCreateMutex();

返回 pdTRUE if the semaphore was given.

xSemaphoreGiveFromISR (xSemaphore, pxHigherPriorityTaskWoken)

Macro to release a semaphore. The semaphore must have previously been created with a call to xSemaphoreCreateBinary() or xSemaphoreCreateCounting().

Mutex type semaphores (those created using a call to xSemaphoreCreateMutex()) must not be used with this macro.

This macro can be used from an ISR.

Example usage:

```
#define LONG_TIME 0xffff
#define TICKS_TO_WAIT 10
SemaphoreHandle_t xSemaphore = NULL;

// Repetitive task.
void vATask( void * pvParameters )
{
 for( ;; )
 {
// We want this task to run every 10 ticks of a timer.  The semaphore
// was created before this task was started.

// Block waiting for the semaphore to become available.
if( xSemaphoreTake( xSemaphore, LONG_TIME ) == pdTRUE )
```

(下页继续)

(续上页)

```
    {
// It is time to execute.

// ...

// We have finished our task.  Return to the top of the loop where
// we will block on the semaphore until it is time to execute
// again.  Note when using the semaphore for synchronisation with an
// ISR in this manner there is no need to 'give' the semaphore back.
    }
 }
}

// Timer ISR
void vTimerISR( void * pvParameters )
{
static uint8_t ucLocalTickCount = 0;
static BaseType_t xHigherPriorityTaskWoken;

 // A timer tick has occurred.

 // ... Do other time functions.

 // Is it time for vATask () to run?
 xHigherPriorityTaskWoken = pdFALSE;
 ucLocalTickCount++;
 if( ucLocalTickCount >= TICKS_TO_WAIT )
 {
// Unblock the task by releasing the semaphore.
    xSemaphoreGiveFromISR( xSemaphore, &xHigherPriorityTaskWoken );

// Reset the count so we release the semaphore again in 10 ticks time.
    ucLocalTickCount = 0;
 }

 if( xHigherPriorityTaskWoken != pdFALSE )
 {
// We can force a context switch here.  Context switching from an
// ISR uses port specific syntax.  Check the demo task for your port
// to find the syntax required.
 }
}
```

参数

- **xSemaphore** -- A handle to the semaphore being released. This is the handle returned when the semaphore was created.
- **pxHigherPriorityTaskWoken** -- xSemaphoreGiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if giving the semaphore caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xSemaphoreGiveFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

返回 pdTRUE if the semaphore was successfully given, otherwise errQUEUE_FULL.

xSemaphoreTakeFromISR (xSemaphore, pxHigherPriorityTaskWoken)

Macro to take a semaphore from an ISR. The semaphore must have previously been created with a call to xSemaphoreCreateBinary() or xSemaphoreCreateCounting().

Mutex type semaphores (those created using a call to xSemaphoreCreateMutex()) must not be used with this macro.

This macro can be used from an ISR, however taking a semaphore from an ISR is not a common operation. It

is likely to only be useful when taking a counting semaphore when an interrupt is obtaining an object from a resource pool (when the semaphore count indicates the number of resources available).

参数

- **xSemaphore** -- A handle to the semaphore being taken. This is the handle returned when the semaphore was created.
- **pxHigherPriorityTaskWoken** -- xSemaphoreTakeFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if taking the semaphore caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xSemaphoreTakeFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

返回 pdTRUE if the semaphore was successfully taken, otherwise pdFALSE

xSemaphoreCreateMutex()

Creates a new mutex type semaphore instance, and returns a handle by which the new mutex can be referenced.

Internally, within the FreeRTOS implementation, mutex semaphores use a block of memory, in which the mutex structure is stored. If a mutex is created using xSemaphoreCreateMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateMutex() function. (see https://www.FreeRTOS.org/a00111.html). If a mutex is created using xSemaphoreCreateMutexStatic() then the application writer must provided the memory. xSemaphoreCreateMutexStatic() therefore allows a mutex to be created without using any dynamic memory allocation.

Mutexes created using this function can be accessed using the xSemaphoreTake() and xSemaphoreGive() macros. The xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros must not be used.

This type of semaphore uses a priority inheritance mechanism so a task 'taking' a semaphore MUST ALWAYS 'give' the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See xSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always 'gives' the semaphore and another always 'takes' the semaphore) and from within interrupt service routines.

Example usage:

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
 // Semaphore cannot be used before a call to xSemaphoreCreateMutex().
 // This is a macro so pass the variable in directly.
 xSemaphore = xSemaphoreCreateMutex();

 if( xSemaphore != NULL )
 {
// The semaphore was created successfully.
// The semaphore can now be used.
 }
}
```

返回 If the mutex was successfully created then a handle to the created semaphore is returned. If there was not enough heap to allocate the mutex data structures then NULL is returned.

xSemaphoreCreateMutexStatic(pxMutexBuffer)

Creates a new mutex type semaphore instance, and returns a handle by which the new mutex can be referenced.

Internally, within the FreeRTOS implementation, mutex semaphores use a block of memory, in which the mutex structure is stored. If a mutex is created using xSemaphoreCreateMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateMutex() function. (see https:

//www.FreeRTOS.org/a00111.html). If a mutex is created using xSemaphoreCreateMutexStatic() then the application writer must provided the memory. xSemaphoreCreateMutexStatic() therefore allows a mutex to be created without using any dynamic memory allocation.

Mutexes created using this function can be accessed using the xSemaphoreTake() and xSemaphoreGive() macros. The xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros must not be used.

This type of semaphore uses a priority inheritance mechanism so a task 'taking' a semaphore MUST ALWAYS 'give' the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See xSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always 'gives' the semaphore and another always 'takes' the semaphore) and from within interrupt service routines.

Example usage:

```
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xMutexBuffer;

void vATask( void * pvParameters )
{
 // A mutex cannot be used before it has been created.  xMutexBuffer is
 // into xSemaphoreCreateMutexStatic() so no dynamic memory allocation is
 // attempted.
 xSemaphore = xSemaphoreCreateMutexStatic( &xMutexBuffer );

 // As no dynamic memory allocation was performed, xSemaphore cannot be NULL,
 // so there is no need to check it.
}
```

参数

- **pxMutexBuffer** -- Must point to a variable of type StaticSemaphore_t, which will be used to hold the mutex's data structure, removing the need for the memory to be allocated dynamically.

返回 If the mutex was successfully created then a handle to the created mutex is returned. If pxMutexBuffer was NULL then NULL is returned.

xSemaphoreCreateRecursiveMutex()

Creates a new recursive mutex type semaphore instance, and returns a handle by which the new recursive mutex can be referenced.

Internally, within the FreeRTOS implementation, recursive mutexes use a block of memory, in which the mutex structure is stored. If a recursive mutex is created using xSemaphoreCreateRecursiveMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateRecursiveMutex() function. (see https://www.FreeRTOS.org/a00111.html). If a recursive mutex is created using xSemaphoreCreateRecursiveMutexStatic() then the application writer must provide the memory that will get used by the mutex. xSemaphoreCreateRecursiveMutexStatic() therefore allows a recursive mutex to be created without using any dynamic memory allocation.

Mutexes created using this macro can be accessed using the xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros. The xSemaphoreTake() and xSemaphoreGive() macros must not be used.

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called xSemaphoreGiveRecursive() for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

This type of semaphore uses a priority inheritance mechanism so a task 'taking' a semaphore MUST ALWAYS 'give' the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See xSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always 'gives' the semaphore and another always 'takes' the semaphore) and from within interrupt service routines.

Example usage:

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
 // Semaphore cannot be used before a call to xSemaphoreCreateMutex().
 // This is a macro so pass the variable in directly.
 xSemaphore = xSemaphoreCreateRecursiveMutex();

 if( xSemaphore != NULL )
 {
// The semaphore was created successfully.
// The semaphore can now be used.
 }
}
```

返回 xSemaphore Handle to the created mutex semaphore. Should be of type SemaphoreHandle_t.

xSemaphoreCreateRecursiveMutexStatic(pxStaticSemaphore)

Creates a new recursive mutex type semaphore instance, and returns a handle by which the new recursive mutex can be referenced.

Internally, within the FreeRTOS implementation, recursive mutexes use a block of memory, in which the mutex structure is stored. If a recursive mutex is created using xSemaphoreCreateRecursiveMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateRecursiveMutex() function. (see https://www.FreeRTOS.org/a00111.html). If a recursive mutex is created using xSemaphoreCreateRecursiveMutexStatic() then the application writer must provide the memory that will get used by the mutex. xSemaphoreCreateRecursiveMutexStatic() therefore allows a recursive mutex to be created without using any dynamic memory allocation.

Mutexes created using this macro can be accessed using the xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros. The xSemaphoreTake() and xSemaphoreGive() macros must not be used.

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called xSemaphoreGiveRecursive() for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

This type of semaphore uses a priority inheritance mechanism so a task 'taking' a semaphore MUST ALWAYS 'give' the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See xSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always 'gives' the semaphore and another always 'takes' the semaphore) and from within interrupt service routines.

Example usage:

```
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xMutexBuffer;
```

(下页继续)

(续上页)

```
void vATask( void * pvParameters )
{
 // A recursive semaphore cannot be used before it is created.  Here a
 // recursive mutex is created using xSemaphoreCreateRecursiveMutexStatic().
 // The address of xMutexBuffer is passed into the function, and will hold
 // the mutexes data structures - so no dynamic memory allocation will be
 // attempted.
 xSemaphore = xSemaphoreCreateRecursiveMutexStatic( &xMutexBuffer );

 // As no dynamic memory allocation was performed, xSemaphore cannot be NULL,
 // so there is no need to check it.
}
```

参数

- **pxStaticSemaphore** -- Must point to a variable of type StaticSemaphore_t, which will then be used to hold the recursive mutex's data structure, removing the need for the memory to be allocated dynamically.

返回 If the recursive mutex was successfully created then a handle to the created recursive mutex is returned. If pxStaticSemaphore was NULL then NULL is returned.

xSemaphoreCreateCounting(uxMaxCount, uxInitialCount)

Creates a new counting semaphore instance, and returns a handle by which the new counting semaphore can be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a counting semaphore! https://www.FreeRTOS.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, counting semaphores use a block of memory, in which the counting semaphore structure is stored. If a counting semaphore is created using xSemaphoreCreateCounting() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateCounting() function. (see https://www.FreeRTOS.org/a00111.html). If a counting semaphore is created using xSemaphoreCreateCountingStatic() then the application writer can instead optionally provide the memory that will get used by the counting semaphore. xSemaphoreCreateCountingStatic() therefore allows a counting semaphore to be created without using any dynamic memory allocation.

Counting semaphores are typically used for two things:

1) Counting events.

In this usage scenario an event handler will 'give' a semaphore each time an event occurs (incrementing the semaphore count value), and a handler task will 'take' a semaphore each time it processes an event (decrementing the semaphore count value). The count value is therefore the difference between the number of events that have occurred and the number that have been processed. In this case it is desirable for the initial count value to be zero.

2) Resource management.

In this usage scenario the count value indicates the number of resources available. To obtain control of a resource a task must first obtain a semaphore - decrementing the semaphore count value. When the count value reaches zero there are no free resources. When a task finishes with the resource it 'gives' the semaphore back - incrementing the semaphore count value. In this case it is desirable for the initial count value to be equal to the maximum count value, indicating that all resources are free.

Example usage:

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
```

(下页继续)

(续上页)

```
SemaphoreHandle_t xSemaphore = NULL;

 // Semaphore cannot be used before a call to xSemaphoreCreateCounting().
 // The max value to which the semaphore can count should be 10, and the
 // initial value assigned to the count should be 0.
 xSemaphore = xSemaphoreCreateCounting( 10, 0 );

 if( xSemaphore != NULL )
 {
// The semaphore was created successfully.
// The semaphore can now be used.
 }
}
```

参数

- **uxMaxCount** -- The maximum count value that can be reached. When the semaphore reaches this value it can no longer be 'given'.
- **uxInitialCount** -- The count value assigned to the semaphore when it is created.

返回 Handle to the created semaphore. Null if the semaphore could not be created.

xSemaphoreCreateCountingStatic(uxMaxCount, uxInitialCount, pxSemaphoreBuffer)

Creates a new counting semaphore instance, and returns a handle by which the new counting semaphore can be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a counting semaphore! https://www.FreeRTOS.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, counting semaphores use a block of memory, in which the counting semaphore structure is stored. If a counting semaphore is created using xSemaphoreCreateCounting() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateCounting() function. (see https://www.FreeRTOS.org/a00111.html). If a counting semaphore is created using xSemaphoreCreateCountingStatic() then the application writer must provide the memory. xSemaphoreCreateCountingStatic() therefore allows a counting semaphore to be created without using any dynamic memory allocation.

Counting semaphores are typically used for two things:

1) Counting events.

In this usage scenario an event handler will 'give' a semaphore each time an event occurs (incrementing the semaphore count value), and a handler task will 'take' a semaphore each time it processes an event (decrementing the semaphore count value). The count value is therefore the difference between the number of events that have occurred and the number that have been processed. In this case it is desirable for the initial count value to be zero.

2) Resource management.

In this usage scenario the count value indicates the number of resources available. To obtain control of a resource a task must first obtain a semaphore - decrementing the semaphore count value. When the count value reaches zero there are no free resources. When a task finishes with the resource it 'gives' the semaphore back - incrementing the semaphore count value. In this case it is desirable for the initial count value to be equal to the maximum count value, indicating that all resources are free.

Example usage:

```
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters )
```

(下页继续)

(续上页)

```
{
SemaphoreHandle_t xSemaphore = NULL;

 // Counting semaphore cannot be used before they have been created.  Create
 // a counting semaphore using xSemaphoreCreateCountingStatic().  The max
 // value to which the semaphore can count is 10, and the initial value
 // assigned to the count will be 0.  The address of xSemaphoreBuffer is
 // passed in and will be used to hold the semaphore structure, so no dynamic
 // memory allocation will be used.
 xSemaphore = xSemaphoreCreateCounting( 10, 0, &xSemaphoreBuffer );

 // No memory allocation was attempted so xSemaphore cannot be NULL, so there
 // is no need to check its value.
}
```

参数

- **uxMaxCount** -- The maximum count value that can be reached. When the semaphore reaches this value it can no longer be 'given'.
- **uxInitialCount** -- The count value assigned to the semaphore when it is created.
- **pxSemaphoreBuffer** -- Must point to a variable of type StaticSemaphore_t, which will then be used to hold the semaphore's data structure, removing the need for the memory to be allocated dynamically.

返回 If the counting semaphore was successfully created then a handle to the created counting semaphore is returned. If pxSemaphoreBuffer was NULL then NULL is returned.

vSemaphoreDelete(xSemaphore)

Delete a semaphore. This function must be used with care. For example, do not delete a mutex type semaphore if the mutex is held by a task.

参数

- **xSemaphore** -- A handle to the semaphore to be deleted.

xSemaphoreGetMutexHolder(xSemaphore)

If xMutex is indeed a mutex type semaphore, return the current mutex holder. If xMutex is not a mutex type semaphore, or the mutex is available (not held by a task), return NULL.

Note: This is a good way of determining if the calling task is the mutex holder, but not a good way of determining the identity of the mutex holder as the holder may change between the function exiting and the returned value being tested.

xSemaphoreGetMutexHolderFromISR(xSemaphore)

If xMutex is indeed a mutex type semaphore, return the current mutex holder. If xMutex is not a mutex type semaphore, or the mutex is available (not held by a task), return NULL.

uxSemaphoreGetCount(xSemaphore)

If the semaphore is a counting semaphore then uxSemaphoreGetCount() returns its current count value. If the semaphore is a binary semaphore then uxSemaphoreGetCount() returns 1 if the semaphore is available, and 0 if the semaphore is not available.

uxSemaphoreGetCountFromISR(xSemaphore)

semphr.h

```
UBaseType_t uxSemaphoreGetCountFromISR( SemaphoreHandle_t xSemaphore );
```

If the semaphore is a counting semaphore then uxSemaphoreGetCountFromISR() returns its current count value. If the semaphore is a binary semaphore then uxSemaphoreGetCountFromISR() returns 1 if the semaphore is available, and 0 if the semaphore is not available.

xSemaphoreGetStaticBuffer(xSemaphore, ppxSemaphoreBuffer)

Retrieve pointer to a statically created binary semaphore, counting semaphore, or mutex semaphore's data structure buffer. This is the same buffer that is supplied at the time of creation.

参数

- **xSemaphore** -- The semaphore for which to retrieve the buffer.
- **ppxSemaphoreBuffer** -- Used to return a pointer to the semaphore's data structure buffer.

返回 pdTRUE if buffer was retrieved, pdFALSE otherwise.

Type Definitions

typedef *QueueHandle_t* **SemaphoreHandle_t**

定时器 API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/timers.h
- This header file can be included with:

```
#include "freertos/timers.h"
```

Functions

TimerHandle_t **xTimerCreate**(const char *const pcTimerName, const TickType_t xTimerPeriodInTicks, const BaseType_t xAutoReload, void *const pvTimerID, *TimerCallbackFunction_t* pxCallbackFunction)

Creates a new software timer instance, and returns a handle by which the created software timer can be referenced.

Internally, within the FreeRTOS implementation, software timers use a block of memory, in which the timer data structure is stored. If a software timer is created using xTimerCreate() then the required memory is automatically dynamically allocated inside the xTimerCreate() function. (see https://www.FreeRTOS.org/a00111.html). If a software timer is created using xTimerCreateStatic() then the application writer must provide the memory that will get used by the software timer. xTimerCreateStatic() therefore allows a software timer to be created without using any dynamic memory allocation.

Timers are created in the dormant state. The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() and xTimerChangePeriodFromISR() API functions can all be used to transition a timer into the active state.

Example usage:

```
* #define NUM_TIMERS 5
*
* // An array to hold handles to the created timers.
* TimerHandle_t xTimers[ NUM_TIMERS ];
*
* // An array to hold a count of the number of times each timer expires.
* int32_t lExpireCounters[ NUM_TIMERS ] = { 0 };
*
* // Define a callback function that will be used by multiple timer instances.
* // The callback function does nothing but count the number of times the
* // associated timer expires, and stop the timer once the timer has expired
* // 10 times.
* void vTimerCallback( TimerHandle_t pxTimer )
```

(下页继续)

(续上页)

```
* {
* int32_t lArrayIndex;
* const int32_t xMaxExpiryCountBeforeStopping = 10;
*
*     // Optionally do something if the pxTimer parameter is NULL.
*     configASSERT( pxTimer );
*
*     // Which timer expired?
*     lArrayIndex = ( int32_t ) pvTimerGetTimerID( pxTimer );
*
*     // Increment the number of times that pxTimer has expired.
*     lExpireCounters[ lArrayIndex ] += 1;
*
*     // If the timer has expired 10 times then stop it from running.
*     if( lExpireCounters[ lArrayIndex ] == xMaxExpiryCountBeforeStopping )
*     {
*         // Do not use a block time if calling a timer API function from a
*         // timer callback function, as doing so could cause a deadlock!
*         xTimerStop( pxTimer, 0 );
*     }
* }
*
* void main( void )
* {
* int32_t x;
*
*     // Create then start some timers.  Starting the timers before the␣
↪scheduler
*     // has been started means the timers will start running immediately that
*     // the scheduler starts.
*     for( x = 0; x < NUM_TIMERS; x++ )
*     {
*         xTimers[ x ] = xTimerCreate(    "Timer",             // Just a text␣
↪name, not used by the kernel.
*                                         ( 100 * ( x + 1 ) ), // The timer␣
↪period in ticks.
*                                         pdTRUE,              // The timers␣
↪will auto-reload themselves when they expire.
*                                         ( void * ) x,        // Assign each␣
↪timer a unique id equal to its array index.
*                                         vTimerCallback       // Each timer␣
↪calls the same callback when it expires.
*                                       );
*
*         if( xTimers[ x ] == NULL )
*         {
*             // The timer was not created.
*         }
*         else
*         {
*             // Start the timer.  No block time is specified, and even if one␣
↪was
*             // it would be ignored because the scheduler has not yet been
*             // started.
*             if( xTimerStart( xTimers[ x ], 0 ) != pdPASS )
*             {
*                 // The timer could not be set into the Active state.
*             }
*         }
*     }
*
```

(下页继续)

(续上页)

```
*     // ...
*     // Create tasks here.
*     // ...
*
*     // Starting the scheduler will start the timers running as they have␣
↪already
*     // been set into the active state.
*     vTaskStartScheduler();
*
*     // Should not reach here.
*     for( ;; );
* }
*
```

参数

- **pcTimerName** -- A text name that is assigned to the timer. This is done purely to assist debugging. The kernel itself only ever references a timer by its handle, and never by its name.
- **xTimerPeriodInTicks** -- The timer period. The time is defined in tick periods so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xTimerPeriodInTicks should be set to 100. Alternatively, if the timer must expire after 500ms, then xPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000. Time timer period must be greater than 0.
- **xAutoReload** -- If xAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the xTimerPeriodInTicks parameter. If xAutoReload is set to pdFALSE then the timer will be a one-shot timer and enter the dormant state after it expires.
- **pvTimerID** -- An identifier that is assigned to the timer being created. Typically this would be used in the timer callback function to identify which timer expired when the same callback function is assigned to more than one timer.
- **pxCallbackFunction** -- The function to call when the timer expires. Callback functions must have the prototype defined by TimerCallbackFunction_t, which is "void vCallbackFunction(TimerHandle_t xTimer);".

返回 If the timer is successfully created then a handle to the newly created timer is returned. If the timer cannot be created because there is insufficient FreeRTOS heap remaining to allocate the timer structures then NULL is returned.

TimerHandle_t **xTimerCreateStatic**(const char *const pcTimerName, const TickType_t xTimerPeriodInTicks, const BaseType_t xAutoReload, void *const pvTimerID, *TimerCallbackFunction_t* pxCallbackFunction, StaticTimer_t *pxTimerBuffer)

Creates a new software timer instance, and returns a handle by which the created software timer can be referenced.

Internally, within the FreeRTOS implementation, software timers use a block of memory, in which the timer data structure is stored. If a software timer is created using xTimerCreate() then the required memory is automatically dynamically allocated inside the xTimerCreate() function. (see https://www.FreeRTOS.org/a00111.html). If a software timer is created using xTimerCreateStatic() then the application writer must provide the memory that will get used by the software timer. xTimerCreateStatic() therefore allows a software timer to be created without using any dynamic memory allocation.

Timers are created in the dormant state. The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() and xTimerChangePeriodFromISR() API functions can all be used to transition a timer into the active state.

Example usage:

```
*
* // The buffer used to hold the software timer's data structure.
* static StaticTimer_t xTimerBuffer;
*
* // A variable that will be incremented by the software timer's callback
* // function.
* UBaseType_t uxVariableToIncrement = 0;
*
* // A software timer callback function that increments a variable passed to
* // it when the software timer was created.  After the 5th increment the
* // callback function stops the software timer.
* static void prvTimerCallback( TimerHandle_t xExpiredTimer )
* {
* UBaseType_t *puxVariableToIncrement;
* BaseType_t xReturned;
*
*     // Obtain the address of the variable to increment from the timer ID.
*     puxVariableToIncrement = ( UBaseType_t * ) pvTimerGetTimerID(␣
↪xExpiredTimer );
*
*     // Increment the variable to show the timer callback has executed.
*     ( *puxVariableToIncrement )++;
*
*     // If this callback has executed the required number of times, stop the
*     // timer.
*     if( *puxVariableToIncrement == 5 )
*     {
*         // This is called from a timer callback so must not block.
*         xTimerStop( xExpiredTimer, staticDONT_BLOCK );
*     }
* }
*
*
* void main( void )
* {
*     // Create the software time.  xTimerCreateStatic() has an extra parameter
*     // than the normal xTimerCreate() API function.  The parameter is a␣
↪pointer
*     // to the StaticTimer_t structure that will hold the software timer
*     // structure.  If the parameter is passed as NULL then the structure␣
↪will be
*     // allocated dynamically, just as if xTimerCreate() had been called.
*     xTimer = xTimerCreateStatic( "T1",             // Text name for the task.
↪  Helps debugging only.  Not used by FreeRTOS.
*                                  xTimerPeriod,     // The period of the␣
↪timer in ticks.
*                                  pdTRUE,           // This is an auto-reload␣
↪timer.
*                                  ( void * ) &uxVariableToIncrement,    // A␣
↪variable incremented by the software timer's callback function
*                                  prvTimerCallback, // The function to␣
↪execute when the timer expires.
*                                  &xTimerBuffer );  // The buffer that will␣
↪hold the software timer structure.
*
*     // The scheduler has not started yet so a block time is not used.
*     xReturned = xTimerStart( xTimer, 0 );
*
*     // ...
*     // Create tasks here.
*     // ...
*
```

(下页继续)

(续上页)

```
*     // Starting the scheduler will start the timers running as they have␣
↪already
*     // been set into the active state.
*     vTaskStartScheduler();
*
*     // Should not reach here.
*     for( ;; );
* }
*
```

参数

- **pcTimerName** -- A text name that is assigned to the timer. This is done purely to assist debugging. The kernel itself only ever references a timer by its handle, and never by its name.
- **xTimerPeriodInTicks** -- The timer period. The time is defined in tick periods so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xTimerPeriodInTicks should be set to 100. Alternatively, if the timer must expire after 500ms, then xPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000. The timer period must be greater than 0.
- **xAutoReload** -- If xAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the xTimerPeriodInTicks parameter. If xAutoReload is set to pdFALSE then the timer will be a one-shot timer and enter the dormant state after it expires.
- **pvTimerID** -- An identifier that is assigned to the timer being created. Typically this would be used in the timer callback function to identify which timer expired when the same callback function is assigned to more than one timer.
- **pxCallbackFunction** -- The function to call when the timer expires. Callback functions must have the prototype defined by TimerCallbackFunction_t, which is "void vCallbackFunction(TimerHandle_t xTimer);".
- **pxTimerBuffer** -- Must point to a variable of type StaticTimer_t, which will be then be used to hold the software timer's data structures, removing the need for the memory to be allocated dynamically.

返回 If the timer is created then a handle to the created timer is returned. If pxTimerBuffer was NULL then NULL is returned.

void ***pvTimerGetTimerID**(const *TimerHandle_t* xTimer)

Returns the ID assigned to the timer.

IDs are assigned to timers using the pvTimerID parameter of the call to xTimerCreated() that was used to create the timer, and by calling the vTimerSetTimerID() API function.

If the same callback function is assigned to multiple timers then the timer ID can be used as time specific (timer local) storage.

Example usage:

See the xTimerCreate() API function example usage scenario.

参数 **xTimer** -- The timer being queried.
返回 The ID assigned to the timer being queried.

void **vTimerSetTimerID**(*TimerHandle_t* xTimer, void *pvNewID)

Sets the ID assigned to the timer.

IDs are assigned to timers using the pvTimerID parameter of the call to xTimerCreated() that was used to create the timer.

If the same callback function is assigned to multiple timers then the timer ID can be used as time specific (timer local) storage.

Example usage:

See the xTimerCreate() API function example usage scenario.

参数

- **xTimer** -- The timer being updated.
- **pvNewID** -- The ID to assign to the timer.

BaseType_t **xTimerIsTimerActive**(*TimerHandle_t* xTimer)

Queries a timer to see if it is active or dormant.

A timer will be dormant if: 1) It has been created but not started, or 2) It is an expired one-shot timer that has not been restarted.

Timers are created in the dormant state. The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() and xTimerChangePeriodFromISR() API functions can all be used to transition a timer into the active state.

Example usage:

```
* // This function assumes xTimer has already been created.
* void vAFunction( TimerHandle_t xTimer )
* {
*     if( xTimerIsTimerActive( xTimer ) != pdFALSE ) // or more simply and␣
↪equivalently "if( xTimerIsTimerActive( xTimer ) )"
*     {
*         // xTimer is active, do something.
*     }
*     else
*     {
*         // xTimer is not active, do something else.
*     }
* }
*
```

参数 **xTimer** -- The timer being queried.

返回 pdFALSE will be returned if the timer is dormant. A value other than pdFALSE will be returned if the timer is active.

TaskHandle_t **xTimerGetTimerDaemonTaskHandle**(void)

Simply returns the handle of the timer service/daemon task. It it not valid to call xTimerGetTimerDaemonTaskHandle() before the scheduler has been started.

BaseType_t **xTimerPendFunctionCallFromISR**(*PendedFunction_t* xFunctionToPend, void *pvParameter1, uint32_t ulParameter2, BaseType_t *pxHigherPriorityTaskWoken)

Used from application interrupt service routines to defer the execution of a function to the RTOS daemon task (the timer service task, hence this function is implemented in timers.c and is prefixed with 'Timer').

Ideally an interrupt service routine (ISR) is kept as short as possible, but sometimes an ISR either has a lot of processing to do, or needs to perform processing that is not deterministic. In these cases xTimerPendFunctionCallFromISR() can be used to defer processing of a function to the RTOS daemon task.

A mechanism is provided that allows the interrupt to return directly to the task that will subsequently execute the pended callback function. This allows the callback function to execute contiguously in time with the interrupt - just as if the callback had executed in the interrupt itself.

Example usage:

```
*
*  // The callback function that will execute in the context of the daemon␣
↪task.
*  // Note callback functions must all use this same prototype.
*  void vProcessInterface( void *pvParameter1, uint32_t ulParameter2 )
*  {
*      BaseType_t xInterfaceToService;
*
*      // The interface that requires servicing is passed in the second
*      // parameter.  The first parameter is not used in this case.
*      xInterfaceToService = ( BaseType_t ) ulParameter2;
*
*      // ...Perform the processing here...
*  }
*
*  // An ISR that receives data packets from multiple interfaces
*  void vAnISR( void )
*  {
*      BaseType_t xInterfaceToService, xHigherPriorityTaskWoken;
*
*      // Query the hardware to determine which interface needs processing.
*      xInterfaceToService = prvCheckInterfaces();
*
*      // The actual processing is to be deferred to a task.  Request the
*      // vProcessInterface() callback function is executed, passing in the
*      // number of the interface that needs processing.  The interface to
*      // service is passed in the second parameter.  The first parameter is
*      // not used in this case.
*      xHigherPriorityTaskWoken = pdFALSE;
*      xTimerPendFunctionCallFromISR( vProcessInterface, NULL, ( uint32_t )␣
↪xInterfaceToService, &xHigherPriorityTaskWoken );
*
*      // If xHigherPriorityTaskWoken is now set to pdTRUE then a context
*      // switch should be requested.  The macro used is port specific and will
*      // be either portYIELD_FROM_ISR() or portEND_SWITCHING_ISR() - refer to
*      // the documentation page for the port being used.
*      portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
*
*  }
*
```

参数

- **xFunctionToPend** -- The function to execute from the timer service/ daemon task. The function must conform to the PendedFunction_t prototype.
- **pvParameter1** -- The value of the callback function's first parameter. The parameter has a void * type to allow it to be used to pass any type. For example, unsigned longs can be cast to a void *, or the void * can be used to point to a structure.
- **ulParameter2** -- The value of the callback function's second parameter.
- **pxHigherPriorityTaskWoken** -- As mentioned above, calling this function will result in a message being sent to the timer daemon task. If the priority of the timer daemon task (which is set using configTIMER_TASK_PRIORITY in FreeRTOSConfig.h) is higher than the priority of the currently running task (the task the interrupt interrupted) then *pxHigherPriorityTaskWoken will be set to pdTRUE within xTimerPendFunctionCallFromISR(), indicating that a context switch should be requested before the interrupt exits. For that reason *pxHigherPriorityTaskWoken must be initialised to pdFALSE. See the example code below.

返回 pdPASS is returned if the message was successfully sent to the timer daemon task, otherwise pdFALSE is returned.

BaseType_t **xTimerPendFunctionCall**(*PendedFunction_t* xFunctionToPend, void *pvParameter1, uint32_t ulParameter2, TickType_t xTicksToWait)

Used to defer the execution of a function to the RTOS daemon task (the timer service task, hence this function is implemented in timers.c and is prefixed with 'Timer').

参数

- **xFunctionToPend** -- The function to execute from the timer service/ daemon task. The function must conform to the PendedFunction_t prototype.
- **pvParameter1** -- The value of the callback function's first parameter. The parameter has a void * type to allow it to be used to pass any type. For example, unsigned longs can be cast to a void *, or the void * can be used to point to a structure.
- **ulParameter2** -- The value of the callback function's second parameter.
- **xTicksToWait** -- Calling this function will result in a message being sent to the timer daemon task on a queue. xTicksToWait is the amount of time the calling task should remain in the Blocked state (so not using any processing time) for space to become available on the timer queue if the queue is found to be full.

返回 pdPASS is returned if the message was successfully sent to the timer daemon task, otherwise pdFALSE is returned.

const char *__pcTimerGetName__(*TimerHandle_t* xTimer)

Returns the name that was assigned to a timer when the timer was created.

参数 **xTimer** -- The handle of the timer being queried.

返回 The name assigned to the timer specified by the xTimer parameter.

void **vTimerSetReloadMode**(*TimerHandle_t* xTimer, const BaseType_t xAutoReload)

Updates a timer to be either an auto-reload timer, in which case the timer automatically resets itself each time it expires, or a one-shot timer, in which case the timer will only expire once unless it is manually restarted.

参数

- **xTimer** -- The handle of the timer being updated.
- **xAutoReload** -- If xAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the timer's period (see the xTimerPeriodInTicks parameter of the xTimerCreate() API function). If xAutoReload is set to pdFALSE then the timer will be a one-shot timer and enter the dormant state after it expires.

BaseType_t **xTimerGetReloadMode**(*TimerHandle_t* xTimer)

Queries a timer to determine if it is an auto-reload timer, in which case the timer automatically resets itself each time it expires, or a one-shot timer, in which case the timer will only expire once unless it is manually restarted.

参数 **xTimer** -- The handle of the timer being queried.

返回 If the timer is an auto-reload timer then pdTRUE is returned, otherwise pdFALSE is returned.

UBaseType_t **uxTimerGetReloadMode**(*TimerHandle_t* xTimer)

Queries a timer to determine if it is an auto-reload timer, in which case the timer automatically resets itself each time it expires, or a one-shot timer, in which case the timer will only expire once unless it is manually restarted.

参数 **xTimer** -- The handle of the timer being queried.

返回 If the timer is an auto-reload timer then pdTRUE is returned, otherwise pdFALSE is returned.

TickType_t **xTimerGetPeriod**(*TimerHandle_t* xTimer)

Returns the period of a timer.

参数 **xTimer** -- The handle of the timer being queried.

返回 The period of the timer in ticks.

TickType_t **xTimerGetExpiryTime**(*TimerHandle_t* xTimer)

Returns the time in ticks at which the timer will expire. If this is less than the current tick count then the expiry time has overflowed from the current time.

参数 **xTimer** -- The handle of the timer being queried.

返回 If the timer is running then the time in ticks at which the timer will next expire is returned. If the timer is not running then the return value is undefined.

BaseType_t **xTimerGetStaticBuffer**(*TimerHandle_t* xTimer, StaticTimer_t **ppxTimerBuffer)

Retrieve pointer to a statically created timer's data structure buffer. This is the same buffer that is supplied at the time of creation.

参数

- **xTimer** -- The timer for which to retrieve the buffer.
- **ppxTimerBuffer** -- Used to return a pointer to the timers's data structure buffer.

返回 pdTRUE if the buffer was retrieved, pdFALSE otherwise.

void **vApplicationGetTimerTaskMemory**(StaticTask_t **ppxTimerTaskTCBBuffer, StackType_t **ppxTimerTaskStackBuffer, uint32_t *pulTimerTaskStackSize)

This function is used to provide a statically allocated block of memory to FreeRTOS to hold the Timer Task TCB. This function is required when configSUPPORT_STATIC_ALLOCATION is set. For more information see this URI: https://www.FreeRTOS.org/a00110.html#configSUPPORT_STATIC_ALLOCATION

参数

- **ppxTimerTaskTCBBuffer** -- A handle to a statically allocated TCB buffer
- **ppxTimerTaskStackBuffer** -- A handle to a statically allocated Stack buffer for the idle task
- **pulTimerTaskStackSize** -- A pointer to the number of elements that will fit in the allocated stack buffer

Macros

xTimerStart(xTimer, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerStart() starts a timer that was previously created using the xTimerCreate() API function. If the timer had already been started and was already in the active state, then xTimerStart() has equivalent functionality to the xTimerReset() API function.

Starting a timer ensures the timer is in the active state. If the timer is not stopped, deleted, or reset in the mean time, the callback function associated with the timer will get called 'n' ticks after xTimerStart() was called, where 'n' is the timers defined period.

It is valid to call xTimerStart() before the scheduler has been started, but when this is done the timer will not actually start until the scheduler is started, and the timers expiry time will be relative to when the scheduler is started, not relative to when xTimerStart() was called.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerStart() to be available.

Example usage:

See the xTimerCreate() API function example usage scenario.

参数

- **xTimer** -- The handle of the timer being started/restarted.
- **xTicksToWait** -- Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the start command to be successfully sent to the timer command queue, should the queue already be full when xTimerStart() was called. xTicksToWait is ignored if xTimerStart() is called before the scheduler is started.

返回 pdFAIL will be returned if the start command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will

depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStart() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerStop(xTimer, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerStop() stops a timer that was previously started using either of the The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() or xTimerChangePeriodFromISR() API functions.

Stopping a timer ensures the timer is not in the active state.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerStop() to be available.

Example usage:

See the xTimerCreate() API function example usage scenario.

参数

- **xTimer** -- The handle of the timer being stopped.
- **xTicksToWait** -- Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the stop command to be successfully sent to the timer command queue, should the queue already be full when xTimerStop() was called. xTicksToWait is ignored if xTimerStop() is called before the scheduler is started.

返回 pdFAIL will be returned if the stop command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerChangePeriod(xTimer, xNewPeriod, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerChangePeriod() changes the period of a timer that was previously created using the xTimerCreate() API function.

xTimerChangePeriod() can be called to change the period of an active or dormant state timer.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerChangePeriod() to be available.

Example usage:

```
* // This function assumes xTimer has already been created.  If the timer
* // referenced by xTimer is already active when it is called, then the timer
* // is deleted.  If the timer referenced by xTimer is not active when it is
* // called, then the period of the timer is set to 500ms and the timer is
* // started.
* void vAFunction( TimerHandle_t xTimer )
* {
*     if( xTimerIsTimerActive( xTimer ) != pdFALSE ) // or more simply and␣
↪equivalently "if( xTimerIsTimerActive( xTimer ) )"
*     {
```

(下页继续)

(续上页)

```
*          // xTimer is already active - delete it.
*          xTimerDelete( xTimer );
*      }
*      else
*      {
*          // xTimer is not active, change its period to 500ms.  This will also
*          // cause the timer to start.  Block for a maximum of 100 ticks if the
*          // change period command cannot immediately be sent to the timer
*          // command queue.
*          if( xTimerChangePeriod( xTimer, 500 / portTICK_PERIOD_MS, 100 ) ==␣
↪pdPASS )
*          {
*              // The command was successfully sent.
*          }
*          else
*          {
*              // The command could not be sent, even after waiting for 100␣
↪ticks
*              // to pass.  Take appropriate action here.
*          }
*      }
* }
*
```

参数

- **xTimer** -- The handle of the timer that is having its period changed.
- **xNewPeriod** -- The new period for xTimer. Timer periods are specified in tick periods, so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xNewPeriod should be set to 100. Alternatively, if the timer must expire after 500ms, then xNewPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
- **xTicksToWait** -- Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the change period command to be successfully sent to the timer command queue, should the queue already be full when xTimerChangePeriod() was called. xTicksToWait is ignored if xTimerChangePeriod() is called before the scheduler is started.

返回 pdFAIL will be returned if the change period command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerDelete(xTimer, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerDelete() deletes a timer that was previously created using the xTimerCreate() API function.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerDelete() to be available.

Example usage:

See the xTimerChangePeriod() API function example usage scenario.

参数

- **xTimer** -- The handle of the timer being deleted.
- **xTicksToWait** -- Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the delete command to be successfully sent to the timer command queue, should the queue already be full when xTimerDelete() was called. xTicksToWait is ignored if xTimerDelete() is called before the scheduler is started.

返回 pdFAIL will be returned if the delete command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerReset(xTimer, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerReset() re-starts a timer that was previously created using the xTimerCreate() API function. If the timer had already been started and was already in the active state, then xTimerReset() will cause the timer to re-evaluate its expiry time so that it is relative to when xTimerReset() was called. If the timer was in the dormant state then xTimerReset() has equivalent functionality to the xTimerStart() API function.

Resetting a timer ensures the timer is in the active state. If the timer is not stopped, deleted, or reset in the mean time, the callback function associated with the timer will get called 'n' ticks after xTimerReset() was called, where 'n' is the timers defined period.

It is valid to call xTimerReset() before the scheduler has been started, but when this is done the timer will not actually start until the scheduler is started, and the timers expiry time will be relative to when the scheduler is started, not relative to when xTimerReset() was called.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerReset() to be available.

Example usage:

```
* // When a key is pressed, an LCD back-light is switched on.  If 5 seconds␣
↪pass
* // without a key being pressed, then the LCD back-light is switched off.  In
* // this case, the timer is a one-shot timer.
*
* TimerHandle_t xBacklightTimer = NULL;
*
* // The callback function assigned to the one-shot timer.  In this case the
* // parameter is not used.
* void vBacklightTimerCallback( TimerHandle_t pxTimer )
* {
*     // The timer expired, therefore 5 seconds must have passed since a key
*     // was pressed.  Switch off the LCD back-light.
*     vSetBacklightState( BACKLIGHT_OFF );
* }
*
* // The key press event handler.
* void vKeyPressEventHandler( char cKey )
* {
*     // Ensure the LCD back-light is on, then reset the timer that is
*     // responsible for turning the back-light off after 5 seconds of
*     // key inactivity.  Wait 10 ticks for the command to be successfully sent
*     // if it cannot be sent immediately.
*     vSetBacklightState( BACKLIGHT_ON );
*     if( xTimerReset( xBacklightTimer, 100 ) != pdPASS )
*     {
```

(下页继续)

(续上页)

```
*          // The reset command was not executed successfully.  Take appropriate
*          // action here.
*      }
*
*      // Perform the rest of the key processing here.
* }
*
* void main( void )
* {
* int32_t x;
*
*      // Create then start the one-shot timer that is responsible for turning
*      // the back-light off if no keys are pressed within a 5 second period.
*      xBacklightTimer = xTimerCreate( "BacklightTimer",           // Just a␣
↪text name, not used by the kernel.
*                                      ( 5000 / portTICK_PERIOD_MS), // The␣
↪timer period in ticks.
*                                      pdFALSE,                    // The timer␣
↪is a one-shot timer.
*                                      0,                          // The id is␣
↪not used by the callback so can take any value.
*                                      vBacklightTimerCallback     // The␣
↪callback function that switches the LCD back-light off.
*                                    );
*
*      if( xBacklightTimer == NULL )
*      {
*          // The timer was not created.
*      }
*      else
*      {
*          // Start the timer.  No block time is specified, and even if one was
*          // it would be ignored because the scheduler has not yet been
*          // started.
*          if( xTimerStart( xBacklightTimer, 0 ) != pdPASS )
*          {
*              // The timer could not be set into the Active state.
*          }
*      }
*
*      // ...
*      // Create tasks here.
*      // ...
*
*      // Starting the scheduler will start the timer running as it has already
*      // been set into the active state.
*      vTaskStartScheduler();
*
*      // Should not reach here.
*      for( ;; );
* }
*
```

参数

- **xTimer** -- The handle of the timer being reset/started/restarted.
- **xTicksToWait** -- Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the reset command to be successfully sent to the timer command queue, should the queue already be full when xTimerReset() was called. xTicksToWait is ignored if xTimerReset() is called before the scheduler is started.

返回 pdFAIL will be returned if the reset command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command

was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStart() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerStartFromISR(xTimer, pxHigherPriorityTaskWoken)

A version of xTimerStart() that can be called from an interrupt service routine.

Example usage:

```
* // This scenario assumes xBacklightTimer has already been created.  When a
* // key is pressed, an LCD back-light is switched on.  If 5 seconds pass
* // without a key being pressed, then the LCD back-light is switched off.  In
* // this case, the timer is a one-shot timer, and unlike the example given for
* // the xTimerReset() function, the key press event handler is an interrupt
* // service routine.
*
* // The callback function assigned to the one-shot timer.  In this case the
* // parameter is not used.
* void vBacklightTimerCallback( TimerHandle_t pxTimer )
* {
*     // The timer expired, therefore 5 seconds must have passed since a key
*     // was pressed.  Switch off the LCD back-light.
*     vSetBacklightState( BACKLIGHT_OFF );
* }
*
* // The key press interrupt service routine.
* void vKeyPressEventInterruptHandler( void )
* {
* BaseType_t xHigherPriorityTaskWoken = pdFALSE;
*
*     // Ensure the LCD back-light is on, then restart the timer that is
*     // responsible for turning the back-light off after 5 seconds of
*     // key inactivity.  This is an interrupt service routine so can only
*     // call FreeRTOS API functions that end in "FromISR".
*     vSetBacklightState( BACKLIGHT_ON );
*
*     // xTimerStartFromISR() or xTimerResetFromISR() could be called here
*     // as both cause the timer to re-calculate its expiry time.
*     // xHigherPriorityTaskWoken was initialised to pdFALSE when it was
*     // declared (in this function).
*     if( xTimerStartFromISR( xBacklightTimer, &xHigherPriorityTaskWoken ) !=␣
↪pdPASS )
*     {
*         // The start command was not executed successfully.  Take appropriate
*         // action here.
*     }
*
*     // Perform the rest of the key processing here.
*
*     // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
*     // should be performed.  The syntax required to perform a context switch
*     // from inside an ISR varies from port to port, and from compiler to
*     // compiler.  Inspect the demos for the port you are using to find the
*     // actual syntax required.
*     if( xHigherPriorityTaskWoken != pdFALSE )
*     {
*         // Call the interrupt safe yield function here (actual function
*         // depends on the FreeRTOS port being used).
```

(下页继续)

(续上页)

```
*      }
* }
*
```

参数

- **xTimer** -- The handle of the timer being started/restarted.
- **pxHigherPriorityTaskWoken** -- The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerStartFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerStartFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/ daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then *pxHigherPriorityTaskWoken will get set to pdTRUE internally within the xTimerStartFromISR() function. If xTimerStartFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

返回 pdFAIL will be returned if the start command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStartFromISR() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerStopFromISR(xTimer, pxHigherPriorityTaskWoken)

A version of xTimerStop() that can be called from an interrupt service routine.

Example usage:

```
* // This scenario assumes xTimer has already been created and started.  When
* // an interrupt occurs, the timer should be simply stopped.
*
* // The interrupt service routine that stops the timer.
* void vAnExampleInterruptServiceRoutine( void )
* {
* BaseType_t xHigherPriorityTaskWoken = pdFALSE;
*
*     // The interrupt has occurred - simply stop the timer.
*     // xHigherPriorityTaskWoken was set to pdFALSE where it was defined
*     // (within this function).  As this is an interrupt service routine, only
*     // FreeRTOS API functions that end in "FromISR" can be used.
*     if( xTimerStopFromISR( xTimer, &xHigherPriorityTaskWoken ) != pdPASS )
*     {
*         // The stop command was not executed successfully.  Take appropriate
*         // action here.
*     }
*
*     // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
*     // should be performed.  The syntax required to perform a context switch
*     // from inside an ISR varies from port to port, and from compiler to
*     // compiler.  Inspect the demos for the port you are using to find the
*     // actual syntax required.
*     if( xHigherPriorityTaskWoken != pdFALSE )
*     {
*         // Call the interrupt safe yield function here (actual function
*         // depends on the FreeRTOS port being used).
*     }
* }
*
```

参数

- **xTimer** -- The handle of the timer being stopped.
- **pxHigherPriorityTaskWoken** -- The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerStopFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerStopFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/ daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then *pxHigherPriorityTaskWoken will get set to pdTRUE internally within the xTimerStopFromISR() function. If xTimerStopFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

返回 pdFAIL will be returned if the stop command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerChangePeriodFromISR(xTimer, xNewPeriod, pxHigherPriorityTaskWoken)

A version of xTimerChangePeriod() that can be called from an interrupt service routine.

Example usage:

```
* // This scenario assumes xTimer has already been created and started.  When
* // an interrupt occurs, the period of xTimer should be changed to 500ms.
*
* // The interrupt service routine that changes the period of xTimer.
* void vAnExampleInterruptServiceRoutine( void )
* {
* BaseType_t xHigherPriorityTaskWoken = pdFALSE;
*
*     // The interrupt has occurred - change the period of xTimer to 500ms.
*     // xHigherPriorityTaskWoken was set to pdFALSE where it was defined
*     // (within this function).  As this is an interrupt service routine, only
*     // FreeRTOS API functions that end in "FromISR" can be used.
*     if( xTimerChangePeriodFromISR( xTimer, &xHigherPriorityTaskWoken ) !=␣
↪pdPASS )
*     {
*         // The command to change the timers period was not executed
*         // successfully.  Take appropriate action here.
*     }
*
*     // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
*     // should be performed.  The syntax required to perform a context switch
*     // from inside an ISR varies from port to port, and from compiler to
*     // compiler.  Inspect the demos for the port you are using to find the
*     // actual syntax required.
*     if( xHigherPriorityTaskWoken != pdFALSE )
*     {
*         // Call the interrupt safe yield function here (actual function
*         // depends on the FreeRTOS port being used).
*     }
* }
*
```

参数

- **xTimer** -- The handle of the timer that is having its period changed.
- **xNewPeriod** -- The new period for xTimer. Timer periods are specified in tick periods, so the constant portTICK_PERIOD_MS can be used to convert a time that has been spec-

ified in milliseconds. For example, if the timer must expire after 100 ticks, then xNewPeriod should be set to 100. Alternatively, if the timer must expire after 500ms, then xNewPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000.

- **pxHigherPriorityTaskWoken** -- The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerChangePeriodFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/ daemon task out of the Blocked state. If calling xTimerChangePeriodFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then *pxHigherPriorityTaskWoken will get set to pdTRUE internally within the xTimerChangePeriodFromISR() function. If xTimerChangePeriodFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

返回 pdFAIL will be returned if the command to change the timers period could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerResetFromISR(xTimer, pxHigherPriorityTaskWoken)

A version of xTimerReset() that can be called from an interrupt service routine.

Example usage:

```
* // This scenario assumes xBacklightTimer has already been created.  When a
* // key is pressed, an LCD back-light is switched on.  If 5 seconds pass
* // without a key being pressed, then the LCD back-light is switched off.  In
* // this case, the timer is a one-shot timer, and unlike the example given for
* // the xTimerReset() function, the key press event handler is an interrupt
* // service routine.
*
* // The callback function assigned to the one-shot timer.  In this case the
* // parameter is not used.
* void vBacklightTimerCallback( TimerHandle_t pxTimer )
* {
*     // The timer expired, therefore 5 seconds must have passed since a key
*     // was pressed.  Switch off the LCD back-light.
*     vSetBacklightState( BACKLIGHT_OFF );
* }
*
* // The key press interrupt service routine.
* void vKeyPressEventInterruptHandler( void )
* {
* BaseType_t xHigherPriorityTaskWoken = pdFALSE;
*
*     // Ensure the LCD back-light is on, then reset the timer that is
*     // responsible for turning the back-light off after 5 seconds of
*     // key inactivity.  This is an interrupt service routine so can only
*     // call FreeRTOS API functions that end in "FromISR".
*     vSetBacklightState( BACKLIGHT_ON );
*
*     // xTimerStartFromISR() or xTimerResetFromISR() could be called here
*     // as both cause the timer to re-calculate its expiry time.
*     // xHigherPriorityTaskWoken was initialised to pdFALSE when it was
*     // declared (in this function).
*     if( xTimerResetFromISR( xBacklightTimer, &xHigherPriorityTaskWoken ) !=␣
↪pdPASS )
*     {
```

(下页继续)

(续上页)

```
*          // The reset command was not executed successfully.  Take appropriate
*          // action here.
*      }
*
*      // Perform the rest of the key processing here.
*
*      // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
*      // should be performed.  The syntax required to perform a context switch
*      // from inside an ISR varies from port to port, and from compiler to
*      // compiler.  Inspect the demos for the port you are using to find the
*      // actual syntax required.
*      if( xHigherPriorityTaskWoken != pdFALSE )
*      {
*          // Call the interrupt safe yield function here (actual function
*          // depends on the FreeRTOS port being used).
*      }
* }
*
```

参数

- **xTimer** -- The handle of the timer that is to be started, reset, or restarted.
- **pxHigherPriorityTaskWoken** -- The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerResetFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerResetFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/ daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then *pxHigherPriorityTaskWoken will get set to pdTRUE internally within the xTimerResetFromISR() function. If xTimerResetFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

返回 pdFAIL will be returned if the reset command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerResetFromISR() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Type Definitions

typedef struct tmrTimerControl ***TimerHandle_t**

typedef void (***TimerCallbackFunction_t**)(*TimerHandle_t* xTimer)

Defines the prototype to which timer callback functions must conform.

typedef void (***PendedFunction_t**)(void*, uint32_t)

Defines the prototype to which functions used with the xTimerPendFunctionCallFromISR() function must conform.

事件组 API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/event_groups.h
- This header file can be included with:

```
#include "freertos/event_groups.h"
```

Functions

EventGroupHandle_t **xEventGroupCreate**(void)

Create a new event group.

Internally, within the FreeRTOS implementation, event groups use a [small] block of memory, in which the event group's structure is stored. If an event groups is created using xEventGroupCreate() then the required memory is automatically dynamically allocated inside the xEventGroupCreate() function. (see https://www.FreeRTOS.org/a00111.html). If an event group is created using xEventGroupCreateStatic() then the application writer must instead provide the memory that will get used by the event group. xEventGroupCreateStatic() therefore allows an event group to be created without using any dynamic memory allocation.

Although event groups are not related to ticks, for internal implementation reasons the number of bits available for use in an event group is dependent on the configUSE_16_BIT_TICKS setting in FreeRTOSConfig.h. If configUSE_16_BIT_TICKS is 1 then each event group contains 8 usable bits (bit 0 to bit 7). If configUSE_16_BIT_TICKS is set to 0 then each event group has 24 usable bits (bit 0 to bit 23). The EventBits_t type is used to store event bits within an event group.

Example usage:

```
// Declare a variable to hold the created event group.
EventGroupHandle_t xCreatedEventGroup;

// Attempt to create the event group.
xCreatedEventGroup = xEventGroupCreate();

// Was the event group created successfully?
if( xCreatedEventGroup == NULL )
{
// The event group was not created because there was insufficient
// FreeRTOS heap available.
}
else
{
// The event group was created.
}
```

返回 If the event group was created then a handle to the event group is returned. If there was insufficient FreeRTOS heap available to create the event group then NULL is returned. See https://www.FreeRTOS.org/a00111.html

EventGroupHandle_t **xEventGroupCreateStatic**(StaticEventGroup_t *pxEventGroupBuffer)

Create a new event group.

Internally, within the FreeRTOS implementation, event groups use a [small] block of memory, in which the event group's structure is stored. If an event groups is created using xEventGroupCreate() then the required memory is automatically dynamically allocated inside the xEventGroupCreate() function. (see https://www.FreeRTOS.org/a00111.html). If an event group is created using xEventGroupCreateStatic() then the application writer must instead provide the memory that will get used by the event group. xEventGroupCreateStatic() therefore allows an event group to be created without using any dynamic memory allocation.

Although event groups are not related to ticks, for internal implementation reasons the number of bits available for use in an event group is dependent on the configUSE_16_BIT_TICKS setting in FreeRTOSConfig.h. If configUSE_16_BIT_TICKS is 1 then each event group contains 8 usable bits (bit 0 to bit 7). If configUSE_16_BIT_TICKS is set to 0 then each event group has 24 usable bits (bit 0 to bit 23). The EventBits_t type is used to store event bits within an event group.

Example usage:

```
// StaticEventGroup_t is a publicly accessible structure that has the same
// size and alignment requirements as the real event group structure.  It is
// provided as a mechanism for applications to know the size of the event
// group (which is dependent on the architecture and configuration file
// settings) without breaking the strict data hiding policy by exposing the
// real event group internals.  This StaticEventGroup_t variable is passed
// into the xSemaphoreCreateEventGroupStatic() function and is used to store
// the event group's data structures
StaticEventGroup_t xEventGroupBuffer;

// Create the event group without dynamically allocating any memory.
xEventGroup = xEventGroupCreateStatic( &xEventGroupBuffer );
```

参数 **pxEventGroupBuffer** -- pxEventGroupBuffer must point to a variable of type StaticEventGroup_t, which will be then be used to hold the event group's data structures, removing the need for the memory to be allocated dynamically.

返回 If the event group was created then a handle to the event group is returned. If pxEventGroupBuffer was NULL then NULL is returned.

EventBits_t **xEventGroupWaitBits**(*EventGroupHandle_t* xEventGroup, const *EventBits_t* uxBitsToWaitFor, const BaseType_t xClearOnExit, const BaseType_t xWaitForAllBits, TickType_t xTicksToWait)

[Potentially] block to wait for one or more bits to be set within a previously created event group.

This function cannot be called from an interrupt.

Example usage:

```
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
{
EventBits_t uxBits;
const TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;

// Wait a maximum of 100ms for either bit 0 or bit 4 to be set within
// the event group.  Clear the bits before exiting.
     uxBits = xEventGroupWaitBits(
                 xEventGroup,    // The event group being tested.
                 BIT_0 | BIT_4,  // The bits within the event group to wait␣
↪for.
                 pdTRUE,         // BIT_0 and BIT_4 should be cleared before␣
↪returning.
                 pdFALSE,        // Don't wait for both bits, either bit will␣
↪do.
                 xTicksToWait ); // Wait a maximum of 100ms for either bit to␣
↪be set.

if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) )
     {
// xEventGroupWaitBits() returned because both bits were set.
     }
else if( ( uxBits & BIT_0 ) != 0 )
     {
// xEventGroupWaitBits() returned because just BIT_0 was set.
     }
else if( ( uxBits & BIT_4 ) != 0 )
```

(下页继续)

(续上页)

```
    {
// xEventGroupWaitBits() returned because just BIT_4 was set.
    }
else
    {
// xEventGroupWaitBits() returned because xTicksToWait ticks passed
// without either BIT_0 or BIT_4 becoming set.
    }
}
```

参数

- **xEventGroup** -- The event group in which the bits are being tested. The event group must have previously been created using a call to xEventGroupCreate().
- **uxBitsToWaitFor** -- A bitwise value that indicates the bit or bits to test inside the event group. For example, to wait for bit 0 and/or bit 2 set uxBitsToWaitFor to 0x05. To wait for bits 0 and/or bit 1 and/or bit 2 set uxBitsToWaitFor to 0x07. Etc.
- **xClearOnExit** -- If xClearOnExit is set to pdTRUE then any bits within uxBitsToWaitFor that are set within the event group will be cleared before xEventGroupWaitBits() returns if the wait condition was met (if the function returns for a reason other than a timeout). If xClearOnExit is set to pdFALSE then the bits set in the event group are not altered when the call to xEventGroupWaitBits() returns.
- **xWaitForAllBits** -- If xWaitForAllBits is set to pdTRUE then xEventGroupWaitBits() will return when either all the bits in uxBitsToWaitFor are set or the specified block time expires. If xWaitForAllBits is set to pdFALSE then xEventGroupWaitBits() will return when any one of the bits set in uxBitsToWaitFor is set or the specified block time expires. The block time is specified by the xTicksToWait parameter.
- **xTicksToWait** -- The maximum amount of time (specified in 'ticks') to wait for one/all (depending on the xWaitForAllBits value) of the bits specified by uxBitsToWaitFor to become set. A value of portMAX_DELAY can be used to block indefinitely (provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h).

返回 The value of the event group at the time either the bits being waited for became set, or the block time expired. Test the return value to know which bits were set. If xEventGroupWaitBits() returned because its timeout expired then not all the bits being waited for will be set. If xEventGroupWaitBits() returned because the bits it was waiting for were set then the returned value is the event group value before any bits were automatically cleared in the case that xClearOnExit parameter was set to pdTRUE.

EventBits_t **xEventGroupClearBits** (*EventGroupHandle_t* xEventGroup, const *EventBits_t* uxBitsToClear)

Clear bits within an event group. This function cannot be called from an interrupt.

Example usage:

```
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
{
EventBits_t uxBits;

// Clear bit 0 and bit 4 in xEventGroup.
    uxBits = xEventGroupClearBits(
                              xEventGroup,    // The event group being updated.
                              BIT_0 | BIT_4 );// The bits being cleared.

if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) )
    {
```

(下页继续)

(续上页)

```
// Both bit 0 and bit 4 were set before xEventGroupClearBits() was
// called.  Both will now be clear (not set).
     }
else if( ( uxBits & BIT_0 ) != 0 )
     {
// Bit 0 was set before xEventGroupClearBits() was called.  It will
// now be clear.
     }
else if( ( uxBits & BIT_4 ) != 0 )
     {
// Bit 4 was set before xEventGroupClearBits() was called.  It will
// now be clear.
     }
else
     {
// Neither bit 0 nor bit 4 were set in the first place.
     }
}
```

参数

- **xEventGroup** -- The event group in which the bits are to be cleared.
- **uxBitsToClear** -- A bitwise value that indicates the bit or bits to clear in the event group. For example, to clear bit 3 only, set uxBitsToClear to 0x08. To clear bit 3 and bit 0 set uxBitsToClear to 0x09.

返回 The value of the event group before the specified bits were cleared.

EventBits_t **xEventGroupSetBits**(*EventGroupHandle_t* xEventGroup, const *EventBits_t* uxBitsToSet)

Set bits within an event group. This function cannot be called from an interrupt. xEventGroupSetBitsFromISR() is a version that can be called from an interrupt.

Setting bits in an event group will automatically unblock tasks that are blocked waiting for the bits.

Example usage:

```
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
{
EventBits_t uxBits;

// Set bit 0 and bit 4 in xEventGroup.
     uxBits = xEventGroupSetBits(
                          xEventGroup,    // The event group being updated.
                          BIT_0 | BIT_4 );// The bits being set.

if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) )
     {
// Both bit 0 and bit 4 remained set when the function returned.
     }
else if( ( uxBits & BIT_0 ) != 0 )
     {
// Bit 0 remained set when the function returned, but bit 4 was
// cleared.  It might be that bit 4 was cleared automatically as a
// task that was waiting for bit 4 was removed from the Blocked
// state.
     }
else if( ( uxBits & BIT_4 ) != 0 )
```

(下页继续)

(续上页)

```
    {
// Bit 4 remained set when the function returned, but bit 0 was
// cleared.  It might be that bit 0 was cleared automatically as a
// task that was waiting for bit 0 was removed from the Blocked
// state.
    }
else
    {
// Neither bit 0 nor bit 4 remained set.  It might be that a task
// was waiting for both of the bits to be set, and the bits were
// cleared as the task left the Blocked state.
    }
}
```

参数

- **xEventGroup** -- The event group in which the bits are to be set.
- **uxBitsToSet** -- A bitwise value that indicates the bit or bits to set. For example, to set bit 3 only, set uxBitsToSet to 0x08. To set bit 3 and bit 0 set uxBitsToSet to 0x09.

返回 The value of the event group at the time the call to xEventGroupSetBits() returns. There are two reasons why the returned value might have the bits specified by the uxBitsToSet parameter cleared. First, if setting a bit results in a task that was waiting for the bit leaving the blocked state then it is possible the bit will be cleared automatically (see the xClearBitOnExit parameter of xEventGroupWaitBits()). Second, any unblocked (or otherwise Ready state) task that has a priority above that of the task that called xEventGroupSetBits() will execute and may change the event group value before the call to xEventGroupSetBits() returns.

EventBits_t **xEventGroupSync**(*EventGroupHandle_t* xEventGroup, const *EventBits_t* uxBitsToSet, const *EventBits_t* uxBitsToWaitFor, TickType_t xTicksToWait)

Atomically set bits within an event group, then wait for a combination of bits to be set within the same event group. This functionality is typically used to synchronise multiple tasks, where each task has to wait for the other tasks to reach a synchronisation point before proceeding.

This function cannot be used from an interrupt.

The function will return before its block time expires if the bits specified by the uxBitsToWait parameter are set, or become set within that time. In this case all the bits specified by uxBitsToWait will be automatically cleared before the function returns.

Example usage:

```
// Bits used by the three tasks.
#define TASK_0_BIT     ( 1 << 0 )
#define TASK_1_BIT     ( 1 << 1 )
#define TASK_2_BIT     ( 1 << 2 )

#define ALL_SYNC_BITS ( TASK_0_BIT | TASK_1_BIT | TASK_2_BIT )

// Use an event group to synchronise three tasks.  It is assumed this event
// group has already been created elsewhere.
EventGroupHandle_t xEventBits;

void vTask0( void *pvParameters )
{
EventBits_t uxReturn;
TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;

for( ;; )
  {
```

(下页继续)

(续上页)

```
// Perform task functionality here.

// Set bit 0 in the event flag to note this task has reached the
// sync point.  The other two tasks will set the other two bits defined
// by ALL_SYNC_BITS.  All three tasks have reached the synchronisation
// point when all the ALL_SYNC_BITS are set.  Wait a maximum of 100ms
// for this to happen.
     uxReturn = xEventGroupSync( xEventBits, TASK_0_BIT, ALL_SYNC_BITS,␣
↪xTicksToWait );

if( ( uxReturn & ALL_SYNC_BITS ) == ALL_SYNC_BITS )
     {
// All three tasks reached the synchronisation point before the call
// to xEventGroupSync() timed out.
     }
 }
}

void vTask1( void *pvParameters )
{
for( ;; )
  {
// Perform task functionality here.

// Set bit 1 in the event flag to note this task has reached the
// synchronisation point.  The other two tasks will set the other two
// bits defined by ALL_SYNC_BITS.  All three tasks have reached the
// synchronisation point when all the ALL_SYNC_BITS are set.  Wait
// indefinitely for this to happen.
     xEventGroupSync( xEventBits, TASK_1_BIT, ALL_SYNC_BITS, portMAX_DELAY );

// xEventGroupSync() was called with an indefinite block time, so
// this task will only reach here if the synchronisation was made by all
// three tasks, so there is no need to test the return value.
  }
}

void vTask2( void *pvParameters )
{
for( ;; )
  {
// Perform task functionality here.

// Set bit 2 in the event flag to note this task has reached the
// synchronisation point.  The other two tasks will set the other two
// bits defined by ALL_SYNC_BITS.  All three tasks have reached the
// synchronisation point when all the ALL_SYNC_BITS are set.  Wait
// indefinitely for this to happen.
     xEventGroupSync( xEventBits, TASK_2_BIT, ALL_SYNC_BITS, portMAX_DELAY );

// xEventGroupSync() was called with an indefinite block time, so
// this task will only reach here if the synchronisation was made by all
// three tasks, so there is no need to test the return value.
 }
}
```

参数

- **xEventGroup** -- The event group in which the bits are being tested. The event group must have previously been created using a call to xEventGroupCreate().
- **uxBitsToSet** -- The bits to set in the event group before determining if, and possibly waiting for, all the bits specified by the uxBitsToWait parameter are set.

- **uxBitsToWaitFor** -- A bitwise value that indicates the bit or bits to test inside the event group. For example, to wait for bit 0 and bit 2 set uxBitsToWaitFor to 0x05. To wait for bits 0 and bit 1 and bit 2 set uxBitsToWaitFor to 0x07. Etc.
- **xTicksToWait** -- The maximum amount of time (specified in 'ticks') to wait for all of the bits specified by uxBitsToWaitFor to become set.

返回 The value of the event group at the time either the bits being waited for became set, or the block time expired. Test the return value to know which bits were set. If xEventGroupSync() returned because its timeout expired then not all the bits being waited for will be set. If xEventGroupSync() returned because all the bits it was waiting for were set then the returned value is the event group value before any bits were automatically cleared.

EventBits_t **xEventGroupGetBitsFromISR**(*EventGroupHandle_t* xEventGroup)

A version of xEventGroupGetBits() that can be called from an ISR.

参数 **xEventGroup** -- The event group being queried.

返回 The event group bits at the time xEventGroupGetBitsFromISR() was called.

void **vEventGroupDelete**(*EventGroupHandle_t* xEventGroup)

Delete an event group that was previously created by a call to xEventGroupCreate(). Tasks that are blocked on the event group will be unblocked and obtain 0 as the event group's value.

参数 **xEventGroup** -- The event group being deleted.

BaseType_t **xEventGroupGetStaticBuffer**(*EventGroupHandle_t* xEventGroup, StaticEventGroup_t **ppxEventGroupBuffer)

Retrieve a pointer to a statically created event groups's data structure buffer. It is the same buffer that is supplied at the time of creation.

参数

- **xEventGroup** -- The event group for which to retrieve the buffer.
- **ppxEventGroupBuffer** -- Used to return a pointer to the event groups's data structure buffer.

返回 pdTRUE if the buffer was retrieved, pdFALSE otherwise.

Macros

xEventGroupClearBitsFromISR(xEventGroup, uxBitsToClear)

A version of xEventGroupClearBits() that can be called from an interrupt.

Setting bits in an event group is not a deterministic operation because there are an unknown number of tasks that may be waiting for the bit or bits being set. FreeRTOS does not allow nondeterministic operations to be performed while interrupts are disabled, so protects event groups that are accessed from tasks by suspending the scheduler rather than disabling interrupts. As a result event groups cannot be accessed directly from an interrupt service routine. Therefore xEventGroupClearBitsFromISR() sends a message to the timer task to have the clear operation performed in the context of the timer task.

Example usage:

```
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

// An event group which it is assumed has already been created by a call to
// xEventGroupCreate().
EventGroupHandle_t xEventGroup;

void anInterruptHandler( void )
{
// Clear bit 0 and bit 4 in xEventGroup.
     xResult = xEventGroupClearBitsFromISR(
                          xEventGroup,     // The event group being updated.
```

(下页继续)

(续上页)

```
                          BIT_0 | BIT_4 ); // The bits being set.

if( xResult == pdPASS )
      {
// The message was posted successfully.
          portYIELD_FROM_ISR(pdTRUE);
      }
}
```

备 注: If this function returns pdPASS then the timer task is ready to run and a portYIELD_FROM_ISR(pdTRUE) should be executed to perform the needed clear on the event group. This behavior is different from xEventGroupSetBitsFromISR because the parameter xHigherPriorityTaskWoken is not present.

参数

- **xEventGroup** -- The event group in which the bits are to be cleared.
- **uxBitsToClear** -- A bitwise value that indicates the bit or bits to clear. For example, to clear bit 3 only, set uxBitsToClear to 0x08. To clear bit 3 and bit 0 set uxBitsToClear to 0x09.

返回 If the request to execute the function was posted successfully then pdPASS is returned, otherwise pdFALSE is returned. pdFALSE will be returned if the timer service queue was full.

xEventGroupSetBitsFromISR(xEventGroup, uxBitsToSet, pxHigherPriorityTaskWoken)

A version of xEventGroupSetBits() that can be called from an interrupt.

Setting bits in an event group is not a deterministic operation because there are an unknown number of tasks that may be waiting for the bit or bits being set. FreeRTOS does not allow nondeterministic operations to be performed in interrupts or from critical sections. Therefore xEventGroupSetBitsFromISR() sends a message to the timer task to have the set operation performed in the context of the timer task - where a scheduler lock is used in place of a critical section.

Example usage:

```
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

// An event group which it is assumed has already been created by a call to
// xEventGroupCreate().
EventGroupHandle_t xEventGroup;

void anInterruptHandler( void )
{
BaseType_t xHigherPriorityTaskWoken, xResult;

// xHigherPriorityTaskWoken must be initialised to pdFALSE.
     xHigherPriorityTaskWoken = pdFALSE;

// Set bit 0 and bit 4 in xEventGroup.
     xResult = xEventGroupSetBitsFromISR(
                          xEventGroup,    // The event group being updated.
                          BIT_0 | BIT_4   // The bits being set.
                          &xHigherPriorityTaskWoken );

// Was the message posted successfully?
if( xResult == pdPASS )
     {
// If xHigherPriorityTaskWoken is now set to pdTRUE then a context
```

(下页继续)

(续上页)

```
// switch should be requested.  The macro used is port specific and
// will be either portYIELD_FROM_ISR() or portEND_SWITCHING_ISR() -
// refer to the documentation page for the port being used.
        portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
    }
}
```

参数

- **xEventGroup** -- The event group in which the bits are to be set.
- **uxBitsToSet** -- A bitwise value that indicates the bit or bits to set. For example, to set bit 3 only, set uxBitsToSet to 0x08. To set bit 3 and bit 0 set uxBitsToSet to 0x09.
- **pxHigherPriorityTaskWoken** -- As mentioned above, calling this function will result in a message being sent to the timer daemon task. If the priority of the timer daemon task is higher than the priority of the currently running task (the task the interrupt interrupted) then *pxHigherPriorityTaskWoken will be set to pdTRUE by xEventGroupSetBitsFromISR(), indicating that a context switch should be requested before the interrupt exits. For that reason *pxHigherPriorityTaskWoken must be initialised to pdFALSE. See the example code below.

返回 If the request to execute the function was posted successfully then pdPASS is returned, otherwise pdFALSE is returned. pdFALSE will be returned if the timer service queue was full.

xEventGroupGetBits（xEventGroup）

Returns the current value of the bits in an event group. This function cannot be used from an interrupt.

参数

- **xEventGroup** -- The event group being queried.

返回 The event group bits at the time xEventGroupGetBits() was called.

Type Definitions

typedef struct EventGroupDef_t ***EventGroupHandle_t**

typedef TickType_t **EventBits_t**

流缓冲区 API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/stream_buffer.h
- This header file can be included with:

```
#include "freertos/stream_buffer.h"
```

Functions

BaseType_t **xStreamBufferGetStaticBuffers**(*StreamBufferHandle_t* xStreamBuffer, uint8_t **ppucStreamBufferStorageArea, StaticStreamBuffer_t **ppxStaticStreamBuffer)

Retrieve pointers to a statically created stream buffer's data structure buffer and storage area buffer. These are the same buffers that are supplied at the time of creation.

参数

- **xStreamBuffer** -- The stream buffer for which to retrieve the buffers.
- **ppucStreamBufferStorageArea** -- Used to return a pointer to the stream buffer's storage area buffer.

- **ppxStaticStreamBuffer** -- Used to return a pointer to the stream buffer's data structure buffer.

返回 pdTRUE if buffers were retrieved, pdFALSE otherwise.

size_t **xStreamBufferSend**(*StreamBufferHandle_t* xStreamBuffer, const void *pvTxData, size_t xDataLengthBytes, TickType_t xTicksToWait)

Sends bytes to a stream buffer. The bytes are copied into the stream buffer.

NOTE: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xStreamBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xStreamBufferReceive()) inside a critical section and set the receive block time to 0.

Use xStreamBufferSend() to write to a stream buffer from a task. Use xStreamBufferSendFromISR() to write to a stream buffer from an interrupt service routine (ISR).

Example use:

```
void vAFunction( StreamBufferHandle_t xStreamBuffer )
{
size_t xBytesSent;
uint8_t ucArrayToSend[] = { 0, 1, 2, 3 };
char *pcStringToSend = "String to send";
const TickType_t x100ms = pdMS_TO_TICKS( 100 );

 // Send an array to the stream buffer, blocking for a maximum of 100ms to
 // wait for enough space to be available in the stream buffer.
 xBytesSent = xStreamBufferSend( xStreamBuffer, ( void * ) ucArrayToSend,␣
↪sizeof( ucArrayToSend ), x100ms );

 if( xBytesSent != sizeof( ucArrayToSend ) )
 {
// The call to xStreamBufferSend() times out before there was enough
// space in the buffer for the data to be written, but it did
// successfully write xBytesSent bytes.
 }

 // Send the string to the stream buffer.  Return immediately if there is not
 // enough space in the buffer.
 xBytesSent = xStreamBufferSend( xStreamBuffer, ( void * ) pcStringToSend,␣
↪strlen( pcStringToSend ), 0 );

 if( xBytesSent != strlen( pcStringToSend ) )
 {
// The entire string could not be added to the stream buffer because
// there was not enough free space in the buffer, but xBytesSent bytes
// were sent.  Could try again to send the remaining bytes.
 }
}
```

参数

- **xStreamBuffer** -- The handle of the stream buffer to which a stream is being sent.
- **pvTxData** -- A pointer to the buffer that holds the bytes to be copied into the stream buffer.
- **xDataLengthBytes** -- The maximum number of bytes to copy from pvTxData into the stream buffer.

- **xTicksToWait** -- The maximum amount of time the task should remain in the Blocked state to wait for enough space to become available in the stream buffer, should the stream buffer contain too little space to hold the another xDataLengthBytes bytes. The block time is specified in tick periods, so the absolute time it represents is dependent on the tick frequency. The macro pdMS_TO_TICKS() can be used to convert a time specified in milliseconds into a time specified in ticks. Setting xTicksToWait to portMAX_DELAY will cause the task to wait indefinitely (without timing out), provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h. If a task times out before it can write all xDataLengthBytes into the buffer it will still write as many bytes as possible. A task does not use any CPU time when it is in the blocked state.

返回 The number of bytes written to the stream buffer. If a task times out before it can write all xDataLengthBytes into the buffer it will still write as many bytes as possible.

size_t **xStreamBufferSendFromISR**(*StreamBufferHandle_t* xStreamBuffer, const void *pvTxData, size_t xDataLengthBytes, BaseType_t *const pxHigherPriorityTaskWoken)

Interrupt safe version of the API function that sends a stream of bytes to the stream buffer.

NOTE: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xStreamBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xStreamBufferReceive()) inside a critical section and set the receive block time to 0.

Use xStreamBufferSend() to write to a stream buffer from a task. Use xStreamBufferSendFromISR() to write to a stream buffer from an interrupt service routine (ISR).

Example use:

```
// A stream buffer that has already been created.
StreamBufferHandle_t xStreamBuffer;

void vAnInterruptServiceRoutine( void )
{
size_t xBytesSent;
char *pcStringToSend = "String to send";
BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Initialised to pdFALSE.

 // Attempt to send the string to the stream buffer.
 xBytesSent = xStreamBufferSendFromISR( xStreamBuffer,
                                        ( void * ) pcStringToSend,
                                        strlen( pcStringToSend ),
                                        &xHigherPriorityTaskWoken );

 if( xBytesSent != strlen( pcStringToSend ) )
 {
// There was not enough free space in the stream buffer for the entire
// string to be written, ut xBytesSent bytes were written.
 }

 // If xHigherPriorityTaskWoken was set to pdTRUE inside
 // xStreamBufferSendFromISR() then a task that has a priority above the
 // priority of the currently executing task was unblocked and a context
 // switch should be performed to ensure the ISR returns to the unblocked
 // task.  In most FreeRTOS ports this is done by simply passing
 // xHigherPriorityTaskWoken into portYIELD_FROM_ISR(), which will test the
```

(下页继续)

(续上页)

```
 // variables value, and perform the context switch if necessary.  Check the
 // documentation for the port in use for port specific instructions.
 portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

参数

- **xStreamBuffer** -- The handle of the stream buffer to which a stream is being sent.
- **pvTxData** -- A pointer to the data that is to be copied into the stream buffer.
- **xDataLengthBytes** -- The maximum number of bytes to copy from pvTxData into the stream buffer.
- **pxHigherPriorityTaskWoken** -- It is possible that a stream buffer will have a task blocked on it waiting for data. Calling xStreamBufferSendFromISR() can make data available, and so cause a task that was waiting for data to leave the Blocked state. If calling xStreamBufferSendFromISR() causes a task to leave the Blocked state, and the unblocked task has a priority higher than the currently executing task (the task that was interrupted), then, internally, xStreamBufferSendFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE. If xStreamBufferSendFromISR() sets this value to pdTRUE, then normally a context switch should be performed before the interrupt is exited. This will ensure that the interrupt returns directly to the highest priority Ready state task. *pxHigherPriorityTaskWoken should be set to pdFALSE before it is passed into the function. See the example code below for an example.

返回 The number of bytes actually written to the stream buffer, which will be less than xDataLengthBytes if the stream buffer didn't have enough free space for all the bytes to be written.

size_t **xStreamBufferReceive**(*StreamBufferHandle_t* xStreamBuffer, void *pvRxData, size_t xBufferLengthBytes, TickType_t xTicksToWait)

Receives bytes from a stream buffer.

NOTE: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xStreamBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xStreamBufferReceive()) inside a critical section and set the receive block time to 0.

Use xStreamBufferReceive() to read from a stream buffer from a task. Use xStreamBufferReceiveFromISR() to read from a stream buffer from an interrupt service routine (ISR).

Example use:

```
void vAFunction( StreamBuffer_t xStreamBuffer )
{
uint8_t ucRxData[ 20 ];
size_t xReceivedBytes;
const TickType_t xBlockTime = pdMS_TO_TICKS( 20 );

 // Receive up to another sizeof( ucRxData ) bytes from the stream buffer.
 // Wait in the Blocked state (so not using any CPU processing time) for a
 // maximum of 100ms for the full sizeof( ucRxData ) number of bytes to be
 // available.
 xReceivedBytes = xStreamBufferReceive( xStreamBuffer,
                                        ( void * ) ucRxData,
sizeof( ucRxData ),
                                        xBlockTime );
```

(下页继续)

(续上页)

```
 if( xReceivedBytes > 0 )
 {
// A ucRxData contains another xReceivedBytes bytes of data, which can
// be processed here....
 }
}
```

参数

- **xStreamBuffer** -- The handle of the stream buffer from which bytes are to be received.
- **pvRxData** -- A pointer to the buffer into which the received bytes will be copied.
- **xBufferLengthBytes** -- The length of the buffer pointed to by the pvRxData parameter. This sets the maximum number of bytes to receive in one call. xStreamBufferReceive will return as many bytes as possible up to a maximum set by xBufferLengthBytes.
- **xTicksToWait** -- The maximum amount of time the task should remain in the Blocked state to wait for data to become available if the stream buffer is empty. xStreamBufferReceive() will return immediately if xTicksToWait is zero. The block time is specified in tick periods, so the absolute time it represents is dependent on the tick frequency. The macro pdMS_TO_TICKS() can be used to convert a time specified in milliseconds into a time specified in ticks. Setting xTicksToWait to portMAX_DELAY will cause the task to wait indefinitely (without timing out), provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h. A task does not use any CPU time when it is in the Blocked state.

返回 The number of bytes actually read from the stream buffer, which will be less than xBufferLengthBytes if the call to xStreamBufferReceive() timed out before xBufferLengthBytes were available.

size_t **xStreamBufferReceiveFromISR**(*StreamBufferHandle_t* xStreamBuffer, void *pvRxData, size_t xBufferLengthBytes, BaseType_t *const pxHigherPriorityTaskWoken)

An interrupt safe version of the API function that receives bytes from a stream buffer.

Use xStreamBufferReceive() to read bytes from a stream buffer from a task. Use xStreamBufferReceiveFromISR() to read bytes from a stream buffer from an interrupt service routine (ISR).

Example use:

```
// A stream buffer that has already been created.
StreamBuffer_t xStreamBuffer;

void vAnInterruptServiceRoutine( void )
{
uint8_t ucRxData[ 20 ];
size_t xReceivedBytes;
BaseType_t xHigherPriorityTaskWoken = pdFALSE;  // Initialised to pdFALSE.

 // Receive the next stream from the stream buffer.
 xReceivedBytes = xStreamBufferReceiveFromISR( xStreamBuffer,
                                               ( void * ) ucRxData,
sizeof( ucRxData ),
                                               &xHigherPriorityTaskWoken );

 if( xReceivedBytes > 0 )
 {
// ucRxData contains xReceivedBytes read from the stream buffer.
// Process the stream here....
 }
```

(下页继续)

(续上页)

```
 // If xHigherPriorityTaskWoken was set to pdTRUE inside
 // xStreamBufferReceiveFromISR() then a task that has a priority above the
 // priority of the currently executing task was unblocked and a context
 // switch should be performed to ensure the ISR returns to the unblocked
 // task.  In most FreeRTOS ports this is done by simply passing
 // xHigherPriorityTaskWoken into portYIELD_FROM_ISR(), which will test the
 // variables value, and perform the context switch if necessary.  Check the
 // documentation for the port in use for port specific instructions.
 portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

参数

- **xStreamBuffer** -- The handle of the stream buffer from which a stream is being received.
- **pvRxData** -- A pointer to the buffer into which the received bytes are copied.
- **xBufferLengthBytes** -- The length of the buffer pointed to by the pvRxData parameter. This sets the maximum number of bytes to receive in one call. xStreamBufferReceive will return as many bytes as possible up to a maximum set by xBufferLengthBytes.
- **pxHigherPriorityTaskWoken** -- It is possible that a stream buffer will have a task blocked on it waiting for space to become available. Calling xStreamBufferReceiveFromISR() can make space available, and so cause a task that is waiting for space to leave the Blocked state. If calling xStreamBufferReceiveFromISR() causes a task to leave the Blocked state, and the unblocked task has a priority higher than the currently executing task (the task that was interrupted), then, internally, xStreamBufferReceiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE. If xStreamBufferReceiveFromISR() sets this value to pdTRUE, then normally a context switch should be performed before the interrupt is exited. That will ensure the interrupt returns directly to the highest priority Ready state task. *pxHigherPriorityTaskWoken should be set to pdFALSE before it is passed into the function. See the code example below for an example.

返回 The number of bytes read from the stream buffer, if any.

void **vStreamBufferDelete**(*StreamBufferHandle_t* xStreamBuffer)

Deletes a stream buffer that was previously created using a call to xStreamBufferCreate() or xStreamBufferCreateStatic(). If the stream buffer was created using dynamic memory (that is, by xStreamBufferCreate()), then the allocated memory is freed.

A stream buffer handle must not be used after the stream buffer has been deleted.

参数 **xStreamBuffer** -- The handle of the stream buffer to be deleted.

BaseType_t **xStreamBufferIsFull**(*StreamBufferHandle_t* xStreamBuffer)

Queries a stream buffer to see if it is full. A stream buffer is full if it does not have any free space, and therefore cannot accept any more data.

参数 **xStreamBuffer** -- The handle of the stream buffer being queried.

返回 If the stream buffer is full then pdTRUE is returned. Otherwise pdFALSE is returned.

BaseType_t **xStreamBufferIsEmpty**(*StreamBufferHandle_t* xStreamBuffer)

Queries a stream buffer to see if it is empty. A stream buffer is empty if it does not contain any data.

参数 **xStreamBuffer** -- The handle of the stream buffer being queried.

返回 If the stream buffer is empty then pdTRUE is returned. Otherwise pdFALSE is returned.

BaseType_t **xStreamBufferReset**(*StreamBufferHandle_t* xStreamBuffer)

Resets a stream buffer to its initial, empty, state. Any data that was in the stream buffer is discarded. A stream buffer can only be reset if there are no tasks blocked waiting to either send to or receive from the stream buffer.

参数 **xStreamBuffer** -- The handle of the stream buffer being reset.

返回 If the stream buffer is reset then pdPASS is returned. If there was a task blocked waiting to send to or read from the stream buffer then the stream buffer is not reset and pdFAIL is returned.

size_t **xStreamBufferSpacesAvailable**(*StreamBufferHandle_t* xStreamBuffer)

Queries a stream buffer to see how much free space it contains, which is equal to the amount of data that can be sent to the stream buffer before it is full.

参数 **xStreamBuffer** -- The handle of the stream buffer being queried.

返回 The number of bytes that can be written to the stream buffer before the stream buffer would be full.

size_t **xStreamBufferBytesAvailable**(*StreamBufferHandle_t* xStreamBuffer)

Queries a stream buffer to see how much data it contains, which is equal to the number of bytes that can be read from the stream buffer before the stream buffer would be empty.

参数 **xStreamBuffer** -- The handle of the stream buffer being queried.

返回 The number of bytes that can be read from the stream buffer before the stream buffer would be empty.

BaseType_t **xStreamBufferSetTriggerLevel**(*StreamBufferHandle_t* xStreamBuffer, size_t xTriggerLevel)

A stream buffer's trigger level is the number of bytes that must be in the stream buffer before a task that is blocked on the stream buffer to wait for data is moved out of the blocked state. For example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 1 then the task will be unblocked when a single byte is written to the buffer or the task's block time expires. As another example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 10 then the task will not be unblocked until the stream buffer contains at least 10 bytes or the task's block time expires. If a reading task's block time expires before the trigger level is reached then the task will still receive however many bytes are actually available. Setting a trigger level of 0 will result in a trigger level of 1 being used. It is not valid to specify a trigger level that is greater than the buffer size.

A trigger level is set when the stream buffer is created, and can be modified using xStreamBufferSetTriggerLevel().

参数

- **xStreamBuffer** -- The handle of the stream buffer being updated.
- **xTriggerLevel** -- The new trigger level for the stream buffer.

返回 If xTriggerLevel was less than or equal to the stream buffer's length then the trigger level will be updated and pdTRUE is returned. Otherwise pdFALSE is returned.

BaseType_t **xStreamBufferSendCompletedFromISR**(*StreamBufferHandle_t* xStreamBuffer, BaseType_t *pxHigherPriorityTaskWoken)

For advanced users only.

The sbSEND_COMPLETED() macro is called from within the FreeRTOS APIs when data is sent to a message buffer or stream buffer. If there was a task that was blocked on the message or stream buffer waiting for data to arrive then the sbSEND_COMPLETED() macro sends a notification to the task to remove it from the Blocked state. xStreamBufferSendCompletedFromISR() does the same thing. It is provided to enable application writers to implement their own version of sbSEND_COMPLETED(), and MUST NOT BE USED AT ANY OTHER TIME.

See the example implemented in FreeRTOS/Demo/Minimal/MessageBufferAMP.c for additional information.

参数

- **xStreamBuffer** -- The handle of the stream buffer to which data was written.
- **pxHigherPriorityTaskWoken** -- *pxHigherPriorityTaskWoken should be initialised to pdFALSE before it is passed into xStreamBufferSendCompletedFromISR(). If calling xStreamBufferSendCompletedFromISR() removes a task from the Blocked state, and the task has a priority above the priority of the currently running task, then *pxHigherPriorityTaskWoken will get set to pdTRUE indicating that a context switch should be performed before exiting the ISR.

返回 If a task was removed from the Blocked state then pdTRUE is returned. Otherwise pdFALSE is returned.

BaseType_t **xStreamBufferReceiveCompletedFromISR**(*StreamBufferHandle_t* xStreamBuffer, BaseType_t *pxHigherPriorityTaskWoken)

For advanced users only.

The sbRECEIVE_COMPLETED() macro is called from within the FreeRTOS APIs when data is read out of a message buffer or stream buffer. If there was a task that was blocked on the message or stream buffer waiting for data to arrive then the sbRECEIVE_COMPLETED() macro sends a notification to the task to remove it from the Blocked state. xStreamBufferReceiveCompletedFromISR() does the same thing. It is provided to enable application writers to implement their own version of sbRECEIVE_COMPLETED(), and MUST NOT BE USED AT ANY OTHER TIME.

See the example implemented in FreeRTOS/Demo/Minimal/MessageBufferAMP.c for additional information.

参数

- **xStreamBuffer** -- The handle of the stream buffer from which data was read.
- **pxHigherPriorityTaskWoken** -- *pxHigherPriorityTaskWoken should be initialised to pdFALSE before it is passed into xStreamBufferReceiveCompletedFromISR(). If calling xStreamBufferReceiveCompletedFromISR() removes a task from the Blocked state, and the task has a priority above the priority of the currently running task, then *pxHigherPriorityTaskWoken will get set to pdTRUE indicating that a context switch should be performed before exiting the ISR.

返回 If a task was removed from the Blocked state then pdTRUE is returned. Otherwise pdFALSE is returned.

Macros

xStreamBufferCreateWithCallback(xBufferSizeBytes, xTriggerLevelBytes, pxSendCompletedCallback, pxReceiveCompletedCallback)

Creates a new stream buffer using dynamically allocated memory. See xStreamBufferCreateStatic() for a version that uses statically allocated memory (memory that is allocated at compile time).

configSUPPORT_DYNAMIC_ALLOCATION must be set to 1 or left undefined in FreeRTOSConfig.h for xStreamBufferCreate() to be available.

Example use:

```
void vAFunction( void )
{
StreamBufferHandle_t xStreamBuffer;
const size_t xStreamBufferSizeBytes = 100, xTriggerLevel = 10;

 // Create a stream buffer that can hold 100 bytes.  The memory used to hold
 // both the stream buffer structure and the data in the stream buffer is
 // allocated dynamically.
 xStreamBuffer = xStreamBufferCreate( xStreamBufferSizeBytes, xTriggerLevel );

 if( xStreamBuffer == NULL )
 {
// There was not enough heap memory space available to create the
// stream buffer.
 }
 else
 {
// The stream buffer was created successfully and can now be used.
 }
}
```

参数

- **xBufferSizeBytes** -- The total number of bytes the stream buffer will be able to hold at any one time.
- **xTriggerLevelBytes** -- The number of bytes that must be in the stream buffer before a task that is blocked on the stream buffer to wait for data is moved out of the blocked state. For example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 1 then the task will be unblocked when a single byte is written to the buffer or the task's block time expires. As another example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 10 then the task will not be unblocked until the stream buffer contains at least 10 bytes or the task's block time expires. If a reading task's block time expires before the trigger level is reached then the task will still receive however many bytes are actually available. Setting a trigger level of 0 will result in a trigger level of 1 being used. It is not valid to specify a trigger level that is greater than the buffer size.
- **pxSendCompletedCallback** -- Callback invoked when number of bytes at least equal to trigger level is sent to the stream buffer. If the parameter is NULL, it will use the default implementation provided by sbSEND_COMPLETED macro. To enable the callback, configUSE_SB_COMPLETED_CALLBACK must be set to 1 in FreeRTOSConfig.h.
- **pxReceiveCompletedCallback** -- Callback invoked when more than zero bytes are read from a stream buffer. If the parameter is NULL, it will use the default implementation provided by sbRECEIVE_COMPLETED macro. To enable the callback, configUSE_SB_COMPLETED_CALLBACK must be set to 1 in FreeRTOSConfig.h.

返回 If NULL is returned, then the stream buffer cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the stream buffer data structures and storage area. A non-NULL value being returned indicates that the stream buffer has been created successfully - the returned value should be stored as the handle to the created stream buffer.

xStreamBufferCreateStaticWithCallback(xBufferSizeBytes, xTriggerLevelBytes, pucStreamBufferStorageArea, pxStaticStreamBuffer, pxSendCompletedCallback, pxReceiveCompletedCallback)

Creates a new stream buffer using statically allocated memory. See xStreamBufferCreate() for a version that uses dynamically allocated memory.

configSUPPORT_STATIC_ALLOCATION must be set to 1 in FreeRTOSConfig.h for xStreamBufferCreateStatic() to be available.

Example use:

```
// Used to dimension the array used to hold the streams.  The available space
// will actually be one less than this, so 999.
#define STORAGE_SIZE_BYTES 1000

// Defines the memory that will actually hold the streams within the stream
// buffer.
static uint8_t ucStorageBuffer[ STORAGE_SIZE_BYTES ];

// The variable used to hold the stream buffer structure.
  StaticStreamBuffer_t xStreamBufferStruct;

void MyFunction( void )
  {
  StreamBufferHandle_t xStreamBuffer;
const size_t xTriggerLevel = 1;

   xStreamBuffer = xStreamBufferCreateStatic( sizeof( ucStorageBuffer ),
                                              xTriggerLevel,
                                              ucStorageBuffer,
                                              &xStreamBufferStruct );
```

(下页继续)

(续上页)

```
// As neither the pucStreamBufferStorageArea or pxStaticStreamBuffer
// parameters were NULL, xStreamBuffer will not be NULL, and can be used to
// reference the created stream buffer in other stream buffer API calls.

// Other code that uses the stream buffer can go here.
  }
```

参数

- **xBufferSizeBytes** -- The size, in bytes, of the buffer pointed to by the pucStreamBufferStorageArea parameter.
- **xTriggerLevelBytes** -- The number of bytes that must be in the stream buffer before a task that is blocked on the stream buffer to wait for data is moved out of the blocked state. For example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 1 then the task will be unblocked when a single byte is written to the buffer or the task's block time expires. As another example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 10 then the task will not be unblocked until the stream buffer contains at least 10 bytes or the task's block time expires. If a reading task's block time expires before the trigger level is reached then the task will still receive however many bytes are actually available. Setting a trigger level of 0 will result in a trigger level of 1 being used. It is not valid to specify a trigger level that is greater than the buffer size.
- **pucStreamBufferStorageArea** -- Must point to a uint8_t array that is at least xBufferSizeBytes big. This is the array to which streams are copied when they are written to the stream buffer.
- **pxStaticStreamBuffer** -- Must point to a variable of type StaticStreamBuffer_t, which will be used to hold the stream buffer's data structure.
- **pxSendCompletedCallback** -- Callback invoked when number of bytes at least equal to trigger level is sent to the stream buffer. If the parameter is NULL, it will use the default implementation provided by sbSEND_COMPLETED macro. To enable the callback, configUSE_SB_COMPLETED_CALLBACK must be set to 1 in FreeRTOSConfig.h.
- **pxReceiveCompletedCallback** -- Callback invoked when more than zero bytes are read from a stream buffer. If the parameter is NULL, it will use the default implementation provided by sbRECEIVE_COMPLETED macro. To enable the callback, configUSE_SB_COMPLETED_CALLBACK must be set to 1 in FreeRTOSConfig.h.

返回 If the stream buffer is created successfully then a handle to the created stream buffer is returned. If either pucStreamBufferStorageArea or pxStaticstreamBuffer are NULL then NULL is returned.

Type Definitions

typedef struct StreamBufferDef_t ***StreamBufferHandle_t**

typedef void (***StreamBufferCallbackFunction_t**)(*StreamBufferHandle_t* xStreamBuffer, BaseType_t xIsInsideISR, BaseType_t *const pxHigherPriorityTaskWoken)

Type used as a stream buffer's optional callback.

消息缓冲区 API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/message_buffer.h
- This header file can be included with:

```
#include "freertos/message_buffer.h"
```

Macros

xMessageBufferCreateWithCallback(xBufferSizeBytes, pxSendCompletedCallback, pxReceiveCompletedCallback)

Creates a new message buffer using dynamically allocated memory. See xMessageBufferCreateStatic() for a version that uses statically allocated memory (memory that is allocated at compile time).

configSUPPORT_DYNAMIC_ALLOCATION must be set to 1 or left undefined in FreeRTOSConfig.h for xMessageBufferCreate() to be available.

Example use:

```
void vAFunction( void )
{
MessageBufferHandle_t xMessageBuffer;
const size_t xMessageBufferSizeBytes = 100;

 // Create a message buffer that can hold 100 bytes.  The memory used to hold
 // both the message buffer structure and the messages themselves is allocated
 // dynamically.  Each message added to the buffer consumes an additional 4
 // bytes which are used to hold the length of the message.
 xMessageBuffer = xMessageBufferCreate( xMessageBufferSizeBytes );

 if( xMessageBuffer == NULL )
 {
// There was not enough heap memory space available to create the
// message buffer.
 }
 else
 {
// The message buffer was created successfully and can now be used.
 }
```

参数

- **xBufferSizeBytes** -- The total number of bytes (not messages) the message buffer will be able to hold at any one time. When a message is written to the message buffer an additional sizeof(size_t) bytes are also written to store the message's length. sizeof(size_t) is typically 4 bytes on a 32-bit architecture, so on most 32-bit architectures a 10 byte message will take up 14 bytes of message buffer space.
- **pxSendCompletedCallback** -- Callback invoked when a send operation to the message buffer is complete. If the parameter is NULL or xMessageBufferCreate() is called without the parameter, then it will use the default implementation provided by sbSEND_COMPLETED macro. To enable the callback, configUSE_SB_COMPLETED_CALLBACK must be set to 1 in FreeRTOSConfig.h.
- **pxReceiveCompletedCallback** -- Callback invoked when a receive operation from the message buffer is complete. If the parameter is NULL or xMessageBufferCreate() is called without the parameter, it will use the default implementation provided by sbRECEIVE_COMPLETED macro. To enable the callback, configUSE_SB_COMPLETED_CALLBACK must be set to 1 in FreeRTOSConfig.h.

返回 If NULL is returned, then the message buffer cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the message buffer data structures and storage area. A non-NULL value being returned indicates that the message buffer has been created successfully - the returned value should be stored as the handle to the created message buffer.

xMessageBufferCreateStaticWithCallback(xBufferSizeBytes, pucMessageBufferStorageArea, pxStaticMessageBuffer, pxSendCompletedCallback, pxReceiveCompletedCallback)

Creates a new message buffer using statically allocated memory. See xMessageBufferCreate() for a version that uses dynamically allocated memory.

Example use:

```
// Used to dimension the array used to hold the messages.  The available space
// will actually be one less than this, so 999.
#define STORAGE_SIZE_BYTES 1000

// Defines the memory that will actually hold the messages within the message
// buffer.
static uint8_t ucStorageBuffer[ STORAGE_SIZE_BYTES ];

// The variable used to hold the message buffer structure.
StaticMessageBuffer_t xMessageBufferStruct;

void MyFunction( void )
{
MessageBufferHandle_t xMessageBuffer;

 xMessageBuffer = xMessageBufferCreateStatic( sizeof( ucStorageBuffer ),
                                              ucStorageBuffer,
                                              &xMessageBufferStruct );

 // As neither the pucMessageBufferStorageArea or pxStaticMessageBuffer
 // parameters were NULL, xMessageBuffer will not be NULL, and can be used to
 // reference the created message buffer in other message buffer API calls.

 // Other code that uses the message buffer can go here.
}
```

参数

- **xBufferSizeBytes** -- The size, in bytes, of the buffer pointed to by the pucMessageBufferStorageArea parameter. When a message is written to the message buffer an additional sizeof(size_t) bytes are also written to store the message's length. sizeof(size_t) is typically 4 bytes on a 32-bit architecture, so on most 32-bit architecture a 10 byte message will take up 14 bytes of message buffer space. The maximum number of bytes that can be stored in the message buffer is actually (xBufferSizeBytes - 1).
- **pucMessageBufferStorageArea** -- Must point to a uint8_t array that is at least xBufferSizeBytes big. This is the array to which messages are copied when they are written to the message buffer.
- **pxStaticMessageBuffer** -- Must point to a variable of type StaticMessageBuffer_t, which will be used to hold the message buffer's data structure.
- **pxSendCompletedCallback** -- Callback invoked when a new message is sent to the message buffer. If the parameter is NULL or xMessageBufferCreate() is called without the parameter, then it will use the default implementation provided by sbSEND_COMPLETED macro. To enable the callback, configUSE_SB_COMPLETED_CALLBACK must be set to 1 in FreeRTOSConfig.h.
- **pxReceiveCompletedCallback** -- Callback invoked when a message is read from a message buffer. If the parameter is NULL or xMessageBufferCreate() is called without the parameter, it will use the default implementation provided by sbRECEIVE_COMPLETED macro. To enable the callback, configUSE_SB_COMPLETED_CALLBACK must be set to 1 in FreeRTOSConfig.h.

返回 If the message buffer is created successfully then a handle to the created message buffer is returned. If either pucMessageBufferStorageArea or pxStaticmessageBuffer are NULL then NULL is returned.

xMessageBufferGetStaticBuffers(xMessageBuffer, ppucMessageBufferStorageArea, ppxStaticMessageBuffer)

Retrieve pointers to a statically created message buffer's data structure buffer and storage area buffer. These are the same buffers that are supplied at the time of creation.

参数

- **xMessageBuffer** -- The message buffer for which to retrieve the buffers.
- **ppucMessageBufferStorageArea** -- Used to return a pointer to the message buffer's storage area buffer.
- **ppxStaticMessageBuffer** -- Used to return a pointer to the message buffer's data structure buffer.

返回 pdTRUE if buffers were retrieved, pdFALSE otherwise.

xMessageBufferSend(xMessageBuffer, pvTxData, xDataLengthBytes, xTicksToWait)

Sends a discrete message to the message buffer. The message can be any length that fits within the buffer's free space, and is copied into the buffer.

NOTE: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xMessageBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xMessageBufferRead()) inside a critical section and set the receive block time to 0.

Use xMessageBufferSend() to write to a message buffer from a task. Use xMessageBufferSendFromISR() to write to a message buffer from an interrupt service routine (ISR).

Example use:

```
void vAFunction( MessageBufferHandle_t xMessageBuffer )
{
size_t xBytesSent;
uint8_t ucArrayToSend[] = { 0, 1, 2, 3 };
char *pcStringToSend = "String to send";
const TickType_t x100ms = pdMS_TO_TICKS( 100 );

 // Send an array to the message buffer, blocking for a maximum of 100ms to
 // wait for enough space to be available in the message buffer.
 xBytesSent = xMessageBufferSend( xMessageBuffer, ( void * ) ucArrayToSend,␣
↪sizeof( ucArrayToSend ), x100ms );

 if( xBytesSent != sizeof( ucArrayToSend ) )
 {
// The call to xMessageBufferSend() times out before there was enough
// space in the buffer for the data to be written.
 }

 // Send the string to the message buffer.  Return immediately if there is
 // not enough space in the buffer.
 xBytesSent = xMessageBufferSend( xMessageBuffer, ( void * ) pcStringToSend,␣
↪strlen( pcStringToSend ), 0 );

 if( xBytesSent != strlen( pcStringToSend ) )
 {
// The string could not be added to the message buffer because there was
// not enough free space in the buffer.
 }
}
```

参数

- **xMessageBuffer** -- The handle of the message buffer to which a message is being sent.
- **pvTxData** -- A pointer to the message that is to be copied into the message buffer.

- **xDataLengthBytes** -- The length of the message. That is, the number of bytes to copy from pvTxData into the message buffer. When a message is written to the message buffer an additional sizeof(size_t) bytes are also written to store the message's length. sizeof(size_t) is typically 4 bytes on a 32-bit architecture, so on most 32-bit architecture setting xDataLengthBytes to 20 will reduce the free space in the message buffer by 24 bytes (20 bytes of message data and 4 bytes to hold the message length).
- **xTicksToWait** -- The maximum amount of time the calling task should remain in the Blocked state to wait for enough space to become available in the message buffer, should the message buffer have insufficient space when xMessageBufferSend() is called. The calling task will never block if xTicksToWait is zero. The block time is specified in tick periods, so the absolute time it represents is dependent on the tick frequency. The macro pdMS_TO_TICKS() can be used to convert a time specified in milliseconds into a time specified in ticks. Setting xTicksToWait to portMAX_DELAY will cause the task to wait indefinitely (without timing out), provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h. Tasks do not use any CPU time when they are in the Blocked state.

返回 The number of bytes written to the message buffer. If the call to xMessageBufferSend() times out before there was enough space to write the message into the message buffer then zero is returned. If the call did not time out then xDataLengthBytes is returned.

xMessageBufferSendFromISR(xMessageBuffer, pvTxData, xDataLengthBytes, pxHigherPriorityTaskWoken)

Interrupt safe version of the API function that sends a discrete message to the message buffer. The message can be any length that fits within the buffer's free space, and is copied into the buffer.

NOTE: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xMessageBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xMessageBufferRead()) inside a critical section and set the receive block time to 0.

Use xMessageBufferSend() to write to a message buffer from a task. Use xMessageBufferSendFromISR() to write to a message buffer from an interrupt service routine (ISR).

Example use:

```
// A message buffer that has already been created.
MessageBufferHandle_t xMessageBuffer;

void vAnInterruptServiceRoutine( void )
{
size_t xBytesSent;
char *pcStringToSend = "String to send";
BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Initialised to pdFALSE.

 // Attempt to send the string to the message buffer.
 xBytesSent = xMessageBufferSendFromISR( xMessageBuffer,
                                         ( void * ) pcStringToSend,
                                         strlen( pcStringToSend ),
                                         &xHigherPriorityTaskWoken );

 if( xBytesSent != strlen( pcStringToSend ) )
 {
// The string could not be added to the message buffer because there was
// not enough free space in the buffer.
 }
```

(下页继续)

(续上页)

```
    // If xHigherPriorityTaskWoken was set to pdTRUE inside
    // xMessageBufferSendFromISR() then a task that has a priority above the
    // priority of the currently executing task was unblocked and a context
    // switch should be performed to ensure the ISR returns to the unblocked
    // task.  In most FreeRTOS ports this is done by simply passing
    // xHigherPriorityTaskWoken into portYIELD_FROM_ISR(), which will test the
    // variables value, and perform the context switch if necessary.  Check the
    // documentation for the port in use for port specific instructions.
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

参数

- **xMessageBuffer** -- The handle of the message buffer to which a message is being sent.
- **pvTxData** -- A pointer to the message that is to be copied into the message buffer.
- **xDataLengthBytes** -- The length of the message. That is, the number of bytes to copy from pvTxData into the message buffer. When a message is written to the message buffer an additional sizeof(size_t) bytes are also written to store the message's length. sizeof(size_t) is typically 4 bytes on a 32-bit architecture, so on most 32-bit architecture setting xDataLengthBytes to 20 will reduce the free space in the message buffer by 24 bytes (20 bytes of message data and 4 bytes to hold the message length).
- **pxHigherPriorityTaskWoken** -- It is possible that a message buffer will have a task blocked on it waiting for data. Calling xMessageBufferSendFromISR() can make data available, and so cause a task that was waiting for data to leave the Blocked state. If calling xMessageBufferSendFromISR() causes a task to leave the Blocked state, and the unblocked task has a priority higher than the currently executing task (the task that was interrupted), then, internally, xMessageBufferSendFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE. If xMessageBufferSendFromISR() sets this value to pdTRUE, then normally a context switch should be performed before the interrupt is exited. This will ensure that the interrupt returns directly to the highest priority Ready state task. *pxHigherPriorityTaskWoken should be set to pdFALSE before it is passed into the function. See the code example below for an example.

返回 The number of bytes actually written to the message buffer. If the message buffer didn't have enough free space for the message to be stored then 0 is returned, otherwise xDataLengthBytes is returned.

xMessageBufferReceive(xMessageBuffer, pvRxData, xBufferLengthBytes, xTicksToWait)

Receives a discrete message from a message buffer. Messages can be of variable length and are copied out of the buffer.

NOTE: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xMessageBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xMessageBufferRead()) inside a critical section and set the receive block time to 0.

Use xMessageBufferReceive() to read from a message buffer from a task. Use xMessageBufferReceiveFromISR() to read from a message buffer from an interrupt service routine (ISR).

Example use:

```
void vAFunction( MessageBuffer_t xMessageBuffer )
{
uint8_t ucRxData[ 20 ];
size_t xReceivedBytes;
const TickType_t xBlockTime = pdMS_TO_TICKS( 20 );

 // Receive the next message from the message buffer.  Wait in the Blocked
 // state (so not using any CPU processing time) for a maximum of 100ms for
 // a message to become available.
 xReceivedBytes = xMessageBufferReceive( xMessageBuffer,
                                         ( void * ) ucRxData,
sizeof( ucRxData ),
                                         xBlockTime );

 if( xReceivedBytes > 0 )
 {
// A ucRxData contains a message that is xReceivedBytes long.  Process
// the message here....
 }
}
```

参数

- **xMessageBuffer** -- The handle of the message buffer from which a message is being received.
- **pvRxData** -- A pointer to the buffer into which the received message is to be copied.
- **xBufferLengthBytes** -- The length of the buffer pointed to by the pvRxData parameter. This sets the maximum length of the message that can be received. If xBufferLengthBytes is too small to hold the next message then the message will be left in the message buffer and 0 will be returned.
- **xTicksToWait** -- The maximum amount of time the task should remain in the Blocked state to wait for a message, should the message buffer be empty. xMessageBufferReceive() will return immediately if xTicksToWait is zero and the message buffer is empty. The block time is specified in tick periods, so the absolute time it represents is dependent on the tick frequency. The macro pdMS_TO_TICKS() can be used to convert a time specified in milliseconds into a time specified in ticks. Setting xTicksToWait to portMAX_DELAY will cause the task to wait indefinitely (without timing out), provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h. Tasks do not use any CPU time when they are in the Blocked state.

返回 The length, in bytes, of the message read from the message buffer, if any. If xMessageBufferReceive() times out before a message became available then zero is returned. If the length of the message is greater than xBufferLengthBytes then the message will be left in the message buffer and zero is returned.

xMessageBufferReceiveFromISR(xMessageBuffer, pvRxData, xBufferLengthBytes, pxHigherPriorityTaskWoken)

An interrupt safe version of the API function that receives a discrete message from a message buffer. Messages can be of variable length and are copied out of the buffer.

NOTE: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xMessageBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xMessageBufferRead()) inside a critical section and set the receive block time to 0.

Use xMessageBufferReceive() to read from a message buffer from a task. Use xMessageBufferReceive-

FromISR() to read from a message buffer from an interrupt service routine (ISR).

Example use:

```
// A message buffer that has already been created.
MessageBuffer_t xMessageBuffer;

void vAnInterruptServiceRoutine( void )
{
uint8_t ucRxData[ 20 ];
size_t xReceivedBytes;
BaseType_t xHigherPriorityTaskWoken = pdFALSE;  // Initialised to pdFALSE.

 // Receive the next message from the message buffer.
 xReceivedBytes = xMessageBufferReceiveFromISR( xMessageBuffer,
                                                ( void * ) ucRxData,
sizeof( ucRxData ),
                                                &xHigherPriorityTaskWoken );

 if( xReceivedBytes > 0 )
 {
// A ucRxData contains a message that is xReceivedBytes long.  Process
// the message here....
 }

 // If xHigherPriorityTaskWoken was set to pdTRUE inside
 // xMessageBufferReceiveFromISR() then a task that has a priority above the
 // priority of the currently executing task was unblocked and a context
 // switch should be performed to ensure the ISR returns to the unblocked
 // task.  In most FreeRTOS ports this is done by simply passing
 // xHigherPriorityTaskWoken into portYIELD_FROM_ISR(), which will test the
 // variables value, and perform the context switch if necessary.  Check the
 // documentation for the port in use for port specific instructions.
 portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

参数

- **xMessageBuffer** -- The handle of the message buffer from which a message is being received.
- **pvRxData** -- A pointer to the buffer into which the received message is to be copied.
- **xBufferLengthBytes** -- The length of the buffer pointed to by the pvRxData parameter. This sets the maximum length of the message that can be received. If xBufferLengthBytes is too small to hold the next message then the message will be left in the message buffer and 0 will be returned.
- **pxHigherPriorityTaskWoken** -- It is possible that a message buffer will have a task blocked on it waiting for space to become available. Calling xMessageBufferReceiveFromISR() can make space available, and so cause a task that is waiting for space to leave the Blocked state. If calling xMessageBufferReceiveFromISR() causes a task to leave the Blocked state, and the unblocked task has a priority higher than the currently executing task (the task that was interrupted), then, internally, xMessageBufferReceiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE. If xMessageBufferReceiveFromISR() sets this value to pdTRUE, then normally a context switch should be performed before the interrupt is exited. That will ensure the interrupt returns directly to the highest priority Ready state task. *pxHigherPriorityTaskWoken should be set to pdFALSE before it is passed into the function. See the code example below for an example.

返回 The length, in bytes, of the message read from the message buffer, if any.

vMessageBufferDelete(xMessageBuffer)

Deletes a message buffer that was previously created using a call to xMessageBufferCreate() or xMessage-

BufferCreateStatic(). If the message buffer was created using dynamic memory (that is, by xMessageBufferCreate()), then the allocated memory is freed.

A message buffer handle must not be used after the message buffer has been deleted.

参数

- **xMessageBuffer** -- The handle of the message buffer to be deleted.

xMessageBufferIsFull(xMessageBuffer)

Tests to see if a message buffer is full. A message buffer is full if it cannot accept any more messages, of any size, until space is made available by a message being removed from the message buffer.

参数

- **xMessageBuffer** -- The handle of the message buffer being queried.

返回 If the message buffer referenced by xMessageBuffer is full then pdTRUE is returned. Otherwise pdFALSE is returned.

xMessageBufferIsEmpty(xMessageBuffer)

Tests to see if a message buffer is empty (does not contain any messages).

参数

- **xMessageBuffer** -- The handle of the message buffer being queried.

返回 If the message buffer referenced by xMessageBuffer is empty then pdTRUE is returned. Otherwise pdFALSE is returned.

xMessageBufferReset(xMessageBuffer)

Resets a message buffer to its initial empty state, discarding any message it contained.

A message buffer can only be reset if there are no tasks blocked on it.

参数

- **xMessageBuffer** -- The handle of the message buffer being reset.

返回 If the message buffer was reset then pdPASS is returned. If the message buffer could not be reset because either there was a task blocked on the message queue to wait for space to become available, or to wait for a a message to be available, then pdFAIL is returned.

xMessageBufferSpaceAvailable(xMessageBuffer)

message_buffer.h

```
size_t xMessageBufferSpaceAvailable( MessageBufferHandle_t xMessageBuffer );
```

Returns the number of bytes of free space in the message buffer.

参数

- **xMessageBuffer** -- The handle of the message buffer being queried.

返回 The number of bytes that can be written to the message buffer before the message buffer would be full. When a message is written to the message buffer an additional sizeof(size_t) bytes are also written to store the message's length. sizeof(size_t) is typically 4 bytes on a 32-bit architecture, so if xMessageBufferSpacesAvailable() returns 10, then the size of the largest message that can be written to the message buffer is 6 bytes.

xMessageBufferSpacesAvailable(xMessageBuffer)

xMessageBufferNextLengthBytes(xMessageBuffer)

Returns the length (in bytes) of the next message in a message buffer. Useful if xMessageBufferReceive() returned 0 because the size of the buffer passed into xMessageBufferReceive() was too small to hold the next message.

参数

- **xMessageBuffer** -- The handle of the message buffer being queried.

返回 The length (in bytes) of the next message in the message buffer, or 0 if the message buffer is empty.

xMessageBufferSendCompletedFromISR(xMessageBuffer, pxHigherPriorityTaskWoken)

For advanced users only.

The sbSEND_COMPLETED() macro is called from within the FreeRTOS APIs when data is sent to a message buffer or stream buffer. If there was a task that was blocked on the message or stream buffer waiting for data to arrive then the sbSEND_COMPLETED() macro sends a notification to the task to remove it from the Blocked state. xMessageBufferSendCompletedFromISR() does the same thing. It is provided to enable application writers to implement their own version of sbSEND_COMPLETED(), and MUST NOT BE USED AT ANY OTHER TIME.

See the example implemented in FreeRTOS/Demo/Minimal/MessageBufferAMP.c for additional information.

参数

- **xMessageBuffer** -- The handle of the stream buffer to which data was written.
- **pxHigherPriorityTaskWoken** -- *pxHigherPriorityTaskWoken should be initialised to pdFALSE before it is passed into xMessageBufferSendCompletedFromISR(). If calling xMessageBufferSendCompletedFromISR() removes a task from the Blocked state, and the task has a priority above the priority of the currently running task, then *pxHigherPriorityTaskWoken will get set to pdTRUE indicating that a context switch should be performed before exiting the ISR.

返回 If a task was removed from the Blocked state then pdTRUE is returned. Otherwise pdFALSE is returned.

xMessageBufferReceiveCompletedFromISR(xMessageBuffer, pxHigherPriorityTaskWoken)

For advanced users only.

The sbRECEIVE_COMPLETED() macro is called from within the FreeRTOS APIs when data is read out of a message buffer or stream buffer. If there was a task that was blocked on the message or stream buffer waiting for data to arrive then the sbRECEIVE_COMPLETED() macro sends a notification to the task to remove it from the Blocked state. xMessageBufferReceiveCompletedFromISR() does the same thing. It is provided to enable application writers to implement their own version of sbRECEIVE_COMPLETED(), and MUST NOT BE USED AT ANY OTHER TIME.

See the example implemented in FreeRTOS/Demo/Minimal/MessageBufferAMP.c for additional information.

参数

- **xMessageBuffer** -- The handle of the stream buffer from which data was read.
- **pxHigherPriorityTaskWoken** -- *pxHigherPriorityTaskWoken should be initialised to pdFALSE before it is passed into xMessageBufferReceiveCompletedFromISR(). If calling xMessageBufferReceiveCompletedFromISR() removes a task from the Blocked state, and the task has a priority above the priority of the currently running task, then *pxHigherPriorityTaskWoken will get set to pdTRUE indicating that a context switch should be performed before exiting the ISR.

返回 If a task was removed from the Blocked state then pdTRUE is returned. Otherwise pdFALSE is returned.

Type Definitions

typedef *StreamBufferHandle_t* **MessageBufferHandle_t**

Type by which message buffers are referenced. For example, a call to xMessageBufferCreate() returns an MessageBufferHandle_t variable that can then be used as a parameter to xMessageBufferSend(), xMessageBufferReceive(), etc. Message buffer is essentially built as a stream buffer hence its handle is also set to same type as a stream buffer handle.

2.9.13 FreeRTOS（附加功能）

ESP-IDF 为 FreeRTOS 提供了多种附加功能。这些附加功能适用于 ESP-IDF 支持的所有 FreeRTOS 实现，即 ESP-IDF FreeRTOS 和 Amazon SMP FreeRTOS。本文档介绍了这些附加功能，内容包括以下几个部分：

目录

- *FreeRTOS（附加功能）*
  - *概述*
  - *环形 buffer*
  - *ESP-IDF tick 钩子和 idle 钩子*
  - *TLSP 删除回调*
  - *IDF 附加 API*
  - *组件专用功能*
  - *API 参考*

概述

ESP-IDF 为 FreeRTOS 提供了以下附加功能：

- **环形 buffer**：FIFO 缓冲区，支持任意长度的数据项。
- **ESP-IDF tick 钩子和 idle 钩子**：ESP-IDF 提供了多个自定义的 tick 钩子和 idle 钩子，相较于 FreeRTOS，支持的钩子数量更多且更灵活。
- **线程本地存储指针 (TLSP) 删除回调**：当一个任务被删除时，TLSP 删除回调会自动运行，从而自动清理 TLSP。
- **IDF 附加 API**：专用于 ESP-IDF 的附加函数，用于增强 FreeRTOS 的功能。
- **组件专用功能**：目前只添加了一个专用于组件的功能，即 `ORIG_INCLUDE_PATH`。

环形 buffer

FreeRTOS 提供了流 buffer 和消息 buffer，作为在任务和 ISR 之间发送任意大小数据的主要机制。然而，FreeRTOS 流 buffer 和消息 buffer 具有以下限制：

- 仅支持单一的发送者和单一的接收者
- 数据通过复制的方式进行传递
- 无法为延迟发送（即发送获取）预留 buffer 空间

为此，ESP-IDF 提供了一个单独的环形 buffer 来解决上述问题。

ESP-IDF 环形 buffer 是一个典型的 FIFO buffer，支持任意大小的数据项。在数据项大小可变的情况下，环形 buffer 比 FreeRTOS 队列更节约内存，可以替代 FreeRTOS 队列使用。环形 buffer 的容量不是由可以存储的数据项数量衡量的，而是由用于存储数据项的内存量来衡量的。

环形 buffer 提供了 API 来发送数据项，或为环形 buffer 中的数据项分配空间，以便进行手动填充。为提高效率，**数据项都通过引用的方式从环形 buffer 中检索出来**。因此，所有检索出的数据项也 **必须**通过 *`vRingbufferReturnItem()`* 或 *`vRingbufferReturnItemFromISR()`* 返回到环形 buffer，以便将其从环形 buffer 中完全移除。

环形 buffer 分为以下三种类型：

不可分割 buffer：确保将一个数据项存储在连续的内存中，并且在任何情况下都不会尝试分割数据项。当数据项必须占用连续的内存时，请使用不可分割 buffer。**仅不可分割 buffer 允许为延迟发送保留缓冲空间**。更多信息请参考函数 *`xRingbufferSendAcquire()`* 和 *`xRingbufferSendComplete()`* 的文档。

可分割 buffer：当数据项在 buffer 末尾绕回时，如果 buffer 头部和尾部的总空间足够，则支持将一个数据项分成两部分进行存储。可分割 buffer 比不可分割 buffer 更节省内存，但在检索时可能会返回数据项的两个部分。

字节 buffer：不将数据存储为单独的数据项。所有数据都存储为字节序列，每次可以发送或检索任意大小的字节。当不需要单独维护数据项时，推荐使用字节 buffer，例如字节流。

备注： 不可分割 buffer 和可分割 buffer 在 32 位对齐地址上存储数据项。因此，在检索一个数据项时，数据项指针一定也是 32 位对齐的。这在向 DMA 发送数据时非常有用。

备注： 存储在不可分割或可分割 buffer 中的每个数据项 **需要额外的 8 字节用于标头**。数据项大小会向上取整为 32 位对齐大小，即 4 字节的倍数，实际的数据项大小则记录在标头中。不可分割和可分割 buffer 的大小在创建时也会向上取整。

使用方法 以下示例演示了如何使用 *xRingbufferCreate()* 和 *xRingbufferSend()* 来创建环形 buffer，并向其发送数据项：

```
#include "freertos/ringbuf.h"
static char tx_item[] = "test_item";

...

    //创建环形 buffer
    RingbufHandle_t buf_handle;
    buf_handle = xRingbufferCreate(1028, RINGBUF_TYPE_NOSPLIT);
    if (buf_handle == NULL) {
        printf("Failed to create ring buffer\n");
    }

    //发送一个数据项
    UBaseType_t res =  xRingbufferSend(buf_handle, tx_item, sizeof(tx_item), pdMS_
↪TO_TICKS(1000));
    if (res != pdTRUE) {
        printf("Failed to send item\n");
    }
```

以下示例演示了如何使用 *xRingbufferSendAcquire()* 和 *xRingbufferSendComplete()* 代替 *xRingbufferSend()* 来获取环形 buffer（*RINGBUF_TYPE_NOSPLIT* 类型）上的内存，然后向其发送一个数据项。虽然增加了一个步骤，但可以实现获取要写入内存的地址，并自行写入内存。

```
#include "freertos/ringbuf.h"
#include "soc/lldesc.h"

typedef struct {
    lldesc_t dma_desc;
    uint8_t buf[1];
} dma_item_t;

#define DMA_ITEM_SIZE(N) (sizeof(lldesc_t)+(((N)+3)&(~3)))

...

    //为 DMA 描述符和相应的数据 buffer 检索空间
    //此步骤必须通过 SendAcquire 完成，否则，复制时地址可能会不同
    dma_item_t *item;
    UBaseType_t res =  xRingbufferSendAcquire(buf_handle,
                       (void**) &item, DMA_ITEM_SIZE(buffer_size), pdMS_TO_
↪TICKS(1000));
    if (res != pdTRUE) {
        printf("Failed to acquire memory for item\n");
    }
    item->dma_desc = (lldesc_t) {
        .size = buffer_size,
        .length = buffer_size,
        .eof = 0,
        .owner = 1,
        .buf = item->buf,
    };
    //实际发送到环形 buffer 以供使用
```

(下页继续)

(续上页)

```
    res = xRingbufferSendComplete(buf_handle, &item);
    if (res != pdTRUE) {
        printf("Failed to send item\n");
    }
```

以下示例演示了使用*xRingbufferReceive()* 和*vRingbufferReturnItem()* 从 **不可分割环形 buffer** 中检索和返回数据项：

```
...

    //从不可分割环形 buffer 中接收一个数据项
    size_t item_size;
    char *item = (char *)xRingbufferReceive(buf_handle, &item_size, pdMS_TO_
↪TICKS(1000));

    //Check received item
    if (item != NULL) {
        //Print item
        for (int i = 0; i < item_size; i++) {
            printf("%c", item[i]);
        }
        printf("\n");
        //返回数据项
        vRingbufferReturnItem(buf_handle, (void *)item);
    } else {
        //数据项检索失败
        printf("Failed to receive item\n");
    }
```

以下示例演示了使用*xRingbufferReceiveSplit()* 和*vRingbufferReturnItem()* 从 **可分割环形 buffer** 中检索和返回数据项：

```
...

    //从可分割环形 buffer 中接收一个数据项
    size_t item_size1, item_size2;
    char *item1, *item2;
    BaseType_t ret = xRingbufferReceiveSplit(buf_handle, (void **)&item1, (void␣
↪**)&item2, &item_size1, &item_size2, pdMS_TO_TICKS(1000));

    //检查收到的数据项
    if (ret == pdTRUE && item1 != NULL) {
        for (int i = 0; i < item_size1; i++) {
            printf("%c", item1[i]);
        }
        vRingbufferReturnItem(buf_handle, (void *)item1);
        //Check if item was split
        if (item2 != NULL) {
            for (int i = 0; i < item_size2; i++) {
                printf("%c", item2[i]);
            }
            vRingbufferReturnItem(buf_handle, (void *)item2);
        }
        printf("\n");
    } else {
        //接收数据项失败
        printf("Failed to receive item\n");
    }
```

以下示例演示了使用*xRingbufferReceiveUpTo()* 和*vRingbufferReturnItem()* 从 **字节 buffer** 中检索和返回数据项：

```
...

    //从字节 buffer 中接收数据
    size_t item_size;
    char *item = (char *)xRingbufferReceiveUpTo(buf_handle, &item_size, pdMS_TO_
↪TICKS(1000), sizeof(tx_item));

    //检查接收到的数据
    if (item != NULL) {
        //Print item
        for (int i = 0; i < item_size; i++) {
            printf("%c", item[i]);
        }
        printf("\n");
        //返回数据项
        vRingbufferReturnItem(buf_handle, (void *)item);
    } else {
        //接收数据项失败
        printf("Failed to receive item\n");
    }
```

对于以上函数的 ISR 安全版本，请调用*xRingbufferSendFromISR()*、*xRingbufferReceiveFromISR()*、*xRingbufferReceiveSplitFromISR()*、*xRingbufferReceiveUpToFromISR()* 和*vRingbufferReturnItemFromISR()*。

备注： 当字节在环形 buffer 的末端绕回时，需调用 `RingbufferReceive[UpTo][FromISR]()` 两次。

发送到环形 buffer 以下图表将不可分割和可分割 buffer 与字节 buffer 进行对比，说明了三者在发送数据或数据项方面的差异。图表中，假设分别向 **128 字节的 buffer** 发送大小为 **18、3 和 27 字节**的三个数据项：

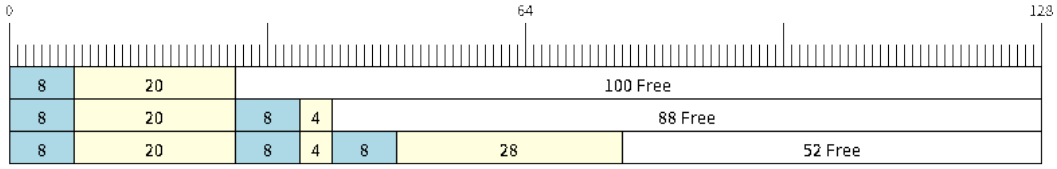


图 48: 向不可分割或可分割的环形 buffer 发送数据项

对于不可分割和可分割 buffer，每个数据项前都有 8 字节标头信息。此外，为了保持整体的 32 位对齐，每个数据项占用的空间都会 **向上取整到最接近的 32 位对齐大小**。数据项的实际大小会记录在标头中，并在检索数据项时返回。

参考上图，18、3 和 27 字节的数据项分别 **向上取整为 20、4 和 28 字节**，然后在每个数据项前面添加一个 8 字节的标头。

字节 buffer 将数据视为一个字节序列，不会产引入任何额外开销，不添加标头信息。因此，发送到字节 buffer 的所有数据都会合并成一个数据项。

参考上图，18、3 和 27 字节的数据项被顺序写入字节 buffer，并 **合并成一个 48 字节的数据项**。

使用 SendAcquire 和 SendComplete 不可分割 buffer 中的数据项严格按照 FIFO 顺序通过 `SendAcquire` 获取，并且必须通过 `SendComplete` 发送到 buffer 以便访问。也可以发送或获取多个数据项，且无需严格遵照获取顺序，但接收数据项却必须遵循 FIFO。所以，如果不为最早获取的数据项调用 `SendComplete`，就无法接收后续数据项。

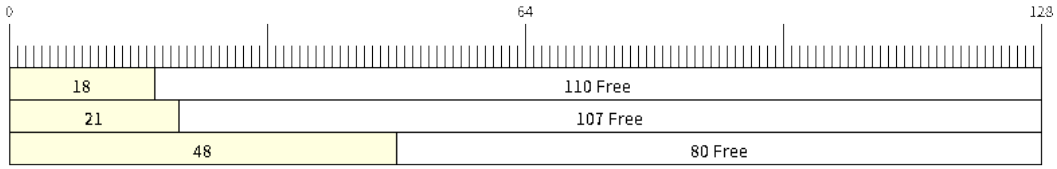


图 49: 向字节 buffer 发送数据项

以下图表说明了当 `SendAcquire` 和 `SendComplete` 顺序不同时的情形。一开始，已经有一个 16 字节的数据项发送到环形 buffer。然后调用 `SendAcquire` 在环形 buffer 上获取 20、8、24 字节的空间。

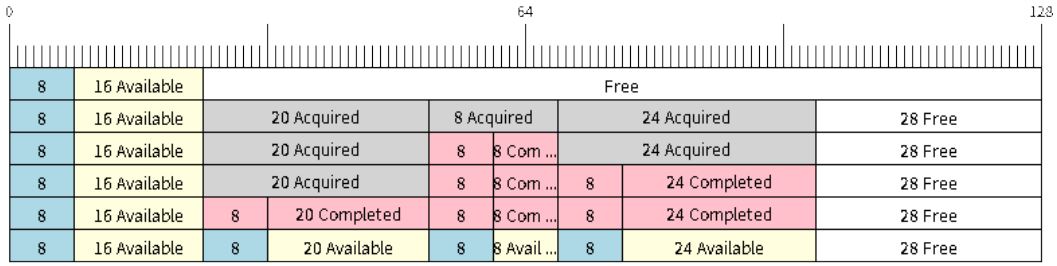


图 50: 在不可分割环形 buffer 中 SendAcquire/SendComplete 数据项

然后填充 buffer，按照 8、24、20 字节的顺序通过 `SendComplete` 将数据项发送到环形 buffer。当 8 字节和 24 字节的数据发送后，仍只能获取 16 字节的数据项。因此，如果不为 20 字节数据项调用 `SendComplete`，就无法获取该数据项，也无法获取 20 字节后的数据项。

当 20 字节数据项最终发送完成后，就可以在 buffer 中最初的 16 字节数据项之后，按照 20、8、24 字节的顺序接收所有的三个数据项。

由于 `SendAcquire` 及 `SendComplete` 要求所获取的 buffer 必须是完整的（未包装的），故可分割 buffer 和字节 buffer 不支持上述调用操作。

绕回 以下图表说明了发送数据项需要绕回时，不可分割、可分割和字节 buffer 之间的差异。图表假设有一个 **128 字节的 buffer**，其中有 **56 字节的空闲空间可以绕回使用**，并发送了一个 **28 字节**的数据项。

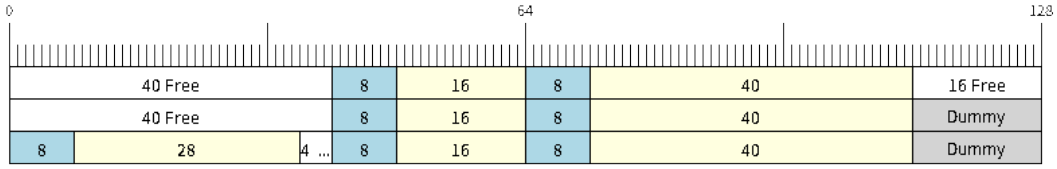


图 51: 在不可分割 buffer 中绕回

不可分割 buffer **只在连续的空闲空间中存储数据项，在任何情况下都不分割数据项**。当 buffer 尾部的空闲空间不足以完全存储数据项及其标头时，尾部的空闲空间将被 **标记为虚拟数据**。然后，数据项将绕回并存储在 buffer 头部的空闲空间中。

参考上图，buffer 尾部的 16 字节空闲空间不足以存储 28 字节的数据项，因此，这 16 字节被标记为虚拟数据，然后将数据项写入了 buffer 头部的空闲空间中。

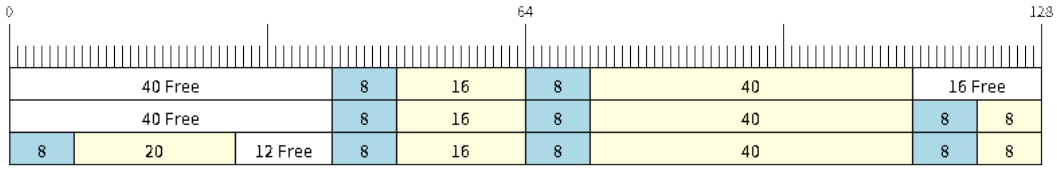


图 52: 在可分割 buffer 中绕回

当 buffer 尾部的空闲空间不足以存储数据项及其标头时，可分割 buffer 会尝试 **将数据项分割成两部分**。分割的的两部分数据项都将有自己的标头，因此会产生额外的 8 字节开销。

参考上图，buffer 尾部的 16 字节空闲空间不足以存储 28 字节的数据项。因此将数据项分割成两部分（8 字节和 20 字节），并将两部分写入 buffer。

备 注: 可分割 buffer 将其分割好的两部分数据视为两个独立的数据项，因此不应调用 *xRingbufferReceive()*。需调用 *xRingbufferReceiveSplit()* 以线程安全的方式接收分割的两部分数据项。

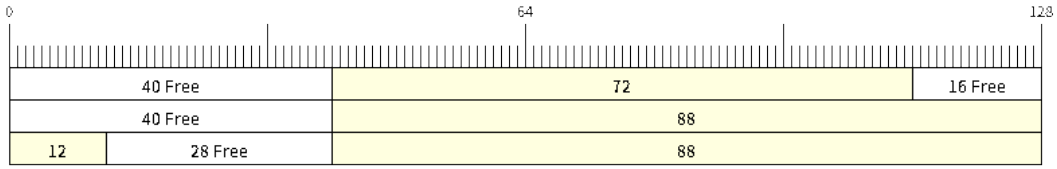


图 53: 在字节 buffer 中绕回

字节 buffer **将尽可能多的数据存储到 buffer 尾部的空闲空间**中。剩余的数据会存储在 buffer 头部的空闲空间。在字节 buffer 中绕回不会产生任何额外开销。

参考上图，buffer 尾部的 16 字节空闲空间不足以完全存储 28 字节的数据，因此，将数据填入这 16 字节空闲空间后，剩余的 12 字节会被写入 buffer 头部的空闲空间。此时，buffer 包含两个独立的连续数据，并且每个连续数据都被字节 buffer 视为一个独立数据项。

检索/返回 以下图表说明了在检索和返回数据时，不可分割、可分割 buffer 和字节 buffer 之间的差异：

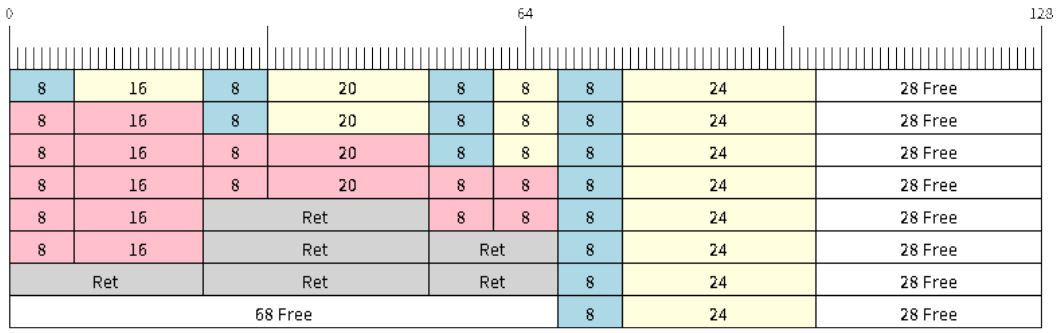


图 54: 在不可分割和可分割环形 buffer 中检索/返回数据项

不可分割 buffer 和可分割 buffer 中的数据项 **按严格的 FIFO 顺序检索**并 **必须返回**，以释放占用的空间。在返回之前可以检索多个数据项，且不必按照检索的顺序返回数据项。但是，释放空间必须按 FIFO 顺序进行，因此如果不返回最早检索的数据项，就无法释放后续数据项占用的空间。

参考上图，**16、20 和 8 字节的数据项按 FIFO 顺序被检索出来**。但是，这些数据项并不是按照被检索的顺序返回的。最先返回的是 20 字节的数据项，然后分别返回 8 字节和 16 字节的数据项。直到第一个数据项（即 16 字节的数据项）返回后，空间才会被释放。

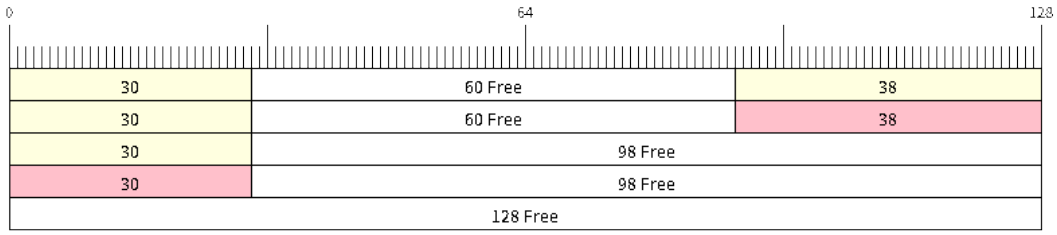


图 55: 在字节 buffer 中检索/返回数据

字节 buffer **不允许在返回之前进行多次检索**（每次检索必须在下一次检索之前返回结果）。使用*xRingbufferReceive()* 或*xRingbufferReceiveFromISR()* 时，会检索所有连续存储的数据。使用*xRingbufferReceiveUpTo()* 或*xRingbufferReceiveUpToFromISR()* 可限制检索的最大字节数。由于每次检索后都必须返回，因此数据一返回就会释放空间。

参考上图，buffer 尾部 38 字节连续存储的数据被检索、返回和释放。然后，下一次调用*xRingbufferReceive()* 或*xRingbufferReceiveFromISR()* 时，buffer 将绕回并对头部的 30 字节连续存储数据进行同样的处理。

使用队列集的环形 buffer 使用*xRingbufferAddToQueueSetRead()* 可以将环形 buffer 添加到 FreeRTOS 队列集中，这样每次环形 buffer 接收一个数据项或数据时，队列集都会收到通知。添加到队列集后，每次从环形 buffer 检索数据项时都应该先调用*xQueueSelectFromSet()*。要检查选定的队列集成员是否为环形 buffer，调用*xRingbufferCanRead()*。

以下示例演示了如何使用包含环形 buffer 的队列集：

```
#include "freertos/queue.h"
#include "freertos/ringbuf.h"

...

    //创建环形 buffer 和队列集
    RingbufHandle_t buf_handle = xRingbufferCreate(1028, RINGBUF_TYPE_NOSPLIT);
    QueueSetHandle_t queue_set = xQueueCreateSet(3);

    //向队列集中添加环形 buffer
    if (xRingbufferAddToQueueSetRead(buf_handle, queue_set) != pdTRUE) {
        printf("Failed to add to queue set\n");
    }

...

    //阻塞队列集
    QueueSetMemberHandle_t member = xQueueSelectFromSet(queue_set, pdMS_TO_
↪TICKS(1000));

    //检查成员是否为环形 buffer
    if (member != NULL && xRingbufferCanRead(buf_handle, member) == pdTRUE) {
        //Member is ring buffer, receive item from ring buffer
```

(下页继续)

(续上页)

```
        size_t item_size;
        char *item = (char *)xRingbufferReceive(buf_handle, &item_size, 0);

        //处理数据项
        ...

    } else {
        ...
    }
```

使用静态分配的环形 buffer *xRingbufferCreateStatic()* 可用于创建具有特定内存需求的环形 buffer（如在外部 RAM 中分配的环形 buffer）。环形 buffer 使用的所有内存块都必须在创建之前手动分配，然后传递给*xRingbufferCreateStatic()* 以初始化为环形 buffer。这些内存块中包括：

- 环形 buffer 的数据结构类型*StaticRingbuffer_t*。
- 环形 buffer 的存储区域，大小为 `xBufferSize`。注意，对于不可分割和可分割 buffer，`xBufferSize` 必须为 32 位对齐大小。

这些块的分配方式取决于具体的需求。例如，静态声明所有块，或动态分配为具有特定功能的块，如外部 RAM。

备注： 当删除通过*xRingbufferCreateStatic()* 创建的环形 buffer 时，*vRingbufferDelete()* 函数不会释放任何内存块。释放内存必须在调用*vRingbufferDelete()* 后手动完成。

下面的代码片段演示了一个完全在外部 RAM 中分配的环形 buffer：

```
#include "freertos/ringbuf.h"
#include "freertos/semphr.h"
#include "esp_heap_caps.h"

#define BUFFER_SIZE     400     //32 位对齐大小
#define BUFFER_TYPE     RINGBUF_TYPE_NOSPLIT
...

//将 环形 buffer 数据结构体和存储区分配到外部 RAM 中
StaticRingbuffer_t *buffer_struct = (StaticRingbuffer_t *)heap_caps_
↪malloc(sizeof(StaticRingbuffer_t), MALLOC_CAP_SPIRAM);
uint8_t *buffer_storage = (uint8_t *)heap_caps_malloc(sizeof(uint8_t)*BUFFER_SIZE,␣
↪MALLOC_CAP_SPIRAM);

//使用手动分配的内存创建环形 buffer
RingbufHandle_t handle = xRingbufferCreateStatic(BUFFER_SIZE, BUFFER_TYPE, buffer_
↪storage, buffer_struct);

...

//使用后删除环形 buffer
vRingbufferDelete(handle);

//手动释放所有内存块
free(buffer_struct);
free(buffer_storage);
```

ESP-IDF tick 钩子和 idle 钩子

FreeRTOS 允许应用程序在编译时提供一个 tick 钩子和一个 idle 钩子：

- FreeRTOS tick 钩子可以通过*CONFIG_FREERTOS_USE_TICK_HOOK* 选项启用。应用程序必须提供 `void vApplicationTickHook( void )` 回调。
- FreeRTOS idle 钩子可以通过*CONFIG_FREERTOS_USE_IDLE_HOOK* 选项启用。应用程序必须提供 `void vApplicationIdleHook( void )` 回调。

然而，FreeRTOS tick 钩子和 idle 钩子有以下不足：

- FreeRTOS 钩子是在编译时注册的
- 每种钩子只能注册一个
- 在多核目标芯片上，FreeRTOS 钩子是对称的，即每个内核的 tick 中断和 idle 任务最终都会调用同一个钩子

因此，ESP-IDF 提供了 tick 钩子和 idle 钩子来补充 FreeRTOS tick 和 idle 钩子的功能。ESP-IDF 钩子具有以下功能：

- 钩子可以在运行时注册和注销
- 可以注册多个钩子。每个内核中，同一类型的钩子最多可以注册 8 个
- 在多核目标芯片上，钩子可以是不对称的，即可以为每个内核注册不同的钩子

使用以下 API 注册和注销 ESP-IDF 钩子：

- 对于 tick 钩子：
  - 用*esp_register_freertos_tick_hook()* 或*esp_register_freertos_tick_hook_for_cpu()* 注册
  - 用*esp_deregister_freertos_tick_hook()* 或*esp_deregister_freertos_tick_hook_for_cpu* 注销
- 对于 idle 钩子：
  - 使用*esp_register_freertos_idle_hook()* 或*esp_register_freertos_idle_hook_for_cpu()* 注册
  - 使用*esp_deregister_freertos_idle_hook()* 或*esp_deregister_freertos_idle_hook_for_cp* 注销

备注： 在 cache 被禁用时，tick 中断仍保持活动，因此任何 tick 钩子（FreeRTOS 或 ESP-IDF）函数都必须放在内部 RAM 中。请参考*SPI flash API documentation* 了解详情。

TLSP 删除回调

原生 FreeRTOS 提供了线程本地存储指针 (TLSP) 功能，这些指针直接存储在特定任务的任务控制块 (TCB) 中。TLSP 允许每个任务拥有自己的数据结构指针集合。在原生 FreeRTOS 中：

- 在任务创建后，需调用*vTaskSetThreadLocalStoragePointer()* 设置任务的 TLSP。
- 在任务的生命周期中，需调用*pvTaskGetThreadLocalStoragePointer()* 获取任务的 TLSP。
- 在删除任务前，需释放 TLSP 指向的内存。

然而，为了能够欧自动释放 TLSP 内存，ESP-IDF 额外提供了 TLSP 删除回调功能。当删除任务时，这些删除回调函数会被自动调用，从而清除 TLSP 内存，无需在每个任务的代码中显式添加内存清除逻辑。

设置 TLSP 删除回调的方式与设置 TLSP 类似。

- *vTaskSetThreadLocalStoragePointerAndDelCallback()* 设置了特定的 TLSP 及其关联的回调。
- 调用原生 FreeRTOS 函数*vTaskSetThreadLocalStoragePointer()* 只会将 TLSP 的关联删除回调设置为 *NULL*，也就是说，在任务删除期间不会调用该 TLSP 的回调。

在实现 TLSP 回调时，应注意以下几点：

- 回调 **绝对不能尝试阻塞或让出**，并且应尽可能缩短临界区的时间。
- 回调是在删除任务的内存即将被释放前调用的。因此，回调可以通过*vTaskDelete()* 本身调用，也可以从空闲任务中调用。

IDF 附加 API

freertos/esp_additions/include/freertos/idf_additions.h 头文件包含了 ESP-IDF 添加的与 FreeRTOS 相关的辅助函数。通过 `#include "freertos/idf_additions.h"` 可添加此头文件。

组件专用功能

除了基本 CMake 构建属性中提供的标准组件变量外，FreeRTOS 组件还提供了参数（目前只有一个参数）以简化与其他模块的集成：

- *ORIG_INCLUDE_PATH* - 包含指向 freeRTOS 根包含文件夹的绝对路径。因此可以直接用 *#include "FreeRTOS.h"* 引用头文件，而无需使用 *#include "freertos/FreeRTOS.h"*。

API 参考

环形 buffer API

Header File

- components/esp_ringbuf/include/freertos/ringbuf.h
- This header file can be included with:

```
#include "freertos/ringbuf.h"
```

- This header file is a part of the API provided by the `esp_ringbuf` component. To declare that your component depends on `esp_ringbuf`, add the following to your CMakeLists.txt:

```
REQUIRES esp_ringbuf
```

or

```
PRIV_REQUIRES esp_ringbuf
```

Functions

RingbufHandle_t **xRingbufferCreate**(size_t xBufferSize, *RingbufferType_t* xBufferType)

Create a ring buffer.

备注： xBufferSize of no-split/allow-split buffers will be rounded up to the nearest 32-bit aligned size.

参数

- **xBufferSize** -- **[in]** Size of the buffer in bytes. Note that items require space for a header in no-split/allow-split buffers
- **xBufferType** -- **[in]** Type of ring buffer, see documentation.

返回 A handle to the created ring buffer, or NULL in case of error.

RingbufHandle_t **xRingbufferCreateNoSplit**(size_t xItemSize, size_t xItemNum)

Create a ring buffer of type RINGBUF_TYPE_NOSPLIT for a fixed item_size.

This API is similar to xRingbufferCreate(), but it will internally allocate additional space for the headers.

参数

- **xItemSize** -- **[in]** Size of each item to be put into the ring buffer
- **xItemNum** -- **[in]** Maximum number of items the buffer needs to hold simultaneously

返回 A RingbufHandle_t handle to the created ring buffer, or NULL in case of error.

RingbufHandle_t **xRingbufferCreateStatic**(size_t xBufferSize, *RingbufferType_t* xBufferType, uint8_t *pucRingbufferStorage, *StaticRingbuffer_t* *pxStaticRingbuffer)

Create a ring buffer but manually provide the required memory.

备注: xBufferSize of no-split/allow-split buffers MUST be 32-bit aligned.

参数

- **xBufferSize** -- **[in]** Size of the buffer in bytes.
- **xBufferType** -- **[in]** Type of ring buffer, see documentation
- **pucRingbufferStorage** -- **[in]** Pointer to the ring buffer's storage area. Storage area must have the same size as specified by xBufferSize
- **pxStaticRingbuffer** -- **[in]** Pointed to a struct of type StaticRingbuffer_t which will be used to hold the ring buffer's data structure

返回 A handle to the created ring buffer

BaseType_t **xRingbufferSend**(*RingbufHandle_t* xRingbuffer, const void *pvItem, size_t xItemSize, TickType_t xTicksToWait)

Insert an item into the ring buffer.

Attempt to insert an item into the ring buffer. This function will block until enough free space is available or until it times out.

备注: For no-split/allow-split ring buffers, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.

备注: For no-split/allow-split buffers, an xItemSize of 0 will result in an item with no data being set (i.e., item only contains the header). For byte buffers, an xItemSize of 0 will simply return pdTRUE without copying any data.

参数

- **xRingbuffer** -- **[in]** Ring buffer to insert the item into
- **pvItem** -- **[in]** Pointer to data to insert. NULL is allowed if xItemSize is 0.
- **xItemSize** -- **[in]** Size of data to insert.
- **xTicksToWait** -- **[in]** Ticks to wait for room in the ring buffer.

返回

- pdTRUE if succeeded
- pdFALSE on time-out or when the data is larger than the maximum permissible size of the buffer

BaseType_t **xRingbufferSendFromISR**(*RingbufHandle_t* xRingbuffer, const void *pvItem, size_t xItemSize, BaseType_t *pxHigherPriorityTaskWoken)

Insert an item into the ring buffer in an ISR.

Attempt to insert an item into the ring buffer from an ISR. This function will return immediately if there is insufficient free space in the buffer.

备注: For no-split/allow-split ring buffers, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.

备注: For no-split/allow-split buffers, an xItemSize of 0 will result in an item with no data being set (i.e., item only contains the header). For byte buffers, an xItemSize of 0 will simply return pdTRUE without copying any

data.

参数

- **xRingbuffer** -- **[in]** Ring buffer to insert the item into
- **pvItem** -- **[in]** Pointer to data to insert. NULL is allowed if xItemSize is 0.
- **xItemSize** -- **[in]** Size of data to insert.
- **pxHigherPriorityTaskWoken** -- **[out]** Value pointed to will be set to pdTRUE if the function woke up a higher priority task.

返回

- pdTRUE if succeeded
- pdFALSE when the ring buffer does not have space.

BaseType_t **xRingbufferSendAcquire**(*RingbufHandle_t* xRingbuffer, void **ppvItem, size_t xItemSize, TickType_t xTicksToWait)

Acquire memory from the ring buffer to be written to by an external source and to be sent later.

Attempt to allocate buffer for an item to be sent into the ring buffer. This function will block until enough free space is available or until it times out.

The item, as well as the following items `SendAcquire` or `Send` after it, will not be able to be read from the ring buffer until this item is actually sent into the ring buffer.

备注: Only applicable for no-split ring buffers now, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.

备注: An xItemSize of 0 will result in a buffer being acquired, but the buffer will have a size of 0.

参数

- **xRingbuffer** -- **[in]** Ring buffer to allocate the memory
- **ppvItem** -- **[out]** Double pointer to memory acquired (set to NULL if no memory were retrieved)
- **xItemSize** -- **[in]** Size of item to acquire.
- **xTicksToWait** -- **[in]** Ticks to wait for room in the ring buffer.

返回

- pdTRUE if succeeded
- pdFALSE on time-out or when the data is larger than the maximum permissible size of the buffer

BaseType_t **xRingbufferSendComplete**(*RingbufHandle_t* xRingbuffer, void *pvItem)

Actually send an item into the ring buffer allocated before by `xRingbufferSendAcquire`.

备 注: Only applicable for no-split ring buffers. Only call for items allocated by `xRingbufferSendAcquire`.

参数

- **xRingbuffer** -- **[in]** Ring buffer to insert the item into
- **pvItem** -- **[in]** Pointer to item in allocated memory to insert.

返回

- pdTRUE if succeeded
- pdFALSE if fail for some reason.

void *`xRingbufferReceive`(*RingbufHandle_t* xRingbuffer, size_t *pxItemSize, TickType_t xTicksToWait)

Retrieve an item from the ring buffer.

Attempt to retrieve an item from the ring buffer. This function will block until an item is available or until it times out.

备注: A call to vRingbufferReturnItem() is required after this to free the item retrieved.

备注: It is possible to receive items with a pxItemSize of 0 on no-split/allow split buffers.

参数

- `xRingbuffer` -- **[in]** Ring buffer to retrieve the item from
- `pxItemSize` -- **[out]** Pointer to a variable to which the size of the retrieved item will be written.
- `xTicksToWait` -- **[in]** Ticks to wait for items in the ring buffer.

返回

- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL on timeout, *pxItemSize is untouched in that case.

void *`xRingbufferReceiveFromISR`(*RingbufHandle_t* xRingbuffer, size_t *pxItemSize)

Retrieve an item from the ring buffer in an ISR.

Attempt to retrieve an item from the ring buffer. This function returns immediately if there are no items available for retrieval

备注: A call to vRingbufferReturnItemFromISR() is required after this to free the item retrieved.

备注: Byte buffers do not allow multiple retrievals before returning an item

备注: Two calls to RingbufferReceiveFromISR() are required if the bytes wrap around the end of the ring buffer.

备注: It is possible to receive items with a pxItemSize of 0 on no-split/allow split buffers.

参数

- `xRingbuffer` -- **[in]** Ring buffer to retrieve the item from
- `pxItemSize` -- **[out]** Pointer to a variable to which the size of the retrieved item will be written.

返回

- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL when the ring buffer is empty, *pxItemSize is untouched in that case.

BaseType_t `xRingbufferReceiveSplit`(*RingbufHandle_t* xRingbuffer, void **ppvHeadItem, void **ppvTailItem, size_t *pxHeadItemSize, size_t *pxTailItemSize, TickType_t xTicksToWait)

Retrieve a split item from an allow-split ring buffer.

Attempt to retrieve a split item from an allow-split ring buffer. If the item is not split, only a single item is retried. If the item is split, both parts will be retrieved. This function will block until an item is available or until it times out.

备注： Call(s) to vRingbufferReturnItem() is required after this to free up the item(s) retrieved.

备注： This function should only be called on allow-split buffers

备注： It is possible to receive items with a pxItemSize of 0 on allow split buffers.

参数

- **xRingbuffer** -- **[in]** Ring buffer to retrieve the item from
- **ppvHeadItem** -- **[out]** Double pointer to first part (set to NULL if no items were retrieved)
- **ppvTailItem** -- **[out]** Double pointer to second part (set to NULL if item is not split)
- **pxHeadItemSize** -- **[out]** Pointer to size of first part (unmodified if no items were retrieved)
- **pxTailItemSize** -- **[out]** Pointer to size of second part (unmodified if item is not split)
- **xTicksToWait** -- **[in]** Ticks to wait for items in the ring buffer.

返回

- pdTRUE if an item (split or unsplit) was retrieved
- pdFALSE when no item was retrieved

BaseType_t **xRingbufferReceiveSplitFromISR**(*RingbufHandle_t* xRingbuffer, void **ppvHeadItem, void **ppvTailItem, size_t *pxHeadItemSize, size_t *pxTailItemSize)

Retrieve a split item from an allow-split ring buffer in an ISR.

Attempt to retrieve a split item from an allow-split ring buffer. If the item is not split, only a single item is retried. If the item is split, both parts will be retrieved. This function returns immediately if there are no items available for retrieval

备注： Calls to vRingbufferReturnItemFromISR() is required after this to free up the item(s) retrieved.

备注： This function should only be called on allow-split buffers

备注： It is possible to receive items with a pxItemSize of 0 on allow split buffers.

参数

- **xRingbuffer** -- **[in]** Ring buffer to retrieve the item from
- **ppvHeadItem** -- **[out]** Double pointer to first part (set to NULL if no items were retrieved)
- **ppvTailItem** -- **[out]** Double pointer to second part (set to NULL if item is not split)
- **pxHeadItemSize** -- **[out]** Pointer to size of first part (unmodified if no items were retrieved)
- **pxTailItemSize** -- **[out]** Pointer to size of second part (unmodified if item is not split)

返回

- pdTRUE if an item (split or unsplit) was retrieved
- pdFALSE when no item was retrieved

void *`xRingbufferReceiveUpTo`(*RingbufHandle_t* xRingbuffer, size_t *pxItemSize, TickType_t xTicksToWait, size_t xMaxSize)

Retrieve bytes from a byte buffer, specifying the maximum amount of bytes to retrieve.

Attempt to retrieve data from a byte buffer whilst specifying a maximum number of bytes to retrieve. This function will block until there is data available for retrieval or until it times out.

备注： A call to vRingbufferReturnItem() is required after this to free up the data retrieved.

备注： This function should only be called on byte buffers

备注： Byte buffers do not allow multiple retrievals before returning an item

备注： Two calls to RingbufferReceiveUpTo() are required if the bytes wrap around the end of the ring buffer.

参数

- **`xRingbuffer`** -- **[in]** Ring buffer to retrieve the item from
- **`pxItemSize`** -- **[out]** Pointer to a variable to which the size of the retrieved item will be written.
- **`xTicksToWait`** -- **[in]** Ticks to wait for items in the ring buffer.
- **`xMaxSize`** -- **[in]** Maximum number of bytes to return.

返回

- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL on timeout, *pxItemSize is untouched in that case.

void *`xRingbufferReceiveUpToFromISR`(*RingbufHandle_t* xRingbuffer, size_t *pxItemSize, size_t xMaxSize)

Retrieve bytes from a byte buffer, specifying the maximum amount of bytes to retrieve. Call this from an ISR.

Attempt to retrieve bytes from a byte buffer whilst specifying a maximum number of bytes to retrieve. This function will return immediately if there is no data available for retrieval.

备注： A call to vRingbufferReturnItemFromISR() is required after this to free up the data received.

备注： This function should only be called on byte buffers

备注： Byte buffers do not allow multiple retrievals before returning an item

参数

- **`xRingbuffer`** -- **[in]** Ring buffer to retrieve the item from
- **`pxItemSize`** -- **[out]** Pointer to a variable to which the size of the retrieved item will be written.
- **`xMaxSize`** -- **[in]** Maximum number of bytes to return. Size of 0 simply returns NULL.

返回

- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL when the ring buffer is empty, *pxItemSize is untouched in that case.

void **vRingbufferReturnItem**(*RingbufHandle_t* xRingbuffer, void *pvItem)

Return a previously-retrieved item to the ring buffer.

备注: If a split item is retrieved, both parts should be returned by calling this function twice

参数

- **xRingbuffer** -- **[in]** Ring buffer the item was retrieved from
- **pvItem** -- **[in]** Item that was received earlier

void **vRingbufferReturnItemFromISR**(*RingbufHandle_t* xRingbuffer, void *pvItem, BaseType_t *pxHigherPriorityTaskWoken)

Return a previously-retrieved item to the ring buffer from an ISR.

备注: If a split item is retrieved, both parts should be returned by calling this function twice

参数

- **xRingbuffer** -- **[in]** Ring buffer the item was retrieved from
- **pvItem** -- **[in]** Item that was received earlier
- **pxHigherPriorityTaskWoken** -- **[out]** Value pointed to will be set to pdTRUE if the function woke up a higher priority task.

void **vRingbufferDelete**(*RingbufHandle_t* xRingbuffer)

Delete a ring buffer.

备注: This function will not deallocate any memory if the ring buffer was created using xRingbufferCreateStatic(). Deallocation must be done manually be the user.

参数 **xRingbuffer** -- **[in]** Ring buffer to delete

size_t **xRingbufferGetMaxItemSize**(*RingbufHandle_t* xRingbuffer)

Get maximum size of an item that can be placed in the ring buffer.

This function returns the maximum size an item can have if it was placed in an empty ring buffer.

备注: The max item size for a no-split buffer is limited to ((buffer_size/2)-header_size). This limit is imposed so that an item of max item size can always be sent to an empty no-split buffer regardless of the internal positions of the buffer's read/write/free pointers.

参数 **xRingbuffer** -- **[in]** Ring buffer to query
返回 Maximum size, in bytes, of an item that can be placed in a ring buffer.

size_t **xRingbufferGetCurFreeSize**(*RingbufHandle_t* xRingbuffer)

Get current free size available for an item/data in the buffer.

This gives the real time free space available for an item/data in the ring buffer. This represents the maximum size an item/data can have if it was currently sent to the ring buffer.

备注: An empty no-split buffer has a max current free size for an item that is limited to ((buffer_size/2)-header_size). See API reference for xRingbufferGetMaxItemSize().

> **警告:** This API is not thread safe. So, if multiple threads are accessing the same ring buffer, it is the application's responsibility to ensure atomic access to this API and the subsequent Send

参数 **xRingbuffer** -- **[in]** Ring buffer to query
返回 Current free size, in bytes, available for an entry

BaseType_t **xRingbufferAddToQueueSetRead**(*RingbufHandle_t* xRingbuffer, *QueueSetHandle_t* xQueueSet)

Add the ring buffer to a queue set. Notified when data has been written to the ring buffer.

This function adds the ring buffer to a queue set, thus allowing a task to block on multiple queues/ring buffers. The queue set is notified when the new data becomes available to read on the ring buffer.

参数

- **xRingbuffer** -- **[in]** Ring buffer to add to the queue set
- **xQueueSet** -- **[in]** Queue set to add the ring buffer to

返回

- pdTRUE on success, pdFALSE otherwise

static inline BaseType_t **xRingbufferCanRead**(*RingbufHandle_t* xRingbuffer, *QueueSetMemberHandle_t* xMember)

Check if the selected queue set member is a particular ring buffer.

This API checks if queue set member returned from xQueueSelectFromSet() is a particular ring buffer. If so, this indicates the ring buffer has items waiting to be retrieved.

参数

- **xRingbuffer** -- **[in]** Ring buffer to check
- **xMember** -- **[in]** Member returned from xQueueSelectFromSet

返回

- pdTRUE when selected queue set member is the ring buffer
- pdFALSE otherwise.

BaseType_t **xRingbufferRemoveFromQueueSetRead**(*RingbufHandle_t* xRingbuffer, *QueueSetHandle_t* xQueueSet)

Remove the ring buffer from a queue set.

This function removes a ring buffer from a queue set. The ring buffer must have been previously added to the queue set using xRingbufferAddToQueueSetRead().

参数

- **xRingbuffer** -- **[in]** Ring buffer to remove from the queue set
- **xQueueSet** -- **[in]** Queue set to remove the ring buffer from

返回

- pdTRUE on success
- pdFALSE otherwise

void **vRingbufferGetInfo**(*RingbufHandle_t* xRingbuffer, UBaseType_t *uxFree, UBaseType_t *uxRead, UBaseType_t *uxWrite, UBaseType_t *uxAcquire, UBaseType_t *uxItemsWaiting)

Get information about ring buffer status.

Get information of a ring buffer's current status such as free/read/write/acquire pointer positions, and number of items waiting to be retrieved. Arguments can be set to NULL if they are not required.

参数

- **xRingbuffer** -- **[in]** Ring buffer handle
- **uxFree** -- **[out]** Pointer use to store free pointer position
- **uxRead** -- **[out]** Pointer use to store read pointer position
- **uxWrite** -- **[out]** Pointer use to store write pointer position
- **uxAcquire** -- **[out]** Pointer use to store acquire pointer position

- **uxItemsWaiting** -- **[out]** Pointer use to store number of items (bytes for byte buffer) waiting to be retrieved

void **xRingbufferPrintInfo**(*RingbufHandle_t* xRingbuffer)

Debugging function to print the internal pointers in the ring buffer.

参数 **xRingbuffer** -- Ring buffer to show

BaseType_t **xRingbufferGetStaticBuffer**(*RingbufHandle_t* xRingbuffer, uint8_t **ppucRingbufferStorage, *StaticRingbuffer_t* **ppxStaticRingbuffer)

Retrieve the pointers to a statically created ring buffer.

参数

- **xRingbuffer** -- **[in]** Ring buffer
- **ppucRingbufferStorage** -- **[out]** Used to return a pointer to the queue's storage area buffer
- **ppxStaticRingbuffer** -- **[out]** Used to return a pointer to the queue's data structure buffer

返回 pdTRUE if buffers were retrieved, pdFALSE otherwise.

RingbufHandle_t **xRingbufferCreateWithCaps**(size_t xBufferSize, *RingbufferType_t* xBufferType, UBaseType_t uxMemoryCaps)

Creates a ring buffer with specific memory capabilities.

This function is similar to xRingbufferCreate(), except that it allows the memory allocated for the ring buffer to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注: A queue created using this function must only be deleted using vRingbufferDeleteWithCaps()

参数

- **xBufferSize** -- **[in]** Size of the buffer in bytes
- **xBufferType** -- **[in]** Type of ring buffer, see documentation.
- **uxMemoryCaps** -- **[in]** Memory capabilities of the queue's memory (see esp_heap_caps.h)

返回 Handle to the created ring buffer or NULL on failure.

void **vRingbufferDeleteWithCaps**(*RingbufHandle_t* xRingbuffer)

Deletes a ring buffer previously created using xRingbufferCreateWithCaps()

参数 **xRingbuffer** -- Ring buffer

Structures

struct **xSTATIC_RINGBUFFER**

Struct that is equivalent in size to the ring buffer's data structure.

The contents of this struct are not meant to be used directly. This structure is meant to be used when creating a statically allocated ring buffer where this struct is of the exact size required to store a ring buffer's control data structure.

Type Definitions

typedef void ***RingbufHandle_t**

Type by which ring buffers are referenced. For example, a call to xRingbufferCreate() returns a RingbufHandle_t variable that can then be used as a parameter to xRingbufferSend(), xRingbufferReceive(), etc.

typedef struct *xSTATIC_RINGBUFFER* **StaticRingbuffer_t**

Struct that is equivalent in size to the ring buffer's data structure.

The contents of this struct are not meant to be used directly. This structure is meant to be used when creating a statically allocated ring buffer where this struct is of the exact size required to store a ring buffer's control data structure.

Enumerations

enum **RingbufferType_t**

Values:

enumerator **RINGBUF_TYPE_NOSPLIT**

No-split buffers will only store an item in contiguous memory and will never split an item. Each item requires an 8 byte overhead for a header and will always internally occupy a 32-bit aligned size of space.

enumerator **RINGBUF_TYPE_ALLOWSPLIT**

Allow-split buffers will split an item into two parts if necessary in order to store it. Each item requires an 8 byte overhead for a header, splitting incurs an extra header. Each item will always internally occupy a 32-bit aligned size of space.

enumerator **RINGBUF_TYPE_BYTEBUF**

Byte buffers store data as a sequence of bytes and do not maintain separate items, therefore byte buffers have no overhead. All data is stored as a sequence of byte and any number of bytes can be sent or retrieved each time.

enumerator **RINGBUF_TYPE_MAX**

钩子 API

Header File

- components/esp_system/include/esp_freertos_hooks.h
- This header file can be included with:

```
#include "esp_freertos_hooks.h"
```

Functions

esp_err_t **esp_register_freertos_idle_hook_for_cpu**(*esp_freertos_idle_cb_t* new_idle_cb, UBaseType_t cpuid)

Register a callback to be called from the specified core's idle hook. The callback should return true if it should be called by the idle hook once per interrupt (or FreeRTOS tick), and return false if it should be called repeatedly as fast as possible by the idle hook.

警告： Idle callbacks MUST NOT, UNDER ANY CIRCUMSTANCES, CALL A FUNCTION THAT MIGHT BLOCK.

参数

- **new_idle_cb** -- **[in]** Callback to be called
- **cpuid** -- **[in]** id of the core

返回

- ESP_OK: Callback registered to the specified core's idle hook
- ESP_ERR_NO_MEM: No more space on the specified core's idle hook to register callback
- ESP_ERR_INVALID_ARG: cpuid is invalid

esp_err_t **esp_register_freertos_idle_hook**(*esp_freertos_idle_cb_t* new_idle_cb)

Register a callback to the idle hook of the core that calls this function. The callback should return true if it should be called by the idle hook once per interrupt (or FreeRTOS tick), and return false if it should be called repeatedly as fast as possible by the idle hook.

> **警告:** Idle callbacks MUST NOT, UNDER ANY CIRCUMSTANCES, CALL A FUNCTION THAT MIGHT BLOCK.

参数 **new_idle_cb** -- **[in]** Callback to be called

返回

- ESP_OK: Callback registered to the calling core's idle hook
- ESP_ERR_NO_MEM: No more space on the calling core's idle hook to register callback

esp_err_t **esp_register_freertos_tick_hook_for_cpu**(*esp_freertos_tick_cb_t* new_tick_cb, UBaseType_t cpuid)

Register a callback to be called from the specified core's tick hook.

参数

- **new_tick_cb** -- **[in]** Callback to be called
- **cpuid** -- **[in]** id of the core

返回

- ESP_OK: Callback registered to specified core's tick hook
- ESP_ERR_NO_MEM: No more space on the specified core's tick hook to register the callback
- ESP_ERR_INVALID_ARG: cpuid is invalid

esp_err_t **esp_register_freertos_tick_hook**(*esp_freertos_tick_cb_t* new_tick_cb)

Register a callback to be called from the calling core's tick hook.

参数 **new_tick_cb** -- **[in]** Callback to be called

返回

- ESP_OK: Callback registered to the calling core's tick hook
- ESP_ERR_NO_MEM: No more space on the calling core's tick hook to register the callback

void **esp_deregister_freertos_idle_hook_for_cpu**(*esp_freertos_idle_cb_t* old_idle_cb, UBaseType_t cpuid)

Unregister an idle callback from the idle hook of the specified core.

参数

- **old_idle_cb** -- **[in]** Callback to be unregistered
- **cpuid** -- **[in]** id of the core

void **esp_deregister_freertos_idle_hook**(*esp_freertos_idle_cb_t* old_idle_cb)

Unregister an idle callback. If the idle callback is registered to the idle hooks of both cores, the idle hook will be unregistered from both cores.

参数 **old_idle_cb** -- **[in]** Callback to be unregistered

void **esp_deregister_freertos_tick_hook_for_cpu**(*esp_freertos_tick_cb_t* old_tick_cb, UBaseType_t cpuid)

Unregister a tick callback from the tick hook of the specified core.

参数

- **old_tick_cb** -- **[in]** Callback to be unregistered

- **cpuid** -- **[in]** id of the core

void **esp_deregister_freertos_tick_hook**(*esp_freertos_tick_cb_t* old_tick_cb)

Unregister a tick callback. If the tick callback is registered to the tick hooks of both cores, the tick hook will be unregistered from both cores.

参数 **old_tick_cb** -- **[in]** Callback to be unregistered

Type Definitions

typedef bool (***esp_freertos_idle_cb_t**)(void)

typedef void (***esp_freertos_tick_cb_t**)(void)

附加 API

Header File

- components/freertos/esp_additions/include/freertos/idf_additions.h
- This header file can be included with:

```
#include "freertos/idf_additions.h"
```

Functions

BaseType_t **xTaskCreatePinnedToCore**(TaskFunction_t pxTaskCode, const char *const pcName, const uint32_t ulStackDepth, void *const pvParameters, UBaseType_t uxPriority, *TaskHandle_t* *const pxCreatedTask, const BaseType_t xCoreID)

Create a new task that is pinned to a particular core.

This function is similar to xTaskCreate(), but allows the creation of a pinned task. The task's pinned core is specified by the xCoreID argument. If xCoreID is set to tskNO_AFFINITY, then the task is unpinned and can run on any core.

备注: If (configNUMBER_OF_CORES == 1), setting xCoreID to tskNO_AFFINITY will be be treated as 0.

参数

- **pxTaskCode** -- Pointer to the task entry function.
- **pcName** -- A descriptive name for the task.
- **ulStackDepth** -- The size of the task stack specified as the NUMBER OF BYTES. Note that this differs from vanilla FreeRTOS.
- **pvParameters** -- Pointer that will be used as the parameter for the task being created.
- **uxPriority** -- The priority at which the task should run.
- **pxCreatedTask** -- Used to pass back a handle by which the created task can be referenced.
- **xCoreID** -- The core to which the task is pinned to, or tskNO_AFFINITY if the task has no core affinity.

返回 pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h

TaskHandle_t **xTaskCreateStaticPinnedToCore**(TaskFunction_t pxTaskCode, const char *const pcName, const uint32_t ulStackDepth, void *const pvParameters, UBaseType_t uxPriority, StackType_t *const puxStackBuffer, StaticTask_t *const pxTaskBuffer, const BaseType_t xCoreID)

Create a new static task that is pinned to a particular core.

This function is similar to xTaskCreateStatic(), but allows the creation of a pinned task. The task's pinned core is specified by the xCoreID argument. If xCoreID is set to tskNO_AFFINITY, then the task is unpinned and can run on any core.

备注： If (configNUMBER_OF_CORES == 1), setting xCoreID to tskNO_AFFINITY will be be treated as 0.

参数

- **pxTaskCode** -- Pointer to the task entry function.
- **pcName** -- A descriptive name for the task.
- **ulStackDepth** -- The size of the task stack specified as the NUMBER OF BYTES. Note that this differs from vanilla FreeRTOS.
- **pvParameters** -- Pointer that will be used as the parameter for the task being created.
- **uxPriority** -- The priority at which the task should run.
- **puxStackBuffer** -- Must point to a StackType_t array that has at least ulStackDepth indexes
- **pxTaskBuffer** -- Must point to a variable of type StaticTask_t, which will then be used to hold the task's data structures,
- **xCoreID** -- The core to which the task is pinned to, or tskNO_AFFINITY if the task has no core affinity.

返回 The task handle if the task was created, NULL otherwise.

BaseType_t **xTaskGetCoreID**(*TaskHandle_t* xTask)

Get the current core ID of a particular task.

Helper function to get the core ID of a particular task. If the task is pinned to a particular core, the core ID is returned. If the task is not pinned to a particular core, tskNO_AFFINITY is returned.

If CONFIG_FREERTOS_UNICORE is enabled, this function simply returns 0.

[refactor-todo] See if this needs to be deprecated (IDF-8145)(IDF-8164)

备注： If CONFIG_FREERTOS_SMP is enabled, please call vTaskCoreAffinityGet() instead.

备注： In IDF FreerTOS when configNUMBER_OF_CORES == 1, this function will always return 0,

---

参数 **xTask** -- The task to query
返回 The task's core ID or tskNO_AFFINITY

TaskHandle_t **xTaskGetCurrentTaskHandleForCore**(BaseType_t xCoreID)

Get the handle of the task currently running on a certain core.

Because of the nature of SMP processing, there is no guarantee that this value will still be valid on return and should only be used for debugging purposes.

[refactor-todo] See if this needs to be deprecated (IDF-8145)

参数 **xCoreID** -- The core to query
返回 Handle of the current task running on the queried core

uint8_t ***pxTaskGetStackStart**(*TaskHandle_t* xTask)

Returns the start of the stack associated with xTask.

Returns the lowest stack memory address, regardless of whether the stack grows up or down.

[refactor-todo] Change return type to StackType_t (IDF-8158)

参数 **xTask** -- Handle of the task associated with the stack returned. Set xTask to NULL to return the stack of the calling task.

返回 A pointer to the start of the stack.

void **vTaskSetThreadLocalStoragePointerAndDelCallback**(*TaskHandle_t* xTaskToSet, BaseType_t xIndex, void *pvValue, *TlsDeleteCallbackFunction_t* pvDelCallback)

Set local storage pointer and deletion callback.

Each task contains an array of pointers that is dimensioned by the configNUM_THREAD_LOCAL_STORAGE_POINTERS setting in FreeRTOSConfig.h. The kernel does not use the pointers itself, so the application writer can use the pointers for any purpose they wish.

Local storage pointers set for a task can reference dynamically allocated resources. This function is similar to vTaskSetThreadLocalStoragePointer, but provides a way to release these resources when the task gets deleted. For each pointer, a callback function can be set. This function will be called when task is deleted, with the local storage pointer index and value as arguments.

参数

- **xTaskToSet** -- Task to set thread local storage pointer for
- **xIndex** -- The index of the pointer to set, from 0 to configNUM_THREAD_LOCAL_STORAGE_POINTERS - 1.
- **pvValue** -- Pointer value to set.
- **pvDelCallback** -- Function to call to dispose of the local storage pointer when the task is deleted.

BaseType_t **xTaskCreatePinnedToCoreWithCaps**(TaskFunction_t pvTaskCode, const char *const pcName, const configSTACK_DEPTH_TYPE usStackDepth, void *const pvParameters, UBaseType_t uxPriority, *TaskHandle_t* *const pvCreatedTask, const BaseType_t xCoreID, UBaseType_t uxMemoryCaps)

Creates a pinned task where its stack has specific memory capabilities.

This function is similar to xTaskCreatePinnedToCore(), except that it allows the memory allocated for the task's stack to have specific capabilities (e.g., MALLOC_CAP_SPIRAM).

However, the specified capabilities will NOT apply to the task's TCB as a TCB must always be in internal RAM.

参数

- **pvTaskCode** -- Pointer to the task entry function
- **pcName** -- A descriptive name for the task
- **usStackDepth** -- The size of the task stack specified as the number of bytes
- **pvParameters** -- Pointer that will be used as the parameter for the task being created.
- **uxPriority** -- The priority at which the task should run.
- **pvCreatedTask** -- Used to pass back a handle by which the created task can be referenced.
- **xCoreID** -- Core to which the task is pinned to, or tskNO_AFFINITY if unpinned.
- **uxMemoryCaps** -- Memory capabilities of the task stack's memory (see esp_heap_caps.h)

返回 pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h

static inline BaseType_t **xTaskCreateWithCaps**(TaskFunction_t pvTaskCode, const char *const pcName, configSTACK_DEPTH_TYPE usStackDepth, void *const pvParameters, UBaseType_t uxPriority, *TaskHandle_t* *pvCreatedTask, UBaseType_t uxMemoryCaps)

Creates a task where its stack has specific memory capabilities.

This function is similar to xTaskCreate(), except that it allows the memory allocated for the task's stack to have specific capabilities (e.g., MALLOC_CAP_SPIRAM).

However, the specified capabilities will NOT apply to the task's TCB as a TCB must always be in internal RAM.

备注： A task created using this function must only be deleted using vTaskDeleteWithCaps()

参数

- **pvTaskCode** -- Pointer to the task entry function
- **pcName** -- A descriptive name for the task
- **usStackDepth** -- The size of the task stack specified as the number of bytes
- **pvParameters** -- Pointer that will be used as the parameter for the task being created.
- **uxPriority** -- The priority at which the task should run.
- **pvCreatedTask** -- Used to pass back a handle by which the created task can be referenced.
- **uxMemoryCaps** -- Memory capabilities of the task stack's memory (see esp_heap_caps.h)

返回 pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h

void **vTaskDeleteWithCaps**(*TaskHandle_t* xTaskToDelete)

Deletes a task previously created using xTaskCreateWithCaps() or xTaskCreatePinnedToCoreWithCaps()

备注： It is recommended to use this API to delete tasks from another task's context, rather than self-deletion. When the task is being deleted, it is vital to ensure that it is not running on another core. This API must not be called from an interrupt context.

参数 **xTaskToDelete** -- A handle to the task to be deleted

QueueHandle_t **xQueueCreateWithCaps**(UBaseType_t uxQueueLength, UBaseType_t uxItemSize, UBaseType_t uxMemoryCaps)

Creates a queue with specific memory capabilities.

This function is similar to xQueueCreate(), except that it allows the memory allocated for the queue to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注： A queue created using this function must only be deleted using vQueueDeleteWithCaps()

参数

- **uxQueueLength** -- The maximum number of items that the queue can contain.
- **uxItemSize** -- The number of bytes each item in the queue will require.
- **uxMemoryCaps** -- Memory capabilities of the queue's memory (see esp_heap_caps.h)

返回 Handle to the created queue or NULL on failure.

void **vQueueDeleteWithCaps**(*QueueHandle_t* xQueue)

Deletes a queue previously created using xQueueCreateWithCaps()

参数 **xQueue** -- A handle to the queue to be deleted.

static inline *SemaphoreHandle_t* **xSemaphoreCreateBinaryWithCaps**(UBaseType_t uxMemoryCaps)

Creates a binary semaphore with specific memory capabilities.

This function is similar to vSemaphoreCreateBinary(), except that it allows the memory allocated for the binary semaphore to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注： A binary semaphore created using this function must only be deleted using vSemaphoreDeleteWithCaps()

参数 **uxMemoryCaps** -- Memory capabilities of the binary semaphore's memory (see esp_heap_caps.h)

返回 Handle to the created binary semaphore or NULL on failure.

static inline *SemaphoreHandle_t* **xSemaphoreCreateCountingWithCaps**(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount, UBaseType_t uxMemoryCaps)

Creates a counting semaphore with specific memory capabilities.

This function is similar to xSemaphoreCreateCounting(), except that it allows the memory allocated for the counting semaphore to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注： A counting semaphore created using this function must only be deleted using vSemaphoreDeleteWithCaps()

参数

- **uxMaxCount** -- The maximum count value that can be reached.
- **uxInitialCount** -- The count value assigned to the semaphore when it is created.
- **uxMemoryCaps** -- Memory capabilities of the counting semaphore's memory (see esp_heap_caps.h)

返回 Handle to the created counting semaphore or NULL on failure.

static inline *SemaphoreHandle_t* **xSemaphoreCreateMutexWithCaps**(UBaseType_t uxMemoryCaps)

Creates a mutex semaphore with specific memory capabilities.

This function is similar to xSemaphoreCreateMutex(), except that it allows the memory allocated for the mutex semaphore to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注： A mutex semaphore created using this function must only be deleted using vSemaphoreDeleteWithCaps()

参数 **uxMemoryCaps** -- Memory capabilities of the mutex semaphore's memory (see esp_heap_caps.h)

返回 Handle to the created mutex semaphore or NULL on failure.

static inline *SemaphoreHandle_t* **xSemaphoreCreateRecursiveMutexWithCaps**(UBaseType_t uxMemoryCaps)

Creates a recursive mutex with specific memory capabilities.

This function is similar to xSemaphoreCreateRecursiveMutex(), except that it allows the memory allocated for the recursive mutex to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注： A recursive mutex created using this function must only be deleted using vSemaphoreDeleteWithCaps()

参数 **uxMemoryCaps** -- Memory capabilities of the recursive mutex's memory (see esp_heap_caps.h)

返回 Handle to the created recursive mutex or NULL on failure.

void **vSemaphoreDeleteWithCaps**(*SemaphoreHandle_t* xSemaphore)

Deletes a semaphore previously created using one of the xSemaphoreCreate...WithCaps() functions.

参数 **xSemaphore** -- A handle to the semaphore to be deleted.

static inline *StreamBufferHandle_t* **xStreamBufferCreateWithCaps**(size_t xBufferSizeBytes, size_t xTriggerLevelBytes, UBaseType_t uxMemoryCaps)

Creates a stream buffer with specific memory capabilities.

This function is similar to xStreamBufferCreate(), except that it allows the memory allocated for the stream buffer to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注: A stream buffer created using this function must only be deleted using vStreamBufferDeleteWithCaps()

参数

- **xBufferSizeBytes** -- The total number of bytes the stream buffer will be able to hold at any one time.
- **xTriggerLevelBytes** -- The number of bytes that must be in the stream buffer before unblocking
- **uxMemoryCaps** -- Memory capabilities of the stream buffer's memory (see esp_heap_caps.h)

返回 Handle to the created stream buffer or NULL on failure.

static inline void **vStreamBufferDeleteWithCaps**(*StreamBufferHandle_t* xStreamBuffer)

Deletes a stream buffer previously created using xStreamBufferCreateWithCaps()

参数 **xStreamBuffer** -- A handle to the stream buffer to be deleted.

static inline *MessageBufferHandle_t* **xMessageBufferCreateWithCaps**(size_t xBufferSizeBytes, UBaseType_t uxMemoryCaps)

Creates a message buffer with specific memory capabilities.

This function is similar to xMessageBufferCreate(), except that it allows the memory allocated for the message buffer to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注: A message buffer created using this function must only be deleted using vMessageBufferDeleteWithCaps()

参数

- **xBufferSizeBytes** -- The total number of bytes (not messages) the message buffer will be able to hold at any one time.
- **uxMemoryCaps** -- Memory capabilities of the message buffer's memory (see esp_heap_caps.h)

返回 Handle to the created message buffer or NULL on failure.

static inline void **vMessageBufferDeleteWithCaps**(*MessageBufferHandle_t* xMessageBuffer)

Deletes a stream buffer previously created using xMessageBufferCreateWithCaps()

参数 **xMessageBuffer** -- A handle to the message buffer to be deleted.

EventGroupHandle_t **xEventGroupCreateWithCaps**(UBaseType_t uxMemoryCaps)

Creates an event group with specific memory capabilities.

This function is similar to xEventGroupCreate(), except that it allows the memory allocated for the event group to have specific capabilities (e.g., MALLOC_CAP_INTERNAL).

备注： An event group created using this function must only be deleted using vEventGroupDeleteWithCaps()

参数 **uxMemoryCaps** -- Memory capabilities of the event group's memory (see esp_heap_caps.h)

返回 Handle to the created event group or NULL on failure.

void **vEventGroupDeleteWithCaps**(*EventGroupHandle_t* xEventGroup)

Deletes an event group previously created using xEventGroupCreateWithCaps()

参数 **xEventGroup** -- A handle to the event group to be deleted.

Type Definitions

typedef void (***TlsDeleteCallbackFunction_t**)(int, void*)

Prototype of local storage pointer deletion callback.

2.9.14 堆内存分配

栈 (stack) 和堆 (heap) 的区别

ESP-IDF 应用程序使用常见的计算机架构模式：由程序控制流动态分配的内存（即 **栈**）、由函数调用动态分配的内存（即 **堆**）以及在编译时分配的 **静态内存**。

由于 ESP-IDF 是一个多线程 RTOS 环境，因此每个 RTOS 任务都有自己的栈，这些栈默认在创建任务时从堆中分配。有关栈静态分配的方法，请参阅*xTaskCreateStatic()*。

ESP32-P4 使用多种类型的 RAM，因此具备不同属性的堆，即基于属性的内存分配器允许应用程序按不同目的进行堆分配。

多数情况下，可直接使用 C 标准函数库中的 `malloc()` 和 `free()` 函数实现堆分配。为充分利用各种内存类型及其特性，ESP-IDF 还具有基于内存属性的堆内存分配器。要配备具有特定属性的内存，如*DMA 存储器* 或可执行内存，可以创建具备所需属性的 OR 掩码，将其传递给*heap_caps_malloc()*。

内存属性

ESP32-P4 包含多种类型的 RAM：

- DRAM（数据 RAM）是连接到 CPU 数据总线上的内存，用于存储数据。这是作为堆访问最常见的一种内存。
- IRAM（指令 RAM）是连接到 CPU 指令总线上的内存，通常仅用于存储可执行数据（即指令）。如果作为通用内存访问，则所有访问必须为*32 位可访问内存*。
- D/IRAM 是连接到 CPU 数据总线和指令总线的 RAM，因此可用作指令 RAM 或数据 RAM。

有关内存类型的详细信息，请参阅*存储器类型*。

也可将外部 SPI RAM 连接到 ESP32-P4。通过缓存将*片外 RAM* 集成到 ESP32-P4 的内存映射中，访问方式与 DRAM 类似。

所有的 DRAM 内存都可以单字节访问，因此所有的 DRAM 堆都具有 `MALLOC_CAP_8BIT` 属性。要获取所有 DRAM 堆的剩余空间大小，请调用 `heap_caps_get_free_size(MALLOC_CAP_8BIT)`。

调用 `malloc()` 时，ESP-IDF `malloc()` 内部调用 `heap_caps_malloc_default(size)`，使用属性 `MALLOC_CAP_DEFAULT` 分配内存。该属性可实现字节寻址功能，即存储空间的最小编址单位为字节。

`malloc()` 使用基于属性的分配系统，所以使用*heap_caps_malloc()* 分配的内存可以通过调用标准的 `free()` 函数释放。

可用堆空间

DRAM 启动时，DRAM 堆包含应用程序未静态分配的所有数据内存，减少静态分配的 buffer 将增加可用的空闲堆空间。

调用命令*idf.py size* 可查找静态分配内存大小。

备注： 运行时可用的 DRAM 堆空间可能少于编译时计算的大小，因为启动时会在运行 FreeRTOS 调度程序之前从堆中分配部分内存，包括初始 FreeRTOS 任务的栈内存。

IRAM 启动时，IRAM 堆包含所有应用程序可执行代码未使用的指令内存。

调用命令*idf.py size* 查找应用程序使用的 IRAM 量。

D/IRAM 一些内存在 ESP32-P4 中可用作 DRAM 或 IRAM。如果从 D/IRAM 区域分配内存，则两种类型的内存的可用堆空间都会减少。

堆空间大小 启动时，所有 ESP-IDF 应用程序都会记录全部堆地址（和空间大小）的摘要，级别为 Info：

```
I (252) heap_init: Initializing. RAM available for dynamic allocation:
I (259) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (265) heap_init: At 3FFB2EC8 len 0002D138 (180 KiB): DRAM
I (272) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (278) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (284) heap_init: At 4008944C len 00016BB4 (90 KiB): IRAM
```

查找可用堆 请参阅堆内存信息。

特殊用途

DMA 存储器 使用 `MALLOC_CAP_DMA` 标志分配适合与硬件 DMA 引擎（如 SPI 和 I2S）配合使用的内存，此属性标志不包括外部 PSRAM。

32 位可访问内存 如果某个内存结构体仅以 32 位为单位寻址，例如一个整数或指针数组，则可以使用 `MALLOC_CAP_32BIT` 标志分配。通过这一方式，分配器能够在无法调用 `malloc()` 的情况下提供 IRAM 内存，从而充分利用 ESP32-P4 中的所有可用内存。

请注意，使用 `MALLOC_CAP_32BIT` 分配的内存 只能通过 32 位读写访问，其他类型的访问将导致 LoadStoreError 异常。

外部 SPI 内存 当启用片外 *RAM* 时，可以根据配置调用标准 `malloc` 或通过 `heap_caps_malloc(MALLOC_CAP_SPIRAM)` 分配外部 SPI RAM，详情请参阅配置片外 *RAM*。

线程安全性

堆函数是线程安全的，因此可不受限制，在不同任务中同时调用多个堆函数。

从中断处理程序 (ISR) 上下文中调用 `malloc`、`free` 和相关函数虽然在技术层面可行（请参阅从 *ISR* 调用堆相关函数），但不建议使用此种方法，因为调用堆函数可能会延迟其他中断。建议重构应用程序，将 ISR 使用的任何 buffer 预先分配到 ISR 之外。之后可能会删除从 ISR 调用堆函数的功能。

从 ISR 调用堆相关函数

堆组件中的以下函数可以在中断处理程序 (ISR) 中调用：

- *heap_caps_malloc()*
- heap_caps_malloc_default()
- heap_caps_realloc_default()
- *heap_caps_malloc_prefer()*
- *heap_caps_realloc_prefer()*
- *heap_caps_calloc_prefer()*
- *heap_caps_free()*
- *heap_caps_realloc()*
- *heap_caps_calloc()*
- *heap_caps_aligned_alloc()*
- *heap_caps_aligned_free()*

备注： 不建议使用此种方法。

堆跟踪及调试

以下功能介绍详见堆内存调试：

- 堆信息（释放内存空间等）
- 堆分配与释放函数挂钩
- 堆损坏检测
- 堆跟踪（检测、监控内存泄漏等）

实现说明

堆属性分配器对芯片内存区域的了解源于 SoC 组件，该组件包含芯片的内存布局信息以及每个区域的不同属性。各区域的功能为首要考虑因素，如会优先使用 DRAM 和 IRAM 特定区域而非用途更广的 D/IRAM 区域来分配内存。

每个连续的内存区域都包含其自己的内存堆，由*multi_heap* 函数创建。multi_heap 允许将任何连续的内存区域作为堆使用。

堆属性分配器通过对内存区域的了解初始化每个单独的堆，堆属性 API 中的分配函数将基于所需的属性、可用空间和每个区域使用的首选项为分配函数找到最合适的堆，随后为位于特定内存区域的堆调用*multi_heap_malloc()*。

调用 free() 查找对应释放地址的特定堆，随后在特定的 multi_heap 实例上调用*multi_heap_free()*。

API 参考 - 堆分配

Header File

- components/heap/include/esp_heap_caps.h
- This header file can be included with:

```
#include "esp_heap_caps.h"
```

Functions

esp_err_t **heap_caps_register_failed_alloc_callback**(*esp_alloc_failed_hook_t* callback)

registers a callback function to be invoked if a memory allocation operation fails

参数 **callback** -- caller defined callback to be invoked

返回 ESP_OK if callback was registered.

void ***heap_caps_malloc**(size_t size, uint32_t caps)

Allocate a chunk of memory which has the given capabilities.

Equivalent semantics to libc malloc(), for capability-aware memory.

参数

- **size** -- Size, in bytes, of the amount of memory to allocate
- **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

返回 A pointer to the memory allocated on success, NULL on failure

void **heap_caps_free**(void *ptr)

Free memory previously allocated via heap_caps_malloc() or heap_caps_realloc().

Equivalent semantics to libc free(), for capability-aware memory.

In IDF, `free(p)` is equivalent to `heap_caps_free(p)`.

参数 **ptr** -- Pointer to memory previously returned from heap_caps_malloc() or heap_caps_realloc(). Can be NULL.

void ***heap_caps_realloc**(void *ptr, size_t size, uint32_t caps)

Reallocate memory previously allocated via heap_caps_malloc() or heap_caps_realloc().

Equivalent semantics to libc realloc(), for capability-aware memory.

In IDF, `realloc(p, s)` is equivalent to `heap_caps_realloc(p, s, MALLOC_CAP_8BIT)`.

'caps' parameter can be different to the capabilities that any original 'ptr' was allocated with. In this way, realloc can be used to "move" a buffer if necessary to ensure it meets a new set of capabilities.

参数

- **ptr** -- Pointer to previously allocated memory, or NULL for a new allocation.
- **size** -- Size of the new buffer requested, or 0 to free the buffer.
- **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory desired for the new allocation.

返回 Pointer to a new buffer of size 'size' with capabilities 'caps', or NULL if allocation failed.

void ***heap_caps_aligned_alloc**(size_t alignment, size_t size, uint32_t caps)

Allocate an aligned chunk of memory which has the given capabilities.

Equivalent semantics to libc aligned_alloc(), for capability-aware memory.

参数

- **alignment** -- How the pointer received needs to be aligned must be a power of two
- **size** -- Size, in bytes, of the amount of memory to allocate
- **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

返回 A pointer to the memory allocated on success, NULL on failure

void **heap_caps_aligned_free**(void *ptr)

Used to deallocate memory previously allocated with heap_caps_aligned_alloc.

备注: This function is deprecated, please consider using heap_caps_free() instead

---

参数 **ptr** -- Pointer to the memory allocated

void ***heap_caps_aligned_calloc**(size_t alignment, size_t n, size_t size, uint32_t caps)

Allocate an aligned chunk of memory which has the given capabilities. The initialized value in the memory is set to zero.

参数

- **alignment** -- How the pointer received needs to be aligned must be a power of two
- **n** -- Number of continuing chunks of memory to allocate
- **size** -- Size, in bytes, of a chunk of memory to allocate
- **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

返回 A pointer to the memory allocated on success, NULL on failure

void ***heap_caps_calloc**(size_t n, size_t size, uint32_t caps)

Allocate a chunk of memory which has the given capabilities. The initialized value in the memory is set to zero.

Equivalent semantics to libc calloc(), for capability-aware memory.

In IDF, `calloc(p)` is equivalent to `heap_caps_calloc(p, MALLOC_CAP_8BIT)`.

参数

- **n** -- Number of continuing chunks of memory to allocate
- **size** -- Size, in bytes, of a chunk of memory to allocate
- **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

返回 A pointer to the memory allocated on success, NULL on failure

size_t **heap_caps_get_total_size**(uint32_t caps)

Get the total size of all the regions that have the given capabilities.

This function takes all regions capable of having the given capabilities allocated in them and adds up the total space they have.

参数 **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
返回 total size in bytes

size_t **heap_caps_get_free_size**(uint32_t caps)

Get the total free size of all the regions that have the given capabilities.

This function takes all regions capable of having the given capabilities allocated in them and adds up the free space they have.

备注: Note that because of heap fragmentation it is probably not possible to allocate a single block of memory of this size. Use heap_caps_get_largest_free_block() for this purpose.

参数 **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
返回 Amount of free bytes in the regions

size_t **heap_caps_get_minimum_free_size**(uint32_t caps)

Get the total minimum free memory of all regions with the given capabilities.

This adds all the low watermarks of the regions capable of delivering the memory with the given capabilities.

备注: Note the result may be less than the global all-time minimum available heap of this kind, as "low watermarks" are tracked per-region. Individual regions' heaps may have reached their "low watermarks" at different points in time. However, this result still gives a "worst case" indication for all-time minimum free heap.

参数 **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
返回 Amount of free bytes in the regions

size_t **heap_caps_get_largest_free_block**(uint32_t caps)

Get the largest free block of memory able to be allocated with the given capabilities.

Returns the largest value of `s` for which `heap_caps_malloc(s, caps)` will succeed.

参数 **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
返回 Size of the largest free block in bytes.

esp_err_t **heap_caps_monitor_local_minimum_free_size_start**(void)

Start monitoring the value of minimum_free_bytes from the moment this function is called instead of from startup.

备注: This allows to detect local lows of the minimum_free_bytes value that wouldn't be detected otherwise.

返回 esp_err_t ESP_OK if the function executed properly ESP_FAIL if called when monitoring already active

esp_err_t **heap_caps_monitor_local_minimum_free_size_stop**(void)

Stop monitoring the value of minimum_free_bytes. After this call the minimum_free_bytes value calculated from startup will be returned in heap_caps_get_info and heap_caps_get_minimum_free_size.

返回 esp_err_t ESP_OK if the function executed properly ESP_FAIL if called when monitoring not active

void **heap_caps_get_info**(*multi_heap_info_t* *info, uint32_t caps)

Get heap info for all regions with the given capabilities.

Calls multi_heap_info() on all heaps which share the given capabilities. The information returned is an aggregate across all matching heaps. The meanings of fields are the same as defined for *multi_heap_info_t*, except that `minimum_free_bytes` has the same caveats described in heap_caps_get_minimum_free_size().

参数

- **info** -- Pointer to a structure which will be filled with relevant heap metadata.
- **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

void **heap_caps_print_heap_info**(uint32_t caps)

Print a summary of all memory with the given capabilities.

Calls multi_heap_info on all heaps which share the given capabilities, and prints a two-line summary for each, then a total summary.

参数 **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

bool **heap_caps_check_integrity_all**(bool print_errors)

Check integrity of all heap memory in the system.

Calls multi_heap_check on all heaps. Optionally print errors if heaps are corrupt.

Calling this function is equivalent to calling heap_caps_check_integrity with the caps argument set to MALLOC_CAP_INVALID.

备注: Please increase the value of `CONFIG_ESP_INT_WDT_TIMEOUT_MS` when using this API with PSRAM enabled.

参数 **print_errors** -- Print specific errors if heap corruption is found.
返回 True if all heaps are valid, False if at least one heap is corrupt.

bool **heap_caps_check_integrity**(uint32_t caps, bool print_errors)

Check integrity of all heaps with the given capabilities.

Calls multi_heap_check on all heaps which share the given capabilities. Optionally print errors if the heaps are corrupt.

See also heap_caps_check_integrity_all to check all heap memory in the system and heap_caps_check_integrity_addr to check memory around a single address.

备注： Please increase the value of `CONFIG_ESP_INT_WDT_TIMEOUT_MS` when using this API with PSRAM capability flag.

参数
- **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
- **print_errors** -- Print specific errors if heap corruption is found.

返回 True if all heaps are valid, False if at least one heap is corrupt.

bool **heap_caps_check_integrity_addr**(intptr_t addr, bool print_errors)

Check integrity of heap memory around a given address.

This function can be used to check the integrity of a single region of heap memory, which contains the given address.

This can be useful if debugging heap integrity for corruption at a known address, as it has a lower overhead than checking all heap regions. Note that if the corrupt address moves around between runs (due to timing or other factors) then this approach won't work, and you should call heap_caps_check_integrity or heap_caps_check_integrity_all instead.

备注： The entire heap region around the address is checked, not only the adjacent heap blocks.

参数
- **addr** -- Address in memory. Check for corruption in region containing this address.
- **print_errors** -- Print specific errors if heap corruption is found.

返回 True if the heap containing the specified address is valid, False if at least one heap is corrupt or the address doesn't belong to a heap region.

void **heap_caps_malloc_extmem_enable**(size_t limit)

Enable malloc() in external memory and set limit below which malloc() attempts are placed in internal memory.

When external memory is in use, the allocation strategy is to initially try to satisfy smaller allocation requests with internal memory and larger requests with external memory. This sets the limit between the two, as well as generally enabling allocation in external memory.

参数 **limit** -- Limit, in bytes.

void ***heap_caps_malloc_prefer**(size_t size, size_t num, ...)

Allocate a chunk of memory as preference in decreasing order.

Attention The variable parameters are bitwise OR of MALLOC_CAP_* flags indicating the type of memory. This API prefers to allocate memory with the first parameter. If failed, allocate memory with the next parameter. It will try in this order until allocating a chunk of memory successfully or fail to allocate memories with any of the parameters.

参数
- **size** -- Size, in bytes, of the amount of memory to allocate
- **num** -- Number of variable parameters

返回 A pointer to the memory allocated on success, NULL on failure

void ***heap_caps_realloc_prefer**(void *ptr, size_t size, size_t num, ...)

Reallocate a chunk of memory as preference in decreasing order.

参数
- **ptr** -- Pointer to previously allocated memory, or NULL for a new allocation.
- **size** -- Size of the new buffer requested, or 0 to free the buffer.
- **num** -- Number of variable parameters

返回 Pointer to a new buffer of size 'size', or NULL if allocation failed.

void ***heap_caps_calloc_prefer**(size_t n, size_t size, size_t num, ...)

Allocate a chunk of memory as preference in decreasing order.

参数

- **n** -- Number of continuing chunks of memory to allocate
- **size** -- Size, in bytes, of a chunk of memory to allocate
- **num** -- Number of variable parameters

返回 A pointer to the memory allocated on success, NULL on failure

void **heap_caps_dump**(uint32_t caps)

Dump the full structure of all heaps with matching capabilities.

Prints a large amount of output to serial (because of locking limitations, the output bypasses stdout/stderr). For each (variable sized) block in each matching heap, the following output is printed on a single line:

- Block address (the data buffer returned by malloc is 4 bytes after this if heap debugging is set to Basic, or 8 bytes otherwise).
- Data size (the data size may be larger than the size requested by malloc, either due to heap fragmentation or because of heap debugging level).
- Address of next block in the heap.
- If the block is free, the address of the next free block is also printed.

参数 **caps** -- Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

void **heap_caps_dump_all**(void)

Dump the full structure of all heaps.

Covers all registered heaps. Prints a large amount of output to serial.

Output is the same as for heap_caps_dump.

size_t **heap_caps_get_allocated_size**(void *ptr)

Return the size that a particular pointer was allocated with.

备注: The app will crash with an assertion failure if the pointer is not valid.

参数 **ptr** -- Pointer to currently allocated heap memory. Must be a pointer value previously returned by heap_caps_malloc, malloc, calloc, etc. and not yet freed.

返回 Size of the memory allocated at this block.

void **heap_caps_walk**(uint32_t caps, *heap_caps_walker_cb_t* walker_func, void *user_data)

Function called to walk through the heaps with the given set of capabilities.

参数

- **caps** -- The set of capabilities assigned to the heaps to walk through
- **walker_func** -- Callback called for each block of the heaps being traversed
- **user_data** -- Opaque pointer to user defined data

void **heap_caps_walk_all**(*heap_caps_walker_cb_t* walker_func, void *user_data)

Function called to walk through all heaps defined by the heap component.

参数

- **walker_func** -- Callback called for each block of the heaps being traversed
- **user_data** -- Opaque pointer to user defined data

Structures

struct **walker_heap_info**

Structure used to store heap related data passed to the walker callback function.

Public Members

intptr_t **start**

Start address of the heap in which the block is located.

intptr_t **end**

End address of the heap in which the block is located.

struct **walker_block_info**

Structure used to store block related data passed to the walker callback function.

Public Members

void ***ptr**

Pointer to the block data.

size_t **size**

The size of the block.

bool **used**

Block status. True: used, False: free.

Macros

HEAP_IRAM_ATTR

MALLOC_CAP_EXEC

Flags to indicate the capabilities of the various memory systems.

Memory must be able to run executable code

MALLOC_CAP_32BIT

Memory must allow for aligned 32-bit data accesses.

MALLOC_CAP_8BIT

Memory must allow for 8/16/...-bit data accesses.

MALLOC_CAP_DMA

Memory must be able to accessed by DMA.

MALLOC_CAP_PID2

Memory must be mapped to PID2 memory space (PIDs are not currently used)

MALLOC_CAP_PID3

Memory must be mapped to PID3 memory space (PIDs are not currently used)

MALLOC_CAP_PID4

Memory must be mapped to PID4 memory space (PIDs are not currently used)

MALLOC_CAP_PID5

Memory must be mapped to PID5 memory space (PIDs are not currently used)

MALLOC_CAP_PID6

Memory must be mapped to PID6 memory space (PIDs are not currently used)

MALLOC_CAP_PID7

Memory must be mapped to PID7 memory space (PIDs are not currently used)

MALLOC_CAP_SPIRAM

Memory must be in SPI RAM.

MALLOC_CAP_INTERNAL

Memory must be internal; specifically it should not disappear when flash/spiram cache is switched off.

MALLOC_CAP_DEFAULT

Memory can be returned in a non-capability-specific memory allocation (e.g. malloc(), calloc()) call.

MALLOC_CAP_IRAM_8BIT

Memory must be in IRAM and allow unaligned access.

MALLOC_CAP_RETENTION

Memory must be able to accessed by retention DMA.

MALLOC_CAP_RTCRAM

Memory must be in RTC fast memory.

MALLOC_CAP_TCM

Memory must be in TCM memory.

MALLOC_CAP_DMA_DESC_AHB

Memory must be capable of containing AHB DMA descriptors.

MALLOC_CAP_DMA_DESC_AXI

Memory must be capable of containing AXI DMA descriptors.

MALLOC_CAP_CACHE_ALIGNED

Memory must be aligned to the cache line size of any intermediate caches.

MALLOC_CAP_INVALID

Memory can't be used / list end marker.

Type Definitions

typedef void (*`esp_alloc_failed_hook_t`)(size_t size, uint32_t caps, const char *function_name)

callback called when an allocation operation fails, if registered

Param size in bytes of failed allocation
Param caps capabilities requested of failed allocation
Param function_name function which generated the failure

typedef struct *walker_heap_info* `walker_heap_into_t`

Structure used to store heap related data passed to the walker callback function.

typedef struct *walker_block_info* `walker_block_info_t`

Structure used to store block related data passed to the walker callback function.

typedef bool (*`heap_caps_walker_cb_t`)(*walker_heap_into_t* heap_info, *walker_block_info_t* block_info, void *user_data)

Function callback used to get information of memory block during calls to heap_caps_walk or heap_caps_walk_all.

Param heap_info See walker_heap_into_t
Param block_info See walker_block_info_t
Param user_data Opaque pointer to user defined data
Return True to proceed with the heap traversal False to stop the traversal of the current heap and continue with the traversal of the next heap (if any)

API 参考 - 初始化

Header File

- components/heap/include/esp_heap_caps_init.h
- This header file can be included with:

```
#include "esp_heap_caps_init.h"
```

Functions

void `heap_caps_init`(void)

Initialize the capability-aware heap allocator.

This is called once in the IDF startup code. Do not call it at other times.

void `heap_caps_enable_nonos_stack_heaps`(void)

Enable heap(s) in memory regions where the startup stacks are located.

On startup, the pro/app CPUs have a certain memory region they use as stack, so we cannot do allocations in the regions these stack frames are. When FreeRTOS is completely started, they do not use that memory anymore and heap(s) there can be enabled.

esp_err_t `heap_caps_add_region`(intptr_t start, intptr_t end)

Add a region of memory to the collection of heaps at runtime.

Most memory regions are defined in soc_memory_layout.c for the SoC, and are registered via heap_caps_init(). Some regions can't be used immediately and are later enabled via heap_caps_enable_nonos_stack_heaps().

Call this function to add a region of memory to the heap at some later time.

This function does not consider any of the "reserved" regions or other data in soc_memory_layout, caller needs to consider this themselves.

All memory within the region specified by start & end parameters must be otherwise unused.

The capabilities of the newly registered memory will be determined by the start address, as looked up in the regions specified in soc_memory_layout.c.

Use heap_caps_add_region_with_caps() to register a region with custom capabilities.

备注： Please refer to following example for memory regions allowed for addition to heap based on an existing region (address range for demonstration purpose only):

```
Existing region: 0x1000 <-> 0x3000
New region:      0x1000 <-> 0x3000 (Allowed)
New region:      0x1000 <-> 0x2000 (Allowed)
New region:      0x0000 <-> 0x1000 (Allowed)
New region:      0x3000 <-> 0x4000 (Allowed)
New region:      0x0000 <-> 0x2000 (NOT Allowed)
New region:      0x0000 <-> 0x4000 (NOT Allowed)
New region:      0x1000 <-> 0x4000 (NOT Allowed)
New region:      0x2000 <-> 0x4000 (NOT Allowed)
```

参数

- **start** -- Start address of new region.
- **end** -- End address of new region.

返回 ESP_OK on success, ESP_ERR_INVALID_ARG if a parameter is invalid, ESP_ERR_NOT_FOUND if the specified start address doesn't reside in a known region, or any error returned by heap_caps_add_region_with_caps().

esp_err_t **heap_caps_add_region_with_caps**(const uint32_t caps[], intptr_t start, intptr_t end)

Add a region of memory to the collection of heaps at runtime, with custom capabilities.

Similar to heap_caps_add_region(), only custom memory capabilities are specified by the caller.

备注： Please refer to following example for memory regions allowed for addition to heap based on an existing region (address range for demonstration purpose only):

```
Existing region: 0x1000 <-> 0x3000
New region:      0x1000 <-> 0x3000 (Allowed)
New region:      0x1000 <-> 0x2000 (Allowed)
New region:      0x0000 <-> 0x1000 (Allowed)
New region:      0x3000 <-> 0x4000 (Allowed)
New region:      0x0000 <-> 0x2000 (NOT Allowed)
New region:      0x0000 <-> 0x4000 (NOT Allowed)
New region:      0x1000 <-> 0x4000 (NOT Allowed)
New region:      0x2000 <-> 0x4000 (NOT Allowed)
```

参数

- **caps** -- Ordered array of capability masks for the new region, in order of priority. Must have length SOC_MEMORY_TYPE_NO_PRIOS. Does not need to remain valid after the call returns.
- **start** -- Start address of new region.
- **end** -- End address of new region.

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if a parameter is invalid
- ESP_ERR_NO_MEM if no memory to register new heap.
- ESP_ERR_INVALID_SIZE if the memory region is too small to fit a heap
- ESP_FAIL if region overlaps the start and/or end of an existing region

API 参考 - 多堆 API

（注意：堆属性分配器在内部使用多堆 API，而多数 IDF 程序不需要直接调用此 API。）

Header File

- components/heap/include/multi_heap.h
- This header file can be included with:

```
#include "multi_heap.h"
```

Functions

void ***multi_heap_aligned_alloc**(*multi_heap_handle_t* heap, size_t size, size_t alignment)

allocate a chunk of memory with specific alignment

参数

- **heap** -- Handle to a registered heap.
- **size** -- size in bytes of memory chunk
- **alignment** -- how the memory must be aligned

返回 pointer to the memory allocated, NULL on failure

void ***multi_heap_malloc**(*multi_heap_handle_t* heap, size_t size)

malloc() a buffer in a given heap

Semantics are the same as standard malloc(), only the returned buffer will be allocated in the specified heap.

参数

- **heap** -- Handle to a registered heap.
- **size** -- Size of desired buffer.

返回 Pointer to new memory, or NULL if allocation fails.

void **multi_heap_aligned_free**(*multi_heap_handle_t* heap, void *p)

free() a buffer aligned in a given heap.

备注: This function is deprecated, consider using multi_heap_free() instead

---

参数

- **heap** -- Handle to a registered heap.
- **p** -- NULL, or a pointer previously returned from multi_heap_aligned_alloc() for the same heap.

void **multi_heap_free**(*multi_heap_handle_t* heap, void *p)

free() a buffer in a given heap.

Semantics are the same as standard free(), only the argument 'p' must be NULL or have been allocated in the specified heap.

参数

- **heap** -- Handle to a registered heap.
- **p** -- NULL, or a pointer previously returned from multi_heap_malloc() or multi_heap_realloc() for the same heap.

void ***multi_heap_realloc**(*multi_heap_handle_t* heap, void *p, size_t size)

realloc() a buffer in a given heap.

Semantics are the same as standard realloc(), only the argument 'p' must be NULL or have been allocated in the specified heap.

参数

- **heap** -- Handle to a registered heap.

- **p** -- NULL, or a pointer previously returned from multi_heap_malloc() or multi_heap_realloc() for the same heap.
- **size** -- Desired new size for buffer.

返回 New buffer of 'size' containing contents of 'p', or NULL if reallocation failed.

size_t **multi_heap_get_allocated_size**(*multi_heap_handle_t* heap, void *p)

Return the size that a particular pointer was allocated with.

参数

- **heap** -- Handle to a registered heap.
- **p** -- Pointer, must have been previously returned from multi_heap_malloc() or multi_heap_realloc() for the same heap.

返回 Size of the memory allocated at this block. May be more than the original size argument, due to padding and minimum block sizes.

multi_heap_handle_t **multi_heap_register**(void *start, size_t size)

Register a new heap for use.

This function initialises a heap at the specified address, and returns a handle for future heap operations.

There is no equivalent function for deregistering a heap - if all blocks in the heap are free, you can immediately start using the memory for other purposes.

参数

- **start** -- Start address of the memory to use for a new heap.
- **size** -- Size (in bytes) of the new heap.

返回 Handle of a new heap ready for use, or NULL if the heap region was too small to be initialised.

void **multi_heap_set_lock**(*multi_heap_handle_t* heap, void *lock)

Associate a private lock pointer with a heap.

The lock argument is supplied to the MULTI_HEAP_LOCK() and MULTI_HEAP_UNLOCK() macros, defined in multi_heap_platform.h.

The lock in question must be recursive.

When the heap is first registered, the associated lock is NULL.

参数

- **heap** -- Handle to a registered heap.
- **lock** -- Optional pointer to a locking structure to associate with this heap.

void **multi_heap_dump**(*multi_heap_handle_t* heap)

Dump heap information to stdout.

For debugging purposes, this function dumps information about every block in the heap to stdout.

参数 **heap** -- Handle to a registered heap.

bool **multi_heap_check**(*multi_heap_handle_t* heap, bool print_errors)

Check heap integrity.

Walks the heap and checks all heap data structures are valid. If any errors are detected, an error-specific message can be optionally printed to stderr. Print behaviour can be overridden at compile time by defining MULTI_CHECK_FAIL_PRINTF in multi_heap_platform.h.

备注: This function is not thread-safe as it sets a global variable with the value of print_errors.

参数

- **heap** -- Handle to a registered heap.
- **print_errors** -- If true, errors will be printed to stderr.

返回 true if heap is valid, false otherwise.

size_t **multi_heap_free_size**(*multi_heap_handle_t* heap)

Return free heap size.

Returns the number of bytes available in the heap.

Equivalent to the total_free_bytes member returned by multi_heap_get_heap_info().

Note that the heap may be fragmented, so the actual maximum size for a single malloc() may be lower. To know this size, see the largest_free_block member returned by multi_heap_get_heap_info().

参数 **heap** -- Handle to a registered heap.
返回 Number of free bytes.

size_t **multi_heap_minimum_free_size**(*multi_heap_handle_t* heap)

Return the lifetime minimum free heap size.

Equivalent to the minimum_free_bytes member returned by multi_heap_get_info().

Returns the lifetime "low watermark" of possible values returned from multi_free_heap_size(), for the specified heap.

参数 **heap** -- Handle to a registered heap.
返回 Number of free bytes.

void **multi_heap_get_info**(*multi_heap_handle_t* heap, *multi_heap_info_t* *info)

Return metadata about a given heap.

Fills a *multi_heap_info_t* structure with information about the specified heap.

参数

- **heap** -- Handle to a registered heap.
- **info** -- Pointer to a structure to fill with heap metadata.

void ***multi_heap_aligned_alloc_offs**(*multi_heap_handle_t* heap, size_t size, size_t alignment, size_t offset)

Perform an aligned allocation from the provided offset.

参数

- **heap** -- The heap in which to perform the allocation
- **size** -- The size of the allocation
- **alignment** -- How the memory must be aligned
- **offset** -- The offset at which the alignment should start

返回 void* The ptr to the allocated memory

size_t **multi_heap_reset_minimum_free_bytes**(*multi_heap_handle_t* heap)

Reset the minimum_free_bytes value (setting it to free_bytes) and return the former value.

参数 **heap** -- The heap in which the reset is taking place
返回 size_t the value of minimum_free_bytes before it is reset

void **multi_heap_restore_minimum_free_bytes**(*multi_heap_handle_t* heap, const size_t new_minimum_free_bytes_value)

Set the value of minimum_free_bytes to new_minimum_free_bytes_value or keep the current value of minimum_free_bytes if it is smaller than new_minimum_free_bytes_value.

参数

- **heap** -- The heap in which the restore is taking place
- **new_minimum_free_bytes_value** -- The value to restore the minimum_free_bytes to

void **multi_heap_walk**(*multi_heap_handle_t* heap, *multi_heap_walker_cb_t* walker_func, void *user_data)

Call the tlsf_walk_pool function of the heap given as parameter with the walker function passed as parameter.

参数

- **heap** -- The heap to traverse
- **walker_func** -- The walker to trigger on each block of the heap

- **user_data** -- Opaque pointer to user defined data

Structures

struct **multi_heap_info_t**

Structure to access heap metadata via multi_heap_get_info.

Public Members

size_t **total_free_bytes**

Total free bytes in the heap. Equivalent to multi_free_heap_size().

size_t **total_allocated_bytes**

Total bytes allocated to data in the heap.

size_t **largest_free_block**

Size of the largest free block in the heap. This is the largest malloc-able size.

size_t **minimum_free_bytes**

Lifetime minimum free heap size. Equivalent to multi_minimum_free_heap_size().

size_t **allocated_blocks**

Number of (variable size) blocks allocated in the heap.

size_t **free_blocks**

Number of (variable size) free blocks in the heap.

size_t **total_blocks**

Total number of (variable size) blocks in the heap.

Type Definitions

typedef struct multi_heap_info *__multi_heap_handle_t__

Opaque handle to a registered heap.

typedef bool (*__multi_heap_walker_cb_t__)(void *block_ptr, size_t block_size, int block_used, void *user_data)

Callback called when walking the given heap blocks of memory.

Param block_ptr Pointer to the block data
Param block_size The size of the block
Param block_used Block status. 0: free, 1: allocated
Param user_data Opaque pointer to user defined data
Return True if the walker is expected to continue the heap traversal False if the walker is expected to stop the traversal of the heap

2.9.15 基于 MMU 的存储管理

简介

ESP32-P4 的存储管理单元 (MMU) 相对简单。它可以在物理存储地址和虚拟存储地址之间进行存储地址转换，因此，CPU 可以通过虚拟存储地址来访问物理存储。虚拟存储地址有多种类型，分别具有不同的功能。

ESP-IDF 提供了一个存储器映射驱动程序，用于管理这些物理存储地址和虚拟存储地址之间的关系，实现一些功能。例如，可以通过指针从 SPI flash 中读取内容。

存储映射驱动程序实际上是一个基于属性的虚拟存储地址分配器，允许应用程序为不同的目的分配虚拟存储地址。下文将此驱动程序称为 `esp_mmap` 驱动程序。

ESP-IDF 还提供了一个存储器同步驱动程序来处理潜在的不同步的情况。

物理存储类型

存储映射驱动程序目前支持映射到以下物理存储类型：

- SPI flash
- PSRAM

虚拟存储的功能

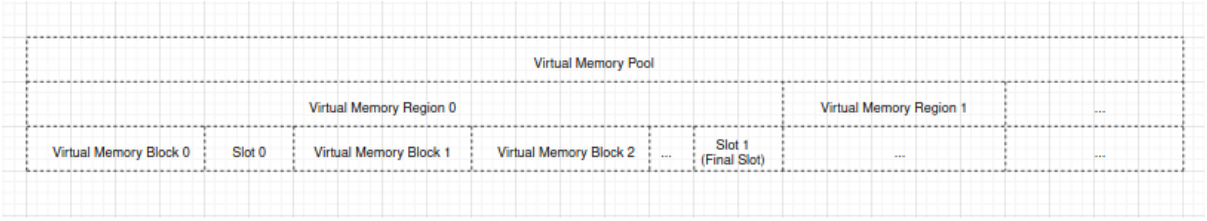
- `MMU_MEM_CAP_EXEC`：此功能表示虚拟存储地址具有执行权限。注意，此权限的范围仅限 MMU 硬件内部。
- `MMU_MEM_CAP_READ`：此功能表示虚拟存储地址具有读取权限。注意，此权限的范围仅限 MMU 硬件内部。
- `MMU_MEM_CAP_WRITE`：此功能表示虚拟存储地址具有写入权限。注意，此权限的范围仅限 MMU 硬件内部。
- `MMU_MEM_CAP_32BIT`：此功能表示允许访问 32 位或 32 位倍数的虚拟存储。
- `MMU_MEM_CAP_8BIT`：此功能表示允许访问 8 位或 8 位倍数的虚拟存储。

如需了解具有特定功能的最大连续可映射块大小，请调用*esp_mmu_map_get_max_consecutive_free_block_size()*。

存储管理驱动程序

驱动程序

术语解释 虚拟存储池由一个或多个虚拟存储区域组成，如下图所示：



- 一个虚拟存储池是指可以映射到物理存储的整个虚拟地址范围。
- 一个虚拟存储区域是指具有相同属性的虚拟地址范围。
- 一个虚拟存储块是指一块动态映射的虚拟地址范围。
- 一个间隙是指两个虚拟存储块之间的虚拟地址范围。
- 一个物理存储块是指一块待映射或已映射到虚拟存储块的物理地址范围。
- 动态映射是指调用 `esp_mmap` 驱动程序 API *esp_mmu_map()* 进行的映射，此 API 会将给定的物理存储块映射到 `esp_mmap` 驱动程序分配的虚拟存储块中。

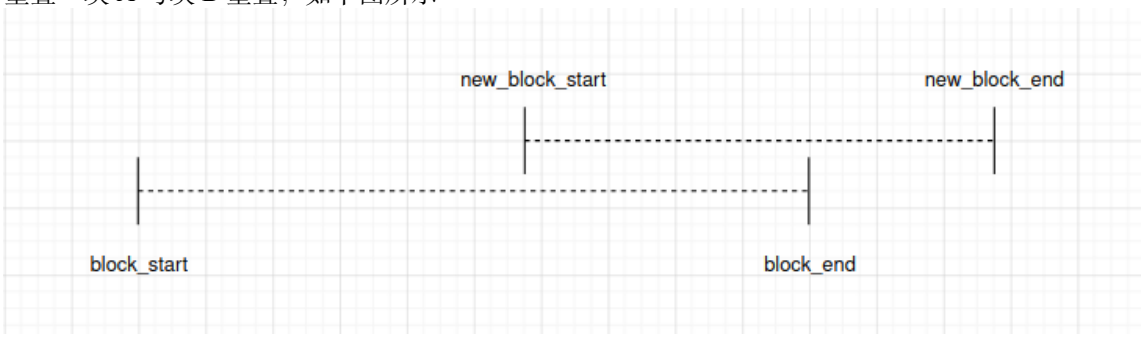
存储块的关系 在映射物理存储块 A 时，存储块 A 与此前已映射的另一个物理存储块 B 之间可能存在以下关系之一：

- 包含：块 A 被完全包含在块 B 内，如下图所示：

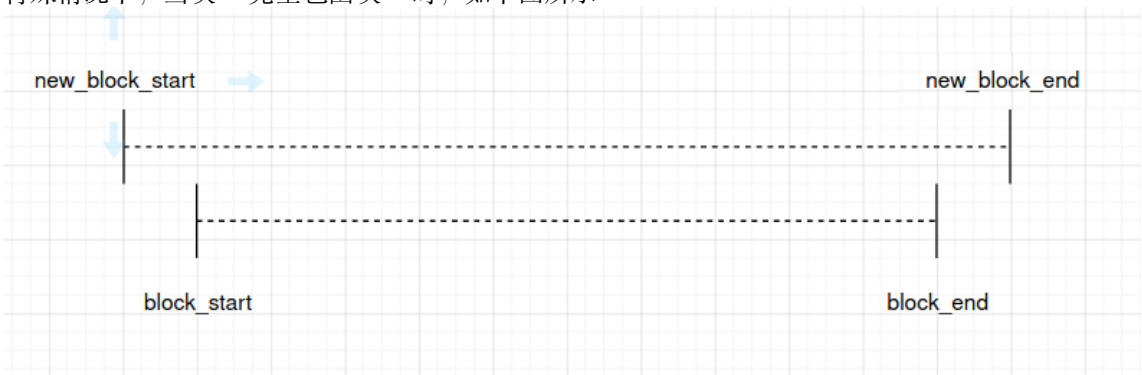
- 相同：块 A 与块 B 完全相同，如下图所示：

注意，`esp_mmap` 驱动程序将相同 **视为包含**。

- 重叠：块 A 与块 B 重叠，如下图所示：



特殊情况下，当块 A 完全包围块 B 时，如下图所示：



注意，`esp_mmap` 驱动程序将这种情况 **视为重叠**。

驱动程序行为

存储映射 可以调用*esp_mmu_map()* 进行动态映射。该 API 会根据你所选择的虚拟存储的属性分配一定大小的虚拟存储块，然后按照要求将此虚拟存储块映射到物理存储块中。esp_mmap 驱动程序支持映射到一种或多种物理存储，因此，应在映射时指定物理存储目标。

默认情况下，物理存储块和虚拟存储块是一对一映射。因此，当调用*esp_mmu_map()* 时，如果存储关系不同，API 的行为也有所不同：

- 如果是“包含”的情形，此 API 将返回*ESP_ERR_INVALID_STATE*。此时，由于之前已映射的虚拟存储包含待映射的虚拟存储，因此返回的 out_ptr 会指向之前映射的虚拟存储的起始地址。
- 如果是“相同”的情形，此 API 的行为与“包含”的情况完全相同。
- 如果是“重叠”的情形，此 API 默认返回*ESP_ERR_INVALID_ARG*。这表明，esp_mmap 驱动程序在默认情况下，不允许将一个物理存储地址映射到多个虚拟存储地址。

但是，可以使用*ESP_MMU_MMAP_FLAG_PADDR_SHARED* flag，代表在物理地址和虚拟地址之间的一对多映射：

- 如果是“重叠”的情形，此 API 将按照要求分配一个新的虚拟存储块，并将其映射到给定的物理存储块中。

取消存储映射 可以调用*esp_mmu_unmap()* 来取消先前存储块的映射。如果尝试取消映射的虚拟存储块实际尚未映射到任何物理存储块，此 API 将返回*ESP_ERR_NOT_FOUND*。

存储地址转换 esp_mmap 驱动程序提供了两个辅助 API，用于虚拟存储地址和物理存储地址之间的转换：

- *esp_mmu_vaddr_to_paddr()* 可将虚拟存储地址转换为物理存储地址。
- *esp_mmu_paddr_to_vaddr()* 可将物理存储地址转换为虚拟存储地址。

存储同步 可以通过一种或多种方法来访问支持 MMU 的物理存储。

SPI flash 可以通过 SPI1（ESP-IDF spi_flash 驱动 API）或指针进行访问。ESP-IDF spi_flash 驱动 API 中已经考虑了存储同步，因此在使用时无需额外考虑存储同步。

PSRAM 可以通过指针进行访问。当只通过指针访问 PSRAM 时，硬件可以保证数据的一致性。

线程安全

esp_mmu_map.h 中的 API 不能确保线程的安全性。

esp_cache.h 中的 API 能够确保线程的安全性。

API 参考

API 参考 - ESP MMAP 驱动

Header File

- components/esp_mm/include/esp_mmu_map.h
- This header file can be included with:

```
#include "esp_mmu_map.h"
```

- This header file is a part of the API provided by the esp_mm component. To declare that your component depends on esp_mm, add the following to your CMakeLists.txt:

```
REQUIRES esp_mm
```

or

```
PRIV_REQUIRES esp_mm
```

Functions

esp_err_t **esp_mmu_map**(*esp_paddr_t* paddr_start, size_t size, mmu_target_t target, mmu_mem_caps_t caps, int flags, void **out_ptr)

Map a physical memory block to external virtual address block, with given capabilities.

备注: This API does not guarantee thread safety

参数

- **paddr_start** -- **[in]** Start address of the physical memory block
- **size** -- **[in]** Size to be mapped. Size will be rounded up by to the nearest multiple of MMU page size
- **target** -- **[in]** Physical memory target you're going to map to, see `mmu_target_t`
- **caps** -- **[in]** Memory capabilities, see `mmu_mem_caps_t`
- **flags** -- **[in]** Mmap flags
- **out_ptr** -- **[out]** Start address of the mapped virtual memory

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument, see printed logs
- ESP_ERR_NOT_SUPPORTED: Only on ESP32, PSRAM is not a supported physical memory target
- ESP_ERR_NOT_FOUND: No enough size free block to use
- ESP_ERR_NO_MEM: Out of memory, this API will allocate some heap memory for internal usage
- ESP_ERR_INVALID_STATE: Paddr is mapped already, this API will return corresponding `vaddr_start + new_block_offset` as per the previously mapped block. Only to-be-mapped paddr block is totally enclosed by a previously mapped block will lead to this error. (Identical scenario will behave similarly) new_block_start new_block_end |------—New Block ------—| |------------—Block ------------—| block_start block_end

esp_err_t **esp_mmu_unmap**(void *ptr)

Unmap a previously mapped virtual memory block.

备注: This API does not guarantee thread safety

参数 **ptr** -- **[in]** Start address of the virtual memory

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Null pointer
- ESP_ERR_NOT_FOUND: Vaddr is not in external memory, or it's not mapped yet

esp_err_t **esp_mmu_map_get_max_consecutive_free_block_size**(mmu_mem_caps_t caps, mmu_target_t target, size_t *out_len)

Get largest consecutive free external virtual memory block size, with given capabilities and given physical target.

参数

- **caps** -- **[in]** Bitwise OR of MMU_MEM_CAP_* flags indicating the memory block

- **target** -- **[in]** Physical memory target you're going to map to, see `mmu_target_t`.
- **out_len** -- **[out]** Largest free block length, in bytes.

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Invalid arguments, could be null pointer

esp_err_t **esp_mmu_map_dump_mapped_blocks**(FILE *stream)

Dump all the previously mapped blocks

备注: This API shall not be called from an ISR.

备注: This API does not guarantee thread safety

参数 **stream** -- stream to print information to; use stdout or stderr to print to the console; use fmemopen/open_memstream to print to a string buffer.

返回

- ESP_OK

esp_err_t **esp_mmu_vaddr_to_paddr**(void *vaddr, *esp_paddr_t* *out_paddr, mmu_target_t *out_target)

Convert virtual address to physical address.

参数

- **vaddr** -- **[in]** Virtual address
- **out_paddr** -- **[out]** Physical address
- **out_target** -- **[out]** Physical memory target, see `mmu_target_t`

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Null pointer, or vaddr is not within external memory
- ESP_ERR_NOT_FOUND: Vaddr is not mapped yet

esp_err_t **esp_mmu_paddr_to_vaddr**(*esp_paddr_t* paddr, mmu_target_t target, mmu_vaddr_t type, void **out_vaddr)

Convert physical address to virtual address.

参数

- **paddr** -- **[in]** Physical address
- **target** -- **[in]** Physical memory target, see `mmu_target_t`
- **type** -- **[in]** Virtual address type, could be either instruction or data
- **out_vaddr** -- **[out]** Virtual address

返回

- ESP_OK
- ESP_ERR_INVALID_ARG: Null pointer
- ESP_ERR_NOT_FOUND: Paddr is not mapped yet

esp_err_t **esp_mmu_paddr_find_caps**(const *esp_paddr_t* paddr, mmu_mem_caps_t *out_caps)

If the physical address is mapped, this API will provide the capabilities of the virtual address where the physical address is mapped to.

备注: : Only return value is ESP_OK(which means physically address is successfully mapped), then caps you get make sense.

备注: This API only check one page (see CONFIG_MMU_PAGE_SIZE), starting from the `paddr`

参数

- **paddr** -- **[in]** Physical address
- **out_caps** -- **[out]** Bitwise OR of MMU_MEM_CAP_* flags indicating the capabilities of a virtual address where the physical address is mapped to.

返回

- ESP_OK: Physical address successfully mapped.
- ESP_ERR_INVALID_ARG: Null pointer
- ESP_ERR_NOT_FOUND: Physical address is not mapped successfully.

Macros

ESP_MMU_MMAP_FLAG_PADDR_SHARED

Share this mapping.

MMU Memory Mapping Driver APIs for MMU supported memory

Driver Backgrounds:

Type Definitions

typedef uint32_t **esp_paddr_t**

Physical memory type.

2.9.16 Memory Synchronization

Introduction

ESP32-P4 can access its connected PSRAM via these ways:

- CPU
- DMA

ESP32-P4 can access its internal memory via these ways:

- CPU
- DMA

By default, CPU accesses the above mentioned memory via cache. Whereas DMA accesses the memory directly, without going through cache.

This leads to potential cache data coherence issue:

- When a DMA transaction changes the content of a piece of memory, and the content has been cached already. Under this condition:
  - CPU may read stale data.
  - the stale data in the cache may be written back to the memory. The new data updated by the previous DMA transaction will be overwritten.
- CPU changes the content of an address. The content is in the cache, but not in the memory yet (cache will write back the content to the memory according to its own strategy). Under this condition:
  - The next DMA transactions to read this content from the memory will get stale data.

There are three common methods to address such cache data coherence issue:

1. Hardware based cache Coherent Interconnect, ESP32-P4 does not have such ability.
2. Use the DMA buffer from non-cacheable memory. Memory that CPU access it without going through cache is called non-cacheable memory.
3. Explicitly call a memory synchronization API to writeback the content in the cache back to the memory, or invalidate the content in the cache.

Memory Synchronisation Driver

The suggested way to deal with such cache data coherence issue is by using the memory synchronization API `esp_cache_msync()` provided by ESP-IDF *esp_mm* component.

Driver Concept Direction of the cache memory synchronization:

- `ESP_CACHE_MSYNC_FLAG_DIR_C2M`, for synchronization from cache to memory.
- `ESP_CACHE_MSYNC_FLAG_DIR_M2C`, for synchronization from memory to cache.

Type of the cache memory synchronization:

- `ESP_CACHE_MSYNC_FLAG_TYPE_DATA`, for synchronization to a data address region.
- `ESP_CACHE_MSYNC_FLAG_TYPE_INST`, for synchronization to an instruction address region.

Driver Behaviour Calling `esp_cache_msync()` will do a synchronization between cache and memory. The first parameter *addr* and the second parameter *size* together describe the memory region that is to be synchronized. About the third parameter *flags*:

- `ESP_CACHE_MSYNC_FLAG_DIR_C2M`. With this flag, content in the specified address region is written back to the memory. This direction is usually used **after** the content of an address is updated by the CPU, e.g. a memset to the address. Operation in this direction should happen **before** a DMA operation to the same address.
- `ESP_CACHE_MSYNC_FLAG_DIR_M2C`. With this flag, content in the specified address region is invalidated from the cache. This direction is usually used **after** the content of an address is updated by the DMA. Operation in this direction should happen **before** a CPU read operation to the same address.

The above two flags help select the synchronization direction. Specially, if neither of these two flags are used, `esp_cache_msync()` will by default select the `ESP_CACHE_MSYNC_FLAG_DIR_C2M` direction. Users are not allowed to set both of the two flags at the same time.

- `ESP_CACHE_MSYNC_FLAG_TYPE_DATA`.
- `ESP_CACHE_MSYNC_FLAG_TYPE_INST`.

The above two flags help select the type of the synchronization address. Specially, if neither of these two flags are used, `esp_cache_msync()` will by default select the `ESP_CACHE_MSYNC_FLAG_TYPE_DATA` direction. Users are not allowed to set both of the two flags at the same time.

- `ESP_CACHE_MSYNC_FLAG_INVALIDATE`. This flag is used to trigger a cache invalidation to the specified address region, after the region is written back to the memory. This flag is mainly used for `ESP_CACHE_MSYNC_FLAG_DIR_C2M` direction. For `ESP_CACHE_MSYNC_FLAG_DIR_M2C` direction, behaviour is the same as if the `ESP_CACHE_MSYNC_FLAG_INVALIDATE` flag is not set.
- `ESP_CACHE_MSYNC_FLAG_UNALIGNED`. This flag force the `esp_cache_msync()` API to do synchronization without checking the address and size alignment. For more details, see chapter *Address Alignment Requirement* following.

Address Alignment Requirement

There is address and size alignment requirement (in bytes) for using `esp_cache_msync()`. The alignment requirement comes from cache.

- An address region whose start address and size both meet the cache memory synchronization alignment requirement is defined as an **aligned address region**.
- An address region whose start address or size does not meet the cache memory synchronization alignment requirement is defined as an **unaligned address region**.

By default, if you specify an unaligned address region, `esp_cache_msync()` will return an `ESP_ERR_INVALID_ARG` error, together with the required alignment.

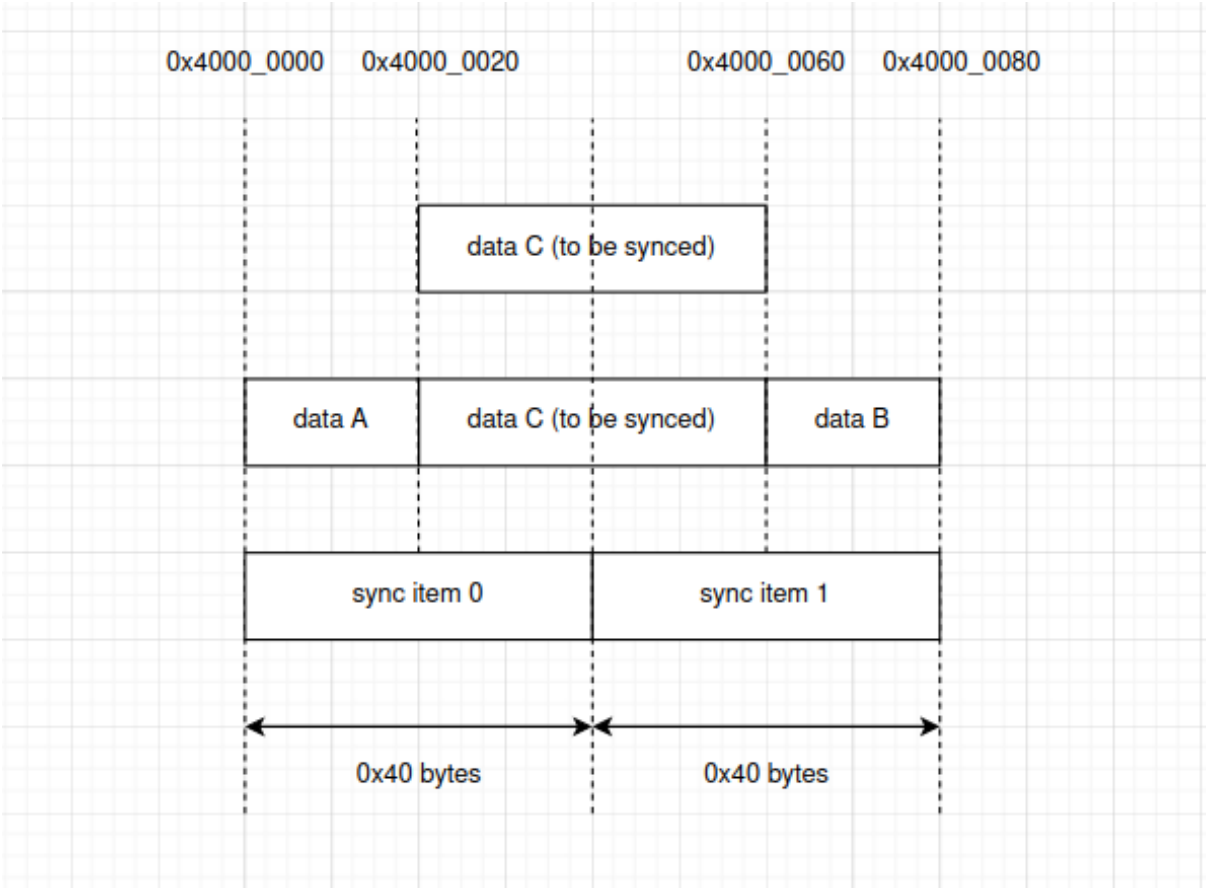
Warning for Address Alignment Requirement You can set the `ESP_CACHE_MSYNC_FLAG_UNALIGNED` flag to bypass such check. Note you should be very careful about using this flag. cache memory synchronization to an unaligned address region may silently corrupt the memory.

For example, assume:

- alignment requirement is 0x40 bytes.
- a call to `esp_cache_msync()`, with *ESP_CACHE_MSYNC_FLAG_DIR_M2C* | *ESP_CACHE_MSYNC_FLAG_UNALIGNED* flags, the specified address region is 0x4000_0020 ~ 0x4000_0060 (see **data C** in below graph).

Above settings will trigger a cache invalidation to the address region 0x4000_0000 ~ 0x4000_0080, see **sync item0** and **sync item1** in the below graph.

If the content in 0x4000_0000 ~ 0x4000_0020 (**data A** in the below graph) or 0x4000_0060 ~ 0x4000_0080 (**data B** in the below graph) are not written back to the memory yet, then these **data A** and **data B** will be discarded.



API Reference

API Reference - ESP Msync Driver

Header File

- components/esp_mm/include/esp_cache.h
- This header file can be included with:

```
#include "esp_cache.h"
```

- This header file is a part of the API provided by the `esp_mm` component. To declare that your component depends on `esp_mm`, add the following to your CMakeLists.txt:

```
REQUIRES esp_mm
```

or

```
PRIV_REQUIRES esp_mm
```

Functions

esp_err_t **esp_cache_msync**(void *addr, size_t size, int flags)

Memory sync between Cache and storage memory.

For cache-to-memory (C2M) direction:

- For cache writeback supported chips (you can refer to SOC_CACHE_WRITEBACK_SUPPORTED in soc_caps.h)
  - This API will do a writeback to synchronise between cache and storage memory
  - With ESP_CACHE_MSYNC_FLAG_INVALIDATE, this API will also invalidate the values that just written
  - Note: although ESP32 is with PSRAM, but cache writeback isn't supported, so this API will do nothing on ESP32
- For other chips, this API will do nothing. The out-of-sync should be already dealt by the SDK

For memory-to-cache (M2C) direction:

- This API will by default do an invalidation

This API is cache-safe and thread-safe

备注: If you don't set direction (ESP_CACHE_MSYNC_FLAG_DIR_x flags), this API is by default C2M direction

---

备注: If you don't set type (ESP_CACHE_MSYNC_FLAG_TYPE_x flags), this API is by default doing msync for data

---

备注: You should not call this during any Flash operations (e.g. esp_flash APIs, nvs and some other APIs that are based on esp_flash APIs)

备注: If XIP_From_PSRAM is enabled (by enabling both CONFIG_SPIRAM_FETCH_INSTRUCTIONS and CONFIG_SPIRAM_RODATA), you can call this API during Flash operations

---

参数

- **addr** -- **[in]** Starting address to do the msync
- **size** -- **[in]** Size to do the msync
- **flags** -- **[in]** Flags, see `ESP_CACHE_MSYNC_FLAG_x`

返回

- ESP_OK:
  - Successful msync
  - For C2M direction, if this chip doesn't support cache writeback, if the input addr is a cache supported one, this API will return ESP_OK
- ESP_ERR_INVALID_ARG: Invalid argument, not cache supported addr, see printed logs

Macros

ESP_CACHE_MSYNC_FLAG_INVALIDATE

Do an invalidation.

Cache msync flags

- For cache-to-memory (C2M) direction: setting this flag will start an invalidation after the cache writeback operation
- For memory-to-cache (M2C) direction: setting / unsetting this flag will behave similarly, trigger an invalidation

ESP_CACHE_MSYNC_FLAG_UNALIGNED

Allow msync to a address block that are not aligned to the data cache line size.

ESP_CACHE_MSYNC_FLAG_DIR_C2M

Cache msync direction: from Cache to memory.

备注： If you don't set direction (ESP_CACHE_MSYNC_FLAG_DIR_x flags), it is by default cache-to-memory (C2M) direction

---

ESP_CACHE_MSYNC_FLAG_DIR_M2C

Cache msync direction: from memory to Cache.

ESP_CACHE_MSYNC_FLAG_TYPE_DATA

Cache msync type: data.

备注： If you don't set type (ESP_CACHE_MSYNC_FLAG_TYPE_x flags), it is by default data type

---

ESP_CACHE_MSYNC_FLAG_TYPE_INST

Cache msync type: instruction.

2.9.17 堆内存调试

概述

ESP-IDF 集成了用于请求堆内存信息、堆内存损坏检测 和堆内存跟踪 的工具，有助于跟踪内存相关错误。有关堆内存分配器的基本信息，请参阅堆内存分配。

堆内存信息

要获取堆内存状态的相关信息，请调用以下函数：

- *heap_caps_get_free_size()* 返回不同属性内存的当前空闲内存。
- *heap_caps_get_largest_free_block()* 返回堆中最大的空闲块，也是当前可分配的最大内存块。跟踪此值并将其与总空闲堆对比，可以检测堆碎片化情况。
- *heap_caps_get_minimum_free_size()* 可以跟踪堆启动以来的“低水位”。
- *heap_caps_get_info()* 返回一个*multi_heap_info_t* 结构体，包含上述函数的信息，以及一些额外的特定堆内存数据（分配数量等）。

- *heap_caps_print_heap_info()* 将*heap_caps_get_info()* 返回的信息摘要打印到标准输出。
- *heap_caps_dump()* 和*heap_caps_dump_all()* 输出堆中每个内存块结构的详细信息。注意，这可能会产生大量输出。

堆内存分配与释放钩子函数

通过堆内存分配及释放检测钩子，可获取每次堆内存分配及释放操作成功的提示：

- 定义 esp_heap_trace_alloc_hook() 获取堆内存分配操作成功的提示
- 定义 esp_heap_trace_free_hook() 获取堆内存释放操作成功的提示

要启用此功能，请设置*CONFIG_HEAP_USE_HOOKS* 选项。esp_heap_trace_alloc_hook() 和 esp_heap_trace_free_hook() 具有弱声明（即 __attribute__((weak))），因此无需为这两个钩子提供声明。鉴于从 ISR 中分配和释放堆内存在技术上是可行的（**但强烈不建议**），esp_heap_trace_alloc_hook() 和 esp_heap_trace_free_hook() 可能会从 ISR 中调用。

不建议在钩子函数中执行（或调用 API 函数执行）阻塞操作或堆内存分配与释放。一般而言，最好保持代码简洁，避免在钩子函数中进行复杂计算。

要定义堆内存分配及释放钩子，请参阅如下示例：

```
#include "esp_heap_caps.h"

void esp_heap_trace_alloc_hook(void* ptr, size_t size, uint32_t caps)
{
  ...
}
void esp_heap_trace_free_hook(void* ptr)
{
  ...
}

void app_main()
{
    ...
}
```

堆内存损坏检测

堆内存损坏检测可检测到各类堆内存错误，包括：

- 越界写入和缓冲区溢出
- 写入已释放的内存
- 从已释放或未初始化的内存读取

断言 如 heap/multi_heap.c 等堆的实现方式包含许多断言，堆内存损坏则断言失败。为高效检测堆内存损坏，请确保在项目配置中通过*CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL* 选项启用断言。

如果堆完整性断言失败，将打印一行类似 CORRUPT HEAP: multi_heap.c:225 detected at 0x3ffbb71c 的内容，打印的内存地址即内容损坏的堆结构地址。

调用*heap_caps_check_integrity_all()* 或相关函数可手动检测堆内存完整性，该函数可以检测所有请求的堆内存完整性，在禁用断言时仍可生效。若此完整性检测检测到错误，将打印相应错误及内容损坏的堆内存结构地址。

内存分配失败钩子 用户可以使用*heap_caps_register_failed_alloc_callback()* 注册回调函数，每次内存分配操作失败时都会调用该函数。

此外，若启用*CONFIG_HEAP_ABORT_WHEN_ALLOCATION_FAILS* 选项，可以在任何分配操作失败时，自动中止系统。

要注册内存分配失败的回调函数，请参阅如下示例：

```
#include "esp_heap_caps.h"

void heap_caps_alloc_failed_hook(size_t requested_size, uint32_t caps, const char␣
↪*function_name)
{
  printf("%s was called but failed to allocate %d bytes with 0x%X capabilities. \n
↪",function_name, requested_size, caps);
}

void app_main()
{
    ...
    esp_err_t error = heap_caps_register_failed_alloc_callback(heap_caps_alloc_
↪failed_hook);
    ...
    void *ptr = heap_caps_malloc(allocation_size, MALLOC_CAP_DEFAULT);
    ...
}
```

定位堆内存损坏 内存损坏可能是最难定位和修复的错误类型之一，因为导致内存损坏的原因可能与问题的表现毫不相干。以下是有关定位堆内存损坏的一些提示：

- 如果系统崩溃并提示 `CORRUPT HEAP:`，打印信息中通常包含栈跟踪，但此栈跟踪往往无效，因为系统会在检测到堆内存损坏后崩溃，但实际的损坏通常发生在其他位置，且损坏时间早于系统发现的时间。
- 将堆内存调试*配置项* 级别增加到“轻度影响”或“全面”可以得到更准确的信息，定位首个内存损坏的地址。
- 在代码中定期调用*heap_caps_check_integrity_all()* 或*heap_caps_check_integrity_addr()* 可以定位内存损坏发生的确切时间。可以反复调整检测函数位置，以定位导致堆内存损坏的代码块。
- 根据损坏的内存地址，按照*JTAG 调试* 在此地址上设置监视点，并在写入时使 CPU 暂停。
- 如果没有 JTAG，但大致了解损坏发生的时间，则可以通过*esp_cpu_set_watchpoint()* 在软件中提前设置监视点，触发监视点将导致致命错误。函数使用示例为 `esp_cpu_set_watchpoint(0, (void *)addr, 4, ESP_WATCHPOINT_STORE)`。注意，监视点在各个 CPU 独立存在，并且仅设置在当前运行的 CPU 上，因此，若无法确定哪个 CPU 破坏了内存，则需要在两个 CPU 上分别调用此函数。
- 对于缓冲区溢出，以 `HEAP_TRACE_ALL` 模式进行*堆内存跟踪* 可以看到哪些调用函数正在从堆中分配哪些地址，详情请参阅*堆内存跟踪定位堆内存损坏*。如果可以找到在已损坏地址的前一地址分配内存的函数，这些函数很可能就是使缓冲区溢出的函数。
- 调用*heap_caps_dump()* 或*heap_caps_dump_all()* 提示损坏区域周围堆块的情况，并了解哪些堆块可能已经溢出或下溢等。

配置项 暂时提高堆内存损坏检测级别，可以进一步获取有关堆内存损坏错误的详细信息。

在项目配置菜单中，可以在 `Component config` 下找到 `Heap memory debugging` 菜单，其中的*CONFIG_HEAP_CORRUPTION_DETECTION* 选项可以设置为以下三种级别：

基本模式（无污染） 此为默认级别，默认情况下，不会启用任何特殊的堆内存损坏检测功能。但会启用提供的断言。如果堆的任何内部数据结构出现覆盖或损坏，就会打印出一个堆内存损坏错误。这通常表示缓冲区溢出或越界写入。

启用断言时，如果出现重复释放相同内存的情况（即双重释放），则会触发断言。

在基本模式调用*heap_caps_check_integrity()*，可以检查所有堆结构的完整性，并在出错时打印错误信息。

轻微影响模式 在此级别下，每个分配的内存块都会在头尾加入“canary 字节”进行“污染”。如果应用程序在已分配缓冲区的边界外写入数据，则会破坏这些 canary 字节，导致完整性检查失败。

头 canary 字的值为 `0xABBA1234`（按字节顺序为 `3412BAAB`），尾 canary 字的值为 `0xBAAD5678`（按字节顺序为 `7856ADBA`）。

基本模式下的堆内存损坏检测可以检测到大多数越界写入，在检测到错误前的越界字节数取决于堆属性。但轻微影响模式更精确，可以检测到单个字节的越界写入。

启用轻微影响模式检测会增加内存使用量，每次内存分配需要 9 至 12 个额外的字节，具体取决于对齐方式。

在轻微影响模式下，每次调用*heap_caps_free()* 时，都会检查要释放的缓冲区头尾 canary 字节是否匹配预期值。

调用*heap_caps_check_integrity()* 时，会检查所有已分配的堆内存块的 canary 字节是否匹配预期值。

以上两种情况检查的是，在缓冲区返回给用户之前，已分配块的前 4 个字节是否为 `0xABBA1234`，以及在缓冲区返回给用户之后，最后 4 个字节是否为 `0xBAAD5678`。

如果检查到字节与上述值不同，通常表示缓冲区越界或下溢。其中越界表示在写入内存时，写入的数据超过了所分配内存的大小，导致写入到了未分配的内存区域；下溢表示在读取内存时，读取的数据超出了所分配内存的范围，读取了未分配的内存区域的数据。

全面检测模式 此级别包含了轻微影响模式的检测功能，此外还会检查未初始化访问和使用已释放内存产生的错误。此模式会将所有新分配的内存填充为 `0xCE`，将所有已释放的内存填充为 `0xFE`。

启用全面检测模式会对运行性能产生实质影响，因为每次*heap_caps_malloc()* 或*heap_caps_free()* 操作完成时，都需要将所有内存设置为分配模式，并检查内存。但是，此模式更容易检测到其他方式难以发现的内存损坏错误。建议只在调试时启用此模式，请勿在生产环境中启用。

全面检测模式下程序崩溃 全面检测模式下，如果应用程序在读取或写入与 `0xCECECECE` 相关地址时崩溃，表示它读取了未初始化内存。此时，应修改应用程序，使用*heap_caps_calloc()* 将内存清零，或在使用前初始化内存。在栈分配的自动变量中也可能存在 `0xCECECECE` 的值，因为 ESP-IDF 中的大多数任务栈最初由堆分配，而在 C 中，栈内存默认未初始化。

如果应用程序崩溃，且异常寄存器转储指示某些地址或值为 `0xFEFEFEFE`，表示它读取了已释放的堆内存。此时，应修改应用程序，避免访问已释放的堆内存。

调用*heap_caps_malloc()* 或*heap_caps_realloc()* 时，如果在已释放的内存中找到了不同于 `0xFEFEFEFE` 的值，将导致应用程序崩溃。这表示应用程序写入了已经释放的内存，从而产生错误。

全面检测模式下手动堆内存检测 调用*heap_caps_check_integrity()* 会打印与 `0xFEFEFEFE`、`0xABBA1234`、或 `0xBAAD5678` 相关的错误。在不同情况下，检测器均会检测给定模式，若未找到，则输出相应错误：

- 对于已释放的堆内存块，检测器会检查是否所有字节都设置为 `0xFE`，检测到任何其他值都表示错误写入了已释放内存。
- 对于已分配的堆内存块，检测器的检查模式与轻微影响模式相同，即在每个分配的缓冲区头部和尾部检查 canary 字节 `0xABBA1234` 和 `0xBAAD5678`，检测到任何其他字节都表示缓冲区越界或下溢。

堆任务跟踪

堆任务跟踪可获取堆内存分配的各任务信息，应用程序须指定计划跟踪堆分配的堆属性。

示例代码可参考 system/heap_task_tracking。

堆内存跟踪

堆内存跟踪支持跟踪用于分配或释放内存的代码，且支持以下两种跟踪模式：

- 独立模式。此模式下，跟踪数据保存在设备上（因此收集的信息大小受指定缓冲区限制），并由设备上的代码完成分析。部分 API 可访问和转储收集的信息。
- 主机模式。此模式不受独立模式所受限制，其跟踪数据使用 app_trace 库通过 JTAG 连接发送到主机，随后使用特殊工具完成分析。

堆内存跟踪具有以下两种功能：

- 泄漏检测：检测已分配但未释放的内存。
- 堆内存使用分析：显示在跟踪运行期间所有分配或释放内存的函数。

如何判断内存泄漏 如果怀疑存在内存泄漏，首先要找出程序中存在泄漏的部分。调用 *heap_caps_get_free_size()* 或堆内存信息 中的其他相关函数，跟踪应用程序的内存使用情况，尝试将泄漏范围缩小到某个或一系列空闲内存始终减少而没有恢复的函数。

独立模式 确定存在泄漏的代码后，请执行以下步骤：

- 启用 *CONFIG_HEAP_TRACING_DEST* 选项。
- 在程序早期调用函数 *heap_trace_init_standalone()* 注册一个可用于记录内存跟踪的缓冲区。
- 在有内存泄漏之嫌的代码块前，调用函数 *heap_trace_start()* 记录系统中的所有内存分配和释放操作。
- 在有内存泄露之嫌的代码块后，调用函数 *heap_trace_stop()* 停止跟踪。
- 调用函数 *heap_trace_dump()* 导出内存跟踪结果。

应用程序代码初始化、启动和停止堆内存跟踪的一般过程，见以下代码片段示例：

```
#include "esp_heap_trace.h"

#define NUM_RECORDS 100
static heap_trace_record_t trace_record[NUM_RECORDS]; // 该缓冲区必须在内部 RAM 中

...

void app_main()
{
    ...
    ESP_ERROR_CHECK( heap_trace_init_standalone(trace_record, NUM_RECORDS) );
    ...
}

void some_function()
{
    ESP_ERROR_CHECK( heap_trace_start(HEAP_TRACE_LEAKS) );

    do_something_you_suspect_is_leaking();

    ESP_ERROR_CHECK( heap_trace_stop() );
    heap_trace_dump();
    ...
}
```

堆内存跟踪堆输出将类似以下格式的内容：

```
2 allocations trace (100 entry buffer)
32 bytes (@ 0x3ffaf214) allocated CPU 0 ccount 0x2e9b7384 caller
8 bytes (@ 0x3ffaf804) allocated CPU 0 ccount 0x2e9b79c0 caller
40 bytes 'leaked' in trace (2 allocations)
total allocations 2 total frees 0
```

备注： 以上示例输出使用*IDF 监视器*，自动将 PC 地址解码为其源文件和行号。

第一行表示与缓冲区的总大小相比，缓冲区内的分配条目数量。

在 HEAP_TRACE_LEAKS 模式下，对跟踪的每个未释放的已分配内存，打印的信息中都会包含以下内容：

- XX bytes：分配的字节数。
- @ 0x...：从*heap_caps_malloc()* 或*heap_caps_calloc()* 返回的堆地址。
- Internal 或 PSRAM：分配内存的一般位置。
- CPU x：分配过程中运行的 CPU（CPU 0 或 CPU 1）。
- ccount 0x...：分配时的 CCOUNT（CPU 循环计数器）寄存器值，CPU 0 与 CPU 1 中的这一值不同。

最后，将打印“泄漏”的总字节数（即在跟踪期间分配但未释放的总字节数），以及它所代表的总分配次数。

如果跟踪缓冲区不足以容纳所有分配，则会打印警告。如果看到此警告，请考虑缩短跟踪时间，或增加跟踪缓冲区中记录的数量。

主机模式 确定存在泄漏的代码后，请执行以下步骤：

- 在项目配置菜单中，导航至 Component settings > Heap Memory Debugging > *CONFIG_HEAP_TRACING_DEST* 并选择 Host-Based。
- 在项目配置菜单中，导航至 Component settings > Application Level Tracing > *CONFIG_APPTRACE_DESTINATION1* 并选择 Trace memory。
- 在项目配置菜单中，导航至 Component settings > Application Level Tracing > FreeRTOS SystemView Tracing 并启用*CONFIG_APPTRACE_SV_ENABLE*。
- 在程序早期，调用函数*heap_trace_init_tohost()*，初始化 JTAG 堆内存跟踪模块。
- 在有内存泄漏之嫌的代码块前，调用函数*heap_trace_start()* 开始记录系统中的内存分配和释放操作。
  主机模式忽略该函数参数，堆内存跟踪模块以 HEAP_TRACE_ALL 传递后的方式运行，即所有的内存分配和释放都发送到主机。
- 在有内存泄露之嫌的代码块后，调用函数*heap_trace_stop()* 停止跟踪。

应用程序代码初始化、启动和停止基于主机模式堆内存跟踪的一般过程，请参阅以下代码片段示例：

```
#include "esp_heap_trace.h"

...

void app_main()
{
    ...
    ESP_ERROR_CHECK( heap_trace_init_tohost() );
    ...
}

void some_function()
{
    ESP_ERROR_CHECK( heap_trace_start(HEAP_TRACE_LEAKS) );

    do_something_you_suspect_is_leaking();

    ESP_ERROR_CHECK( heap_trace_stop() );
    ...
}
```

要收集并分析堆内存跟踪结果，请在主机上完成以下操作：

1. 构建程序并将其下载到目标设备，详情请参阅*第五步：开始使用 ESP-IDF 吧*。

2. 运行 OpenOCD（请参阅*JTAG 调试*）。

备注： 使用此功能需要 `v0.10.0-esp32-20181105` 或更高版本的 OpenOCD。

3. 使用 GDB 可以自动启动和/或停止跟踪，为此应准备特殊的 `gdbinit` 文件：

```
target remote :3333

mon reset halt
maintenance flush register-cache

tb heap_trace_start
commands
mon esp sysview start file:///tmp/heap.svdat
c
end

tb heap_trace_stop
commands
mon esp sysview stop
end

c
```

使用此文件，GDB 将连接到目标设备、重置该设备，在程序触发*heap_trace_start()* 断点时开始跟踪，在程序触发*heap_trace_stop()* 断点时停止跟踪。跟踪数据将保存至 `/tmp/heap_log.svdat`。

4. 使用命令 `riscv32-esp-elf-gdb -x gdbinit </path/to/program/elf>` 运行 GDB
5. 调用*heap_trace_stop()* 函数使程序停止运行时，退出 GDB，跟踪数据将保存至 `/tmp/heap.svdat`
6. 运行处理脚本 `$IDF_PATH/tools/esp_app_trace/sysviewtrace_proc.py -p -b </path/to/program/elf> /tmp/heap_log.svdat`

堆内存跟踪堆输出将类似以下格式的内容：

```
Parse trace from '/tmp/heap.svdat'...
Stop parsing trace. (Timeout 0.000000 sec while reading 1 byte!)
Process events from '['/tmp/heap.svdat']'...
[0.002244575] HEAP: Allocated 1 byte @ 0x3ffaffd8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.002258425] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.002563725] HEAP: Freed bytes @ 0x3ffaffe0 from the task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.002782950] HEAP: Freed bytes @ 0x3ffb40b8 from the task "main" on core 0 by:
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590

[0.002798700] HEAP: Freed bytes @ 0x3ffb50bc from the task "main" on core 0 by:
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590
```

(下页继续)

(续上页)

```
[0.102436025] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.102449800] HEAP: Allocated 4 bytes @ 0x3ffaffe8 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.102666150] HEAP: Freed bytes @ 0x3ffaffe8 from the task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202436200] HEAP: Allocated 3 bytes @ 0x3ffaffe8 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202451725] HEAP: Allocated 6 bytes @ 0x3ffafff0 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202667075] HEAP: Freed bytes @ 0x3ffafff0 from the task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302436000] HEAP: Allocated 4 bytes @ 0x3ffafff0 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302451475] HEAP: Allocated 8 bytes @ 0x3ffb40b8 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302667500] HEAP: Freed bytes @ 0x3ffb40b8 from the task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

Processing completed.

Processed 1019 events

=============== HEAP TRACE REPORT ===============

Processed 14 heap events.

[0.002244575] HEAP: Allocated 1 bytes @ 0x3ffaffd8 from the task "alloc" on core 0␣
↪by:
```

(下页继续)

(续上页)

```
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.102436025] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202436200] HEAP: Allocated 3 bytes @ 0x3ffaffe8 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302436000] HEAP: Allocated 4 bytes @ 0x3ffafff0 from the task "alloc" on core 0␣
↪by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

Found 10 leaked bytes in 4 blocks.
```

堆内存跟踪定位堆内存损坏 堆内存跟踪也是一种定位堆内存损坏位置的方法。当堆中的某个区域损坏时，可能是因为程序中的其他部分在相邻地址分配内存。

如果大致了解堆内存损坏发生的时间范围，启用 `HEAP_TRACE_ALL` 模式的堆内存跟踪，可以记录分配内存的所有函数及其相应地址。

以此方法使用堆内存跟踪与上文描述的内存泄漏检测类似，对已分配但未释放的内存输出结果相同，但还会显示已释放内存的记录。

性能影响 在 menuconfig 中启用堆内存跟踪，会增加程序代码的大小，即便未运行堆内存跟踪，也会对堆内存分配或释放操作的性能产生较小的负面影响。

运行堆内存跟踪时，堆内存分配或释放操作的速度明显变慢。增加为各内存分配的栈帧深度（见上文）也会造成这种性能影响。

为减轻堆内存跟踪运行时的性能损失，请启用*CONFIG_HEAP_TRACE_HASH_MAP*。此时，将使用哈希映射机制处理堆内存跟踪记录，减少堆内存分配或释放操作的执行时长。设置*CONFIG_HEAP_TRACE_HASH_MAP_SIZE* 的值可以调整哈希映射的大小。

默认情况下，哈希映射会放置在内部 RAM 中，启用*CONFIG_HEAP_TRACE_HASH_MAP_IN_EXT_RAM* 时也可将其放置在外部 RAM 中。要启用此配置，请确保已启用*CONFIG_SPIRAM* 和*CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY*。

内存泄漏误报 并非所有由*heap_trace_dump()* 打印的信息都是内存泄漏，以下情况也可能打印信息：

- 在调用*heap_trace_start()* 后分配且在调用*heap_trace_stop()* 后才释放的内存都会出现在泄漏信息中。
- 系统中的其他任务也可能进行内存分配。根据这些任务的时间安排，报错的这部分内存很可能在调用*heap_trace_stop()* 后释放。
- 当任务第一次使用 stdio，如调用 `heap_caps_printf()` 时，libc 会分配一个锁，即 RTOS 互斥信号量，该分配将持续至任务删除。
- 进行打印浮点数等调用 `heap_caps_printf()` 的操作时，会根据需要，从堆中分配一些内存，这些分配将持续至任务删除。

- 蓝牙、Wi-Fi 和 TCP/IP 库会分配堆内存缓冲区，处理传入或传出的数据，这些内存缓冲区通常持续时间较短。但如果在运行堆内存泄漏跟踪期间，网络底层接收或发送了数据，一些缓冲区可能会出现在堆内存泄漏跟踪输出中。
- 由于存在 TIME_WAIT 状态，TCP 连接在关闭后仍会使用一些内存，TIME_WAIT 状态结束后将释放这些内存。

要区分“真实”和“误报”的内存泄漏，可以在堆内存跟踪运行时多次调用可疑代码，并在堆内存跟踪输出中查找重复出现的内存分配情况。

API 参考 - 堆内存跟踪

Header File

- components/heap/include/esp_heap_trace.h
- This header file can be included with:

```
#include "esp_heap_trace.h"
```

Functions

esp_err_t **heap_trace_init_standalone**(*heap_trace_record_t* *record_buffer, size_t num_records)

Initialise heap tracing in standalone mode.

This function must be called before any other heap tracing functions.

To disable heap tracing and allow the buffer to be freed, stop tracing and then call heap_trace_init_standalone(NULL, 0);

参数

- **record_buffer** -- Provide a buffer to use for heap trace data. Note: External RAM is allowed, but it prevents recording allocations made from ISR's.
- **num_records** -- Size of the heap trace buffer, as number of record structures.

返回

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing is currently in progress.
- ESP_OK Heap tracing initialised successfully.

esp_err_t **heap_trace_init_tohost**(void)

Initialise heap tracing in host-based mode.

This function must be called before any other heap tracing functions.

返回

- ESP_ERR_INVALID_STATE Heap tracing is currently in progress.
- ESP_OK Heap tracing initialised successfully.

esp_err_t **heap_trace_start**(*heap_trace_mode_t* mode)

Start heap tracing. All heap allocations & frees will be traced, until heap_trace_stop() is called.

备注： heap_trace_init_standalone() must be called to provide a valid buffer, before this function is called.

备注： Calling this function while heap tracing is running will reset the heap trace state and continue tracing.

参数 **mode** -- Mode for tracing.

- HEAP_TRACE_ALL means all heap allocations and frees are traced.
- HEAP_TRACE_LEAKS means only suspected memory leaks are traced. (When memory is freed, the record is removed from the trace buffer.)

返回

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE A non-zero-length buffer has not been set via heap_trace_init_standalone().
- ESP_OK Tracing is started.

esp_err_t **heap_trace_stop**(void)

Stop heap tracing.

返回

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing was not in progress.
- ESP_OK Heap tracing stopped..

esp_err_t **heap_trace_resume**(void)

Resume heap tracing which was previously stopped.

Unlike heap_trace_start(), this function does not clear the buffer of any pre-existing trace records.

The heap trace mode is the same as when heap_trace_start() was last called (or HEAP_TRACE_ALL if heap_trace_start() was never called).

返回

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing was already started.
- ESP_OK Heap tracing resumed.

size_t **heap_trace_get_count**(void)

Return number of records in the heap trace buffer.

It is safe to call this function while heap tracing is running.

esp_err_t **heap_trace_get**(size_t index, *heap_trace_record_t* *record)

Return a raw record from the heap trace buffer.

备注: It is safe to call this function while heap tracing is running, however in HEAP_TRACE_LEAK mode record indexing may skip entries unless heap tracing is stopped first.

参数

- **index** -- Index (zero-based) of the record to return.
- **record** -- **[out]** Record where the heap trace record will be copied.

返回

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing was not initialised.
- ESP_ERR_INVALID_ARG Index is out of bounds for current heap trace record count.
- ESP_OK Record returned successfully.

void **heap_trace_dump**(void)

Dump heap trace record data to stdout.

备注: It is safe to call this function while heap tracing is running, however in HEAP_TRACE_LEAK mode the dump may skip entries unless heap tracing is stopped first.

---

void **heap_trace_dump_caps**(const uint32_t caps)

Dump heap trace from the memory of the capabilities passed as parameter.

参数 caps -- Capability(ies) of the memory from which to dump the trace. Set MALLOC_CAP_INTERNAL to dump heap trace data from internal memory. Set MALLOC_CAP_SPIRAM to dump heap trace data from PSRAM. Set both to dump both heap trace data.

esp_err_t **heap_trace_summary** (*heap_trace_summary_t* *summary)

Get summary information about the result of a heap trace.

备注: It is safe to call this function while heap tracing is running.

Structures

struct **heap_trace_record_t**

Trace record data type. Stores information about an allocated region of memory.

Public Members

uint32_t **ccount**

CCOUNT of the CPU when the allocation was made. LSB (bit value 1) is the CPU number (0 or 1).

void ***address**

Address which was allocated. If NULL, then this record is empty.

size_t **size**

Size of the allocation.

void ***alloced_by**[CONFIG_HEAP_TRACING_STACK_DEPTH]

Call stack of the caller which allocated the memory.

void ***freed_by**[CONFIG_HEAP_TRACING_STACK_DEPTH]

Call stack of the caller which freed the memory (all zero if not freed.)

struct **heap_trace_summary_t**

Stores information about the result of a heap trace.

Public Members

heap_trace_mode_t **mode**

The heap trace mode we just completed / are running.

size_t **total_allocations**

The total number of allocations made during tracing.

size_t **total_frees**

The total number of frees made during tracing.

size_t **count**

The number of records in the internal buffer.

size_t **capacity**

The capacity of the internal buffer.

size_t **high_water_mark**

The maximum value that 'count' got to.

size_t **has_overflowed**

True if the internal buffer overflowed at some point.

Macros

CONFIG_HEAP_TRACING_STACK_DEPTH

Type Definitions

typedef struct *heap_trace_record_t* **heap_trace_record_t**

Trace record data type. Stores information about an allocated region of memory.

Enumerations

enum **heap_trace_mode_t**

Values:

enumerator **HEAP_TRACE_ALL**

enumerator **HEAP_TRACE_LEAKS**

2.9.18 ESP 定时器

API 参考

Header File

- components/esp_timer/include/esp_timer.h
- This header file can be included with:

```
#include "esp_timer.h"
```

- This header file is a part of the API provided by the `esp_timer` component. To declare that your component depends on `esp_timer`, add the following to your CMakeLists.txt:

```
REQUIRES esp_timer
```

or

```
PRIV_REQUIRES esp_timer
```

Functions

esp_err_t **esp_timer_early_init**(void)

Minimal initialization of esp_timer.

This function can be called very early in startup process, after this call only *esp_timer_get_time()* function can be used.

备注: This function is called from startup code. Applications do not need to call this function before using other esp_timer APIs.

返回

- ESP_OK on success

esp_err_t **esp_timer_init**(void)

Initialize esp_timer library.

This function will be called from startup code on every core. If Kconfig option `CONFIG_ESP_TIMER_ISR_AFFINITY` is set to `NO_AFFINITY`, it allocates the timer ISR on MULTIPLE cores and creates the timer task which can be run on any core.

备注: This function is called from startup code. Applications do not need to call this function before using other esp_timer APIs. Before calling this function, *esp_timer_early_init()* must be called by the startup code.

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if allocation has failed
- ESP_ERR_INVALID_STATE if already initialized
- other errors from interrupt allocator

esp_err_t **esp_timer_deinit**(void)

De-initialize esp_timer library.

备注: Normally this function should not be called from applications

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if not yet initialized

esp_err_t **esp_timer_create**(const *esp_timer_create_args_t* *create_args, *esp_timer_handle_t* *out_handle)

Create an esp_timer instance.

备注: When timer no longer needed, delete it using *esp_timer_delete()*.

参数

- **create_args** -- Pointer to a structure with timer creation arguments. Not saved by the library, can be allocated on the stack.
- **out_handle** -- **[out]** Output, pointer to esp_timer_handle_t variable that holds the created timer handle.

返回

- ESP_OK on success

- ESP_ERR_INVALID_ARG if some of the create_args are not valid
- ESP_ERR_INVALID_STATE if esp_timer library is not initialized yet
- ESP_ERR_NO_MEM if memory allocation fails

esp_err_t **esp_timer_start_once**(*esp_timer_handle_t* timer, uint64_t timeout_us)

Start a one-shot timer.

Timer represented by `timer` should not be running when this function is called.

参数

- **timer** -- timer handle created using *esp_timer_create()*
- **timeout_us** -- timer timeout, in microseconds relative to the current moment

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if the timer is already running

esp_err_t **esp_timer_start_periodic**(*esp_timer_handle_t* timer, uint64_t period)

Start a periodic timer.

Timer represented by `timer` should not be running when this function is called. This function starts the timer which will trigger every `period` microseconds.

参数

- **timer** -- timer handle created using *esp_timer_create()*
- **period** -- timer period, in microseconds

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if the timer is already running

esp_err_t **esp_timer_restart**(*esp_timer_handle_t* timer, uint64_t timeout_us)

Restart a currently running timer.

Type of `timer`	Action
One-shot timer	Restarted immediately and times out once in `timeout_us` microseconds
Periodic timer	Restarted immediately with a new period of `timeout_us` microseconds

参数

- **timer** -- timer handle created using *esp_timer_create()*
- **timeout_us** -- Timeout in microseconds relative to the current time. In case of a periodic timer, also represents the new period.

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if the timer is not running

esp_err_t **esp_timer_stop**(*esp_timer_handle_t* timer)

Stop a running timer.

This function stops the timer previously started using *esp_timer_start_once()* or *esp_timer_start_periodic()*.

参数 **timer** -- timer handle created using *esp_timer_create()*

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if the timer is not running

esp_err_t **esp_timer_delete**(*esp_timer_handle_t* timer)

Delete an esp_timer instance.

The timer must be stopped before deleting. A one-shot timer which has expired does not need to be stopped.

参数 **timer** -- timer handle created using *esp_timer_create()*

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if the timer is running

int64_t **esp_timer_get_time**(void)

Get time in microseconds since boot.

返回 Number of microseconds since the initialization of ESP Timer

int64_t **esp_timer_get_next_alarm**(void)

Get the timestamp of the next expected timeout.

返回 Timestamp of the nearest timer event, in microseconds. The timebase is the same as for the values returned by *esp_timer_get_time()*.

int64_t **esp_timer_get_next_alarm_for_wake_up**(void)

Get the timestamp of the next expected timeout excluding those timers that should not interrupt light sleep (such timers have *esp_timer_create_args_t::skip_unhandled_events* enabled)

返回 Timestamp of the nearest timer event, in microseconds. The timebase is the same as for the values returned by *esp_timer_get_time()*.

esp_err_t **esp_timer_get_period**(*esp_timer_handle_t* timer, uint64_t *period)

Get the period of a timer.

This function fetches the timeout period of a timer. For a one-shot timer, the timeout period will be 0.

参数

- **timer** -- timer handle created using *esp_timer_create()*
- **period** -- memory to store the timer period value in microseconds

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the arguments are invalid

esp_err_t **esp_timer_get_expiry_time**(*esp_timer_handle_t* timer, uint64_t *expiry)

Get the expiry time of a one-shot timer.

This function fetches the expiry time of a one-shot timer.

备注: Passing the timer handle of a periodic timer will result in an error.

参数

- **timer** -- timer handle created using *esp_timer_create()*
- **expiry** -- memory to store the timeout value in microseconds

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the arguments are invalid
- ESP_ERR_NOT_SUPPORTED if the timer type is periodic

esp_err_t **esp_timer_dump**(FILE *stream)

Dump the list of timers to a stream.

By default, this function prints the list of active (running) timers. The output format is:

| Name | Period | Alarm |

- Name —timer pointer
- Period —period of timer in microseconds, or 0 for one-shot timer
- Alarm - time of the next alarm in microseconds since boot, or 0 if the timer is not started

To print the list of all created timers, enable Kconfig option `CONFIG_ESP_TIMER_PROFILING`. In this case, the output format is:

| Name | Period | Alarm | Times_armed | Times_trigg | Times_skip | Cb_exec_time |

- Name —timer name
- Period —same as above
- Alarm —same as above
- Times_armed —number of times the timer was armed via esp_timer_start_X
- Times_triggered - number of times the callback was triggered
- Times_skipped - number of times the callback was skipped
- Callback_exec_time - total time taken by callback to execute, across all calls

参数 **stream** -- stream (such as stdout) to which to dump the information
返回
- ESP_OK on success
- ESP_ERR_NO_MEM if can not allocate temporary buffer for the output

void **esp_timer_isr_dispatch_need_yield**(void)

Requests a context switch from a timer callback function.

This only works for a timer that has an ISR dispatch method. The context switch will be called after all ISR dispatch timers have been processed.

bool **esp_timer_is_active**(*esp_timer_handle_t* timer)

Returns status of a timer, active or not.

This function is used to identify if the timer is still active (running) or not.

参数 **timer** -- timer handle created using *esp_timer_create()*
返回
- 1 if timer is still active (running)
- 0 if timer is not active

esp_err_t **esp_timer_new_etm_alarm_event**(*esp_etm_event_handle_t* *out_event)

Get the ETM event handle of esp_timer underlying alarm event.

备注: The created ETM event object can be deleted later using esp_etm_del_event()

备注: The ETM event is generated by the underlying hardware - systimer; therefore, if the esp_timer is not clocked by systimer, then no ETM event will be generated.

参数 **out_event** -- **[out]** Returned ETM event handle
返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Structures

struct **esp_timer_create_args_t**

Timer configuration passed to *esp_timer_create()*

Public Members

esp_timer_cb_t **callback**

Callback function to execute when timer expires.

void ***arg**

Argument to pass to callback.

esp_timer_dispatch_t **dispatch_method**

Dispatch callback from task or ISR; if not specified, esp_timer task.

const char ***name**

Timer name, used in *esp_timer_dump()* function.

bool **skip_unhandled_events**

Setting to skip unhandled events in light sleep for periodic timers.

Type Definitions

typedef struct esp_timer ***esp_timer_handle_t**

Opaque type representing a single timer handle.

typedef void (***esp_timer_cb_t**)(void *arg)

Timer callback function type.

Param arg pointer to opaque user-specific data

Enumerations

enum **esp_timer_dispatch_t**

Method to dispatch timer callback.

Values:

enumerator **ESP_TIMER_TASK**

Callback is dispatched from esp_timer task.

enumerator **ESP_TIMER_ISR**

Callback is dispatched from interrupt handler.

enumerator **ESP_TIMER_MAX**

Sentinel value for the number of callback dispatch methods.

2.9.19 内部 API 和不稳定的 API

该文档列举了一些 API，这些 API 供内部使用或可能在 ESP-IDF 后续版本中被更改或删除。

API 参考

Header File

- components/esp_rom/include/esp_rom_sys.h
- This header file can be included with:

```
#include "esp_rom_sys.h"
```

Functions

void **esp_rom_software_reset_system**(void)

Software Reset digital core include RTC.

It is not recommended to use this function in esp-idf, use esp_restart() instead.

void **esp_rom_software_reset_cpu**(int cpu_no)

Software Reset cpu core.

It is not recommended to use this function in esp-idf, use esp_restart() instead.

参数 **cpu_no** -- : The CPU to reset, 0 for PRO CPU, 1 for APP CPU.

int **esp_rom_printf**(const char *fmt, ...)

Print formated string to console device.

备注: float and long long data are not supported!

参数

- **fmt** -- Format string
- **...** -- Additional arguments, depending on the format string

返回 int: Total number of characters written on success; A negative number on failure.

void **esp_rom_delay_us**(uint32_t us)

Pauses execution for us microseconds.

参数 **us** -- Number of microseconds to pause

void **esp_rom_install_channel_putc**(int channel, void (*putc)(char c))

esp_rom_printf can print message to different channels simultaneously. This function can help install the low level putc function for esp_rom_printf.

参数

- **channel** -- Channel number (startting from 1)
- **putc** -- Function pointer to the putc implementation. Set NULL can disconnect esp_rom_printf with putc.

void **esp_rom_install_uart_printf**(void)

Install UART1 as the default console channel, equivalent to `esp_rom_install_channel_putc(1, esp_rom_output_putc)`

soc_reset_reason_t **esp_rom_get_reset_reason**(int cpu_no)

Get reset reason of CPU.

参数 **cpu_no** -- CPU number

返回 Reset reason code (see in soc/reset_reasons.h)

void **esp_rom_route_intr_matrix**(int cpu_core, uint32_t periph_intr_id, uint32_t cpu_intr_num)

Route peripheral interrupt sources to CPU's interrupt port by matrix.

Usually there're 4 steps to use an interrupt:

a. Route peripheral interrupt source to CPU. e.g. esp_rom_route_intr_matrix(0, ETS_WIFI_MAC_INTR_SOURCE, ETS_WMAC_INUM)
b. Set interrupt handler for CPU
c. Enable CPU interrupt
d. Enable peripheral interrupt

参数

- **cpu_core** -- The CPU number, which the peripheral interrupt will inform to
- **periph_intr_id** -- The peripheral interrupt source number
- **cpu_intr_num** -- The CPU (external) interrupt number. On targets that use CLIC as their interrupt controller, this number represents the external interrupt number. For example, passing `cpu_intr_num = i` to this function would in fact bind peripheral source to CPU interrupt `CLIC_EXT_INTR_NUM_OFFSET + i`.

uint32_t **esp_rom_get_cpu_ticks_per_us**(void)

Get the real CPU ticks per us.

返回 CPU ticks per us

void **esp_rom_set_cpu_ticks_per_us**(uint32_t ticks_per_us)

Set the real CPU tick rate.

备注: Call this function when CPU frequency is changed, otherwise the `esp_rom_delay_us` can be inaccurate.

参数 **ticks_per_us** -- CPU ticks per us

2.9.20 处理器间调用 (IPC)

备注: IPC 指 **处理器间调用** (Inter-Processor Call)，而不是其他操作系统中所指的 **进程间通信** (Inter-Process Communication)。

概述

由于 ESP32-P4 的双核特性，在某些情况下，某个回调必须在特定的内核上下文中运行。例如：

- 为特定内核的中断源分配 ISR，或释放特定内核的中断源时
- 在特定芯片（如 ESP32）上访问某个内核独有的内存，如 RTC Fast Memory 时
- 读取另一个内核的寄存器或状态时

IPC 功能允许一个特定的内核（下文称 “调用内核”）触发另一个内核（下文称 “目标内核”）的回调函数执行，并允许目标内核在任务上下文或中断上下文中执行回调函数。在不同的上下文中，回调函数的实现限制有所不同。

任务上下文中的 IPC

在应用程序启动时，IPC 功能为每个内核创建一个 IPC 任务，从而在任务上下文中执行回调。当调用内核需要在目标内核中执行回调时，回调将在目标内核的 IPC 任务的上下文中执行。

在任务上下文中使用 IPC 功能时，需考虑以下几点：

- IPC 回调应该尽可能简短。**IPC 回调决不能阻塞或让出**。
- IPC 任务是以尽可能高的优先级创建的（即 `configMAX_PRIORITIES - 1`）。

  - 如果启用了*CONFIG_ESP_IPC_USES_CALLERS_PRIORITY*，执行回调前会降低目标内核的 IPC 任务优先级，使其等于调用内核的优先级。
  - 如果禁用了*CONFIG_ESP_IPC_USES_CALLERS_PRIORITY*，目标内核将始终以尽可能高的优先级执行回调。
- 如果回调较为复杂，用户可能需要通过*CONFIG_ESP_IPC_TASK_STACK_SIZE* 来配置 IPC 任务的堆栈大小。
- IPC 功能受内部互斥锁保护。因此，如果同时收到来自两个或多个调用内核的 IPC 请求，将按照“先到先得”的原则按顺序处理。

API 用法 任务上下文中的 IPC 回调具有以下限制：

- 回调类型必须是`esp_ipc_func_t`。
- 回调 **决不能阻塞或让出**，以免导致目标内核的 IPC 任务阻塞或让出。
- 回调必须避免改变 IPC 任务的任何状态，例如，不能调用 `vTaskPrioritySet(NULL, x)`。

IPC 功能提供了以下 API，用于在目标内核的任务上下文中执行回调。这两个 API 允许调用内核在回调执行完成前处于阻塞，或者在回调开始执行后立即返回。

- `esp_ipc_call()` 会在目标内核上触发一个 IPC 调用。在目标内核的 IPC 任务 **开始**执行回调前，此函数会一直处于阻塞状态。
- `esp_ipc_call_blocking()` 会在目标内核上触发一个 IPC。在目标内核的 IPC 任务 **完成**回调执行前，此函数会一直处于阻塞状态。

中断上下文中的 IPC

在某些情况下，我们需要快速获取另一个内核的状态，如在核心转储、GDB Stub、各种单元测试和绕过硬件错误的过程中。IPC ISR 功能会在每个内核上保留一个高优先级中断供 IPC 使用，从而在高优先级中断上下文中执行回调。当调用内核需要在目标内核上执行回调时，该回调将在目标内核的高优先级中断的上下文中执行。

在高优先级中断上下文中使用 IPC 时，需要考虑以下几点：

- 保留的高优先级中断的优先级取决于*CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL* 选项。

当回调执行时，需考虑以下几点：

- 调用内核会禁用 3 级及以下优先级的中断。
- 虽然保留中断的优先级取决于*CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL*，但是在执行 IPC ISR 回调期间，目标内核会禁用所有的中断。

API 用法 高优先级中断 IPC 回调函数的类型必须是`esp_ipc_isr_func_t`，其限制条件与常规中断处理程序相同。回调函数可以用 C 语言编写。

IPC 功能提供了下列 API，以在高优先级中断的上下文中执行回调：

- `esp_ipc_isr_call()` 能够在目标内核上触发一个 IPC 调用。在目标内核 **开始**执行回调前，此函数将一直处于忙等待。
- `esp_ipc_isr_call_blocking()` 能够在目标内核上触发一个 IPC 调用。在目标内核 **完成**回调执行前，此函数将一直处于忙等待。

请前往 examples/system/ipc/ipc_isr/riscv/main/main.c 查看使用示例。

高优先级中断 IPC API 还提供了以下便利函数，这些函数可以暂停或恢复目标内核的执行。这些 API 利用高优先级中断 IPC，但同时提供了自己的内部回调函数：

- `esp_ipc_isr_stall_other_cpu()`：暂停目标内核。调用内核禁用 3 级及以下级别的中断，而目标内核将在所有中断被禁用的情况下进入忙等待。在调用`esp_ipc_isr_release_other_cpu()` 前，目标内核会保持忙等待。

- *esp_ipc_isr_release_other_cpu()*：恢复目标内核。

API 参考

Header File

- components/esp_system/include/esp_ipc.h
- This header file can be included with:

```
#include "esp_ipc.h"
```

Functions

esp_err_t **esp_ipc_call**(uint32_t cpu_id, *esp_ipc_func_t* func, void *arg)

Execute a callback on a given CPU.

Execute a given callback on a particular CPU. The callback must be of type ”esp_ipc_func_t” and will be invoked in the context of the target CPU's IPC task.

- This function will block the target CPU's IPC task has begun execution of the callback
- If another IPC call is ongoing, this function will block until the ongoing IPC call completes
- The stack size of the IPC task can be configured via the CONFIG_ESP_IPC_TASK_STACK_SIZE option

备注： In single-core mode, returns ESP_ERR_INVALID_ARG for cpu_id 1.

参数

- **cpu_id** -- **[in]** CPU where the given function should be executed (0 or 1)
- **func** -- **[in]** Pointer to a function of type void func(void* arg) to be executed
- **arg** -- **[in]** Arbitrary argument of type void* to be passed into the function

返回

- ESP_ERR_INVALID_ARG if cpu_id is invalid
- ESP_ERR_INVALID_STATE if the FreeRTOS scheduler is not running
- ESP_OK otherwise

esp_err_t **esp_ipc_call_blocking**(uint32_t cpu_id, *esp_ipc_func_t* func, void *arg)

Execute a callback on a given CPU until and block until it completes.

This function is identical to esp_ipc_call() except that this function will block until the execution of the callback completes.

备注： In single-core mode, returns ESP_ERR_INVALID_ARG for cpu_id 1.

参数

- **cpu_id** -- **[in]** CPU where the given function should be executed (0 or 1)
- **func** -- **[in]** Pointer to a function of type void func(void* arg) to be executed
- **arg** -- **[in]** Arbitrary argument of type void* to be passed into the function

返回

- ESP_ERR_INVALID_ARG if cpu_id is invalid
- ESP_ERR_INVALID_STATE if the FreeRTOS scheduler is not running
- ESP_OK otherwise

Type Definitions

typedef void (*`esp_ipc_func_t`)(void *arg)

IPC Callback.

A callback of this type should be provided as an argument when calling esp_ipc_call() or esp_ipc_call_blocking().

Header File

- components/esp_system/include/esp_ipc_isr.h
- This header file can be included with:

```
#include "esp_ipc_isr.h"
```

Functions

void `esp_ipc_isr_call` (*esp_ipc_isr_func_t* func, void *arg)

Execute an ISR callback on the other CPU.

Execute a given callback on the other CPU in the context of a High Priority Interrupt.

- This function will busy-wait in a critical section until the other CPU has started execution of the callback
- The callback must be written:
  - in assembly for XTENSA chips (such as ESP32, ESP32S3). The function is invoked using a CALLX0 instruction and can use only a2, a3, a4 registers. See :doc:`IPC in Interrupt Context </api-reference/system/ipc>` doc for more details.
  - in C or assembly for RISCV chips (such as ESP32P4).

备注: This function is not available in single-core mode.

参数

- **`func`** -- **[in]** Pointer to a function of type void func(void* arg) to be executed
- **`arg`** -- **[in]** Arbitrary argument of type void* to be passed into the function

void `esp_ipc_isr_call_blocking` (*esp_ipc_isr_func_t* func, void *arg)

Execute an ISR callback on the other CPU and busy-wait until it completes.

This function is identical to esp_ipc_isr_call() except that this function will busy-wait until the execution of the callback completes.

备注: This function is not available in single-core mode.

参数

- **`func`** -- **[in]** Pointer to a function of type void func(void* arg) to be executed
- **`arg`** -- **[in]** Arbitrary argument of type void* to be passed into the function

void `esp_ipc_isr_stall_other_cpu` (void)

Stall the other CPU.

This function will stall the other CPU. The other CPU is stalled by busy-waiting in the context of a High Priority Interrupt. The other CPU will not be resumed until esp_ipc_isr_release_other_cpu() is called.

- This function is internally implemented using IPC ISR
- This function is used for DPORT workaround.

- If the stall feature is paused using esp_ipc_isr_stall_pause(), this function will have no effect

备注： This function is not available in single-core mode.

备注： It is the caller's responsibility to avoid deadlocking on spinlocks

void **esp_ipc_isr_release_other_cpu**(void)

Release the other CPU.

This function will release the other CPU that was previously stalled from calling esp_ipc_isr_stall_other_cpu()

- This function is used for DPORT workaround.
- If the stall feature is paused using esp_ipc_isr_stall_pause(), this function will have no effect

备注： This function is not available in single-core mode.

void **esp_ipc_isr_stall_pause**(void)

Puase the CPU stall feature.

This function will pause the CPU stall feature. Once paused, calls to esp_ipc_isr_stall_other_cpu() and esp_ipc_isr_release_other_cpu() will have no effect. If a IPC ISR call is already in progress, this function will busy-wait until the call completes before pausing the CPU stall feature.

void **esp_ipc_isr_stall_abort**(void)

Abort a CPU stall.

This function will abort any stalling routine of the other CPU due to a pervious call to esp_ipc_isr_stall_other_cpu(). This function aborts the stall in a non-recoverable manner, thus should only be called in case of a panic().

- This function is used in panic handling code

void **esp_ipc_isr_stall_resume**(void)

Resume the CPU stall feature.

This function will resume the CPU stall feature that was previously paused by calling esp_ipc_isr_stall_pause(). Once resumed, calls to esp_ipc_isr_stall_other_cpu() and esp_ipc_isr_release_other_cpu() will have effect again.

Macros

esp_ipc_isr_asm_call(func, arg)

Execute an ISR callback on the other CPU See esp_ipc_isr_call().

esp_ipc_isr_asm_call_blocking(func, arg)

Execute an ISR callback on the other CPU and busy-wait until it completes See esp_ipc_isr_call_blocking().

Type Definitions

typedef void (*__esp_ipc_isr_func_t__)(void *arg)

IPC ISR Callback.

The callback must be written:

- in assembly for XTENSA chips (such as ESP32, ESP32S3).
- in C or assembly for RISCV chips (such as ESP32P4).

A callback of this type should be provided as an argument when calling esp_ipc_isr_call() or esp_ipc_isr_call_blocking().

2.9.21 中断分配

概述

ESP32-P4 有两个核，每个核有 32 个外部异步中断。每个中断的优先级别都可独立地通过编程设置。此外，每个核还有 3 个核心本地中断源 (CLINT)。详细信息请参见 **ESP32-P4 技术参考手册** [PDF]。

由于中断源数量多于中断，有时多个驱动程序可以共用一个中断。*esp_intr_alloc()* 抽象隐藏了这些实现细节。

驱动程序可以通过调用*esp_intr_alloc()*，或*esp_intr_alloc_intrstatus()* 为某个外设分配中断。通过向此函数传递 flag，可以指定中断类型、优先级和触发方式。然后，中断分配代码会找到适用的中断，使用中断矩阵将其连接到外设，并为其安装给定的中断处理程序和 ISR。

中断分配器提供两种不同的中断类型：共享中断和非共享中断，这两种中断需要不同处理方式。非共享中断在每次调用*esp_intr_alloc()* 时，都会分配一个单独的中断，该中断仅用于与其相连的外设，只调用一个 ISR。共享中断则可以由多个外设触发，当其中一个外设发出中断信号时，会调用多个 ISR。因此，针对共享中断的 ISR 应检查对应外设的中断状态，以确定是否需要采取任何操作。

非共享中断可由电平或边缘触发。共享中断只能由电平触发，因为使用边缘触发可能会错过中断。

要解释为什么共享中断只能由电平触发，以外设 A 和外设 B 共用一个边缘触发中断为例进行说明：当外设 B 触发中断时，会将其中断信号设置为高电平，产生一个从低到高的边缘，进而锁存 CPU 中断位，并触发 ISR。接着，ISR 开始执行，检查到此时外设 A 没有触发中断，于是继续处理外设 B 的中断信号，最后将外设 B 的中断状态清除。最后，CPU 会在 ISR 返回之前清除中断位锁存器。因此在整个中断处理过程中，如果外设 A 触发了中断，该中断会因 CPU 清除中断位锁存器而丢失。

- 外部中断会始终被分配到执行该分配的内核上。
- 释放外部中断必须在分配该中断的内核上进行。
- 允许从另一个内核禁用和启用外部中断。
- 多个外部中断源可以通过向*esp_intr_alloc()* 发送 `ESP_INTR_FLAG_SHARED` flag 来共享一个中断槽。

须注意从未关联到内核的任务中调用*esp_intr_alloc()* 的情况。在任务切换期间，这些任务可能在不同内核之间进行迁移，因此无法确定中断分配到了哪个 CPU，给释放中断句柄造成困难，也可能引起调试问题。建议使用特定 CoreID 参数的*xTaskCreatePinnedToCore()* 来创建中断分配任务，这对于内部中断源而言是必要的。

多核问题

每个 ESP32-P4 内核都同时提供内部中断和外部中断，内部中断由内核自身触发，外部中断由外设触发。但 ESP-IDF 仅使用 ESP32-P4 上的外部中断。大多数 ESP32-P4 中断源都属于外部中断。

每个内核的各个外部中断槽都与中断矩阵相连。通过中断矩阵可将任何外部中断源连接到任何中断槽，也可将多个外部中断源映射到同一个中断槽。外部中断源在 `soc/interrupts.h` 中定义为 `ETS_*_INTR_SOURCE`。

IRAM-safe 中断处理程序

`ESP_INTR_FLAG_IRAM` flag 注册的中断处理程序始终在 IRAM（并从 DRAM 读取其所有数据）中运行，因此在擦除和写入 flash 时无需禁用。

这对于需要保证最小执行延迟的中断来说非常有用，因为 flash 写入和擦除操作可能很慢（擦除可能需要数十毫秒或数百毫秒才能完成）。

如果中断被频繁调用，可以将中断处理程序保留在 IRAM 中，避免 flash cache 丢失。

有关更多详细信息，请参阅*SPI flash API 相关文档*。

多个处理程序共用一个中断源

如果用 ESP_INTR_FLAG_SHARED flag 分配所有处理程序，可能将多个处理程序分配给同一个源。这些程序会被分配给与源关联的中断，并在源可用时按顺序调用。处理程序可以单独禁用和释放。如果启用了一个或多个处理程序，则会将源关联到中断（启用），否则会取消关联。禁用的处理程序永远不会被调用，但是只要启用了源的任何一个处理程序，这个源仍然能被触发。

关联到非共享中断的源不支持此功能。

尽管支持此功能，使用时也必须 **非常小心**。通常存在两种办法可以阻止中断触发：**禁用源**或 **屏蔽外设中断状态**。ESP-IDF 仅处理源本身的启用和禁用，中断源的状态位和屏蔽位须由用户操作。

状态位须在负责该位的处理程序禁用前屏蔽，也可以在另一个启用的中断中屏蔽和处理该状态位。

备注： 如果不屏蔽状态位而让其处于未处理状态，同时禁用这些状态位的处理程序，就会导致无限次触发中断，引起系统崩溃。

排除中断分配故障

CPU 中断在大多数 Espressif SoC 上都是有限的资源。因此，一个运行的程序有可能耗尽 CPU 中断，例如初始化多个外设驱动程序的情况。这通常导致驱动程序的初始化函数返回 ESP_ERR_NOT_FOUND 错误。

这种情况下，可使用*esp_intr_dump()* 函数打印中断列表及其状态。此函数输出通常如下：

```
CPU 0 interrupt status:
Int  Level  Type   Status
0     1    Level  Reserved
1     1    Level  Reserved
2     1    Level  Used: RTC_CORE
3     1    Level  Used: TG0_LACT_LEVEL
...
```

输出列含义如下：

- Int：CPU 中断输入编号。通常不直接在软件中使用，仅作为参考。
- Level：已分配中断的优先级级别。空闲的中断具有标记 *。
- Type：已分配中断的类型（电平或边缘中断）。空闲的中断具有标记 *。
- **Status：中断的可能状态：**
  - Reserved：中断在硬件层面保留，或由 ESP-IDF 的某些部分保留。不能使用*esp_intr_alloc()* 分配。
  - Used: <source>：中断已分配并连接到单个外设。
  - Shared: <source1> <source2> ...：中断已分配并连接到多个外设。参见本文档多个处理程序共用一个中断源 章节。
  - Free：中断未分配，可以由*esp_intr_alloc()* 使用。

如果已确认应用程序的确用完了中断，可组合采用下列建议解决问题：

- 在多核目标上，尝试通过固定在第二个核的任务来初始化某些外设驱动程序。中断通常分配在运行外设驱动程序初始化函数的同一个内核上，因此，通过在第二个内核上运行初始化函数，就可以使用更多的中断输入。
- 找到可接受更高延迟的中断，并用 ESP_INTR_FLAG_SHARED flag（或与 ESP_INTR_FLAG_LOWMED 进行 OR 运算）分配这些中断。对两个或更多外设使用此 flag 能让它们使用单个中断输入，从而为其他外设节约中断输入。参见多个处理程序共用一个中断源。

- 检查是否有些外设驱动程序不需要一直启用，并按需将其初始化或取消初始化。这样可以减少同时分配的中断数量。

API 参考

Header File

- components/esp_hw_support/include/esp_intr_types.h
- This header file can be included with:

```
#include "esp_intr_types.h"
```

Macros

ESP_INTR_CPU_AFFINITY_TO_CORE_ID(cpu_affinity)

Convert esp_intr_cpu_affinity_t to CPU core ID.

Type Definitions

typedef void (***intr_handler_t**)(void *arg)

Function prototype for interrupt handler function

typedef struct intr_handle_data_t ***intr_handle_t**

Handle to an interrupt handler

Enumerations

enum **esp_intr_cpu_affinity_t**

Interrupt CPU core affinity.

This type specify the CPU core that the peripheral interrupt is connected to.

Values:

enumerator **ESP_INTR_CPU_AFFINITY_AUTO**

Install the peripheral interrupt to ANY CPU core, decided by on which CPU the interrupt allocator is running.

enumerator **ESP_INTR_CPU_AFFINITY_0**

Install the peripheral interrupt to CPU core 0.

enumerator **ESP_INTR_CPU_AFFINITY_1**

Install the peripheral interrupt to CPU core 1.

Header File

- components/esp_hw_support/include/esp_intr_alloc.h
- This header file can be included with:

```
#include "esp_intr_alloc.h"
```

Functions

esp_err_t **esp_intr_mark_shared**(int intno, int cpu, bool is_in_iram)

Mark an interrupt as a shared interrupt.

This will mark a certain interrupt on the specified CPU as an interrupt that can be used to hook shared interrupt handlers to.

参数

- **intno** -- The number of the interrupt (0-31)
- **cpu** -- CPU on which the interrupt should be marked as shared (0 or 1)
- **is_in_iram** -- Shared interrupt is for handlers that reside in IRAM and the int can be left enabled while the flash cache is disabled.

返回 ESP_ERR_INVALID_ARG if cpu or intno is invalid ESP_OK otherwise

esp_err_t **esp_intr_reserve**(int intno, int cpu)

Reserve an interrupt to be used outside of this framework.

This will mark a certain interrupt on the specified CPU as reserved, not to be allocated for any reason.

参数

- **intno** -- The number of the interrupt (0-31)
- **cpu** -- CPU on which the interrupt should be marked as shared (0 or 1)

返回 ESP_ERR_INVALID_ARG if cpu or intno is invalid ESP_OK otherwise

esp_err_t **esp_intr_alloc**(int source, int flags, *intr_handler_t* handler, void *arg, *intr_handle_t* *ret_handle)

Allocate an interrupt with the given parameters.

This finds an interrupt that matches the restrictions as given in the flags parameter, maps the given interrupt source to it and hooks up the given interrupt handler (with optional argument) as well. If needed, it can return a handle for the interrupt as well.

The interrupt will always be allocated on the core that runs this function.

If ESP_INTR_FLAG_IRAM flag is used, and handler address is not in IRAM or RTC_FAST_MEM, then ESP_ERR_INVALID_ARG is returned.

参数

- **source** -- The interrupt source. One of the ETS_*_INTR_SOURCE interrupt mux sources, as defined in soc/soc.h, or one of the internal ETS_INTERNAL_*_INTR_SOURCE sources as defined in this header.
- **flags** -- An ORred mask of the ESP_INTR_FLAG_* defines. These restrict the choice of interrupts that this routine can choose from. If this value is 0, it will default to allocating a non-shared interrupt of level 1, 2 or 3. If this is ESP_INTR_FLAG_SHARED, it will allocate a shared interrupt of level 1. Setting ESP_INTR_FLAG_INTRDISABLED will return from this function with the interrupt disabled.
- **handler** -- The interrupt handler. Must be NULL when an interrupt of level >3 is requested, because these types of interrupts aren't C-callable.
- **arg** -- Optional argument for passed to the interrupt handler
- **ret_handle** -- Pointer to an intr_handle_t to store a handle that can later be used to request details or free the interrupt. Can be NULL if no handle is required.

返回 ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_ERR_NOT_FOUND No free interrupt found with the specified flags ESP_OK otherwise

esp_err_t **esp_intr_alloc_intrstatus**(int source, int flags, uint32_t intrstatusreg, uint32_t intrstatusmask, *intr_handler_t* handler, void *arg, *intr_handle_t* *ret_handle)

Allocate an interrupt with the given parameters.

This essentially does the same as esp_intr_alloc, but allows specifying a register and mask combo. For shared interrupts, the handler is only called if a read from the specified register, ANDed with the mask, returns non-zero. By passing an interrupt status register address and a fitting mask, this can be used to accelerate interrupt handling in the case a shared interrupt is triggered; by checking the interrupt statuses first, the code can decide which ISRs can be skipped

参数

- **source** -- The interrupt source. One of the ETS_*_INTR_SOURCE interrupt mux sources, as defined in soc/soc.h, or one of the internal ETS_INTERNAL_*_INTR_SOURCE sources as defined in this header.
- **flags** -- An ORred mask of the ESP_INTR_FLAG_* defines. These restrict the choice of interrupts that this routine can choose from. If this value is 0, it will default to allocating a non-shared interrupt of level 1, 2 or 3. If this is ESP_INTR_FLAG_SHARED, it will allocate a shared interrupt of level 1. Setting ESP_INTR_FLAG_INTRDISABLED will return from this function with the interrupt disabled.
- **intrstatusreg** -- The address of an interrupt status register
- **intrstatusmask** -- A mask. If a read of address intrstatusreg has any of the bits that are 1 in the mask set, the ISR will be called. If not, it will be skipped.
- **handler** -- The interrupt handler. Must be NULL when an interrupt of level >3 is requested, because these types of interrupts aren't C-callable.
- **arg** -- Optional argument for passed to the interrupt handler
- **ret_handle** -- Pointer to an intr_handle_t to store a handle that can later be used to request details or free the interrupt. Can be NULL if no handle is required.

返回 ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_ERR_NOT_FOUND No free interrupt found with the specified flags ESP_OK otherwise

esp_err_t **esp_intr_free**(*intr_handle_t* handle)

Disable and free an interrupt.

Use an interrupt handle to disable the interrupt and release the resources associated with it. If the current core is not the core that registered this interrupt, this routine will be assigned to the core that allocated this interrupt, blocking and waiting until the resource is successfully released.

备注: When the handler shares its source with other handlers, the interrupt status bits it's responsible for should be managed properly before freeing it. see `esp_intr_disable` for more details. Please do not call this function in `esp_ipc_call_blocking`.

参数 **handle** -- The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

返回 ESP_ERR_INVALID_ARG the handle is NULL ESP_FAIL failed to release this handle ESP_OK otherwise

int **esp_intr_get_cpu**(*intr_handle_t* handle)

Get CPU number an interrupt is tied to.

参数 **handle** -- The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

返回 The core number where the interrupt is allocated

int **esp_intr_get_intno**(*intr_handle_t* handle)

Get the allocated interrupt for a certain handle.

参数 **handle** -- The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

返回 The interrupt number

esp_err_t **esp_intr_disable**(*intr_handle_t* handle)

Disable the interrupt associated with the handle.

备注:

a. For local interrupts (ESP_INTERNAL_* sources), this function has to be called on the CPU the interrupt is allocated on. Other interrupts have no such restriction.

b. When several handlers sharing a same interrupt source, interrupt status bits, which are handled in the handler to be disabled, should be masked before the disabling, or handled in other enabled interrupts properly. Miss of interrupt status handling will cause infinite interrupt calls and finally system crash.

参数 **handle** -- The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus
返回 ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

esp_err_t **esp_intr_enable**(*intr_handle_t* handle)

Enable the interrupt associated with the handle.

备注: For local interrupts (ESP_INTERNAL_* sources), this function has to be called on the CPU the interrupt is allocated on. Other interrupts have no such restriction.

参数 **handle** -- The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus
返回 ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

esp_err_t **esp_intr_set_in_iram**(*intr_handle_t* handle, bool is_in_iram)

Set the "in IRAM" status of the handler.

备注: Does not work on shared interrupts.

参数

- **handle** -- The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus
- **is_in_iram** -- Whether the handler associated with this handle resides in IRAM. Handlers residing in IRAM can be called when cache is disabled.

返回 ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

void **esp_intr_noniram_disable**(void)

Disable interrupts that aren't specifically marked as running from IRAM.

void **esp_intr_noniram_enable**(void)

Re-enable interrupts disabled by esp_intr_noniram_disable.

void **esp_intr_enable_source**(int inum)

enable the interrupt source based on its number

参数 **inum** -- interrupt number from 0 to 31

void **esp_intr_disable_source**(int inum)

disable the interrupt source based on its number

参数 **inum** -- interrupt number from 0 to 31

static inline int **esp_intr_flags_to_level**(int flags)

Get the lowest interrupt level from the flags.

参数 **flags** -- The same flags that pass to `esp_intr_alloc_intrstatus` API

static inline int **esp_intr_level_to_flags**(int level)

Get the interrupt flags from the supplied level (priority)

参数 **level** -- The interrupt priority level

esp_err_t **esp_intr_dump**(FILE *stream)

Dump the status of allocated interrupts.

参数 **stream** -- The stream to dump to, if NULL then stdout is used
返回 ESP_OK on success

bool **esp_intr_ptr_in_isr_region**(void *ptr)

Check if the given pointer is in the safe ISR area. In other words, make sure that the pointer's content is accessible at any time, regardless of the cache status.

参数 **ptr** -- Pointer to check
返回 true if `ptr` points to ISR area, false else

Macros

ESP_INTR_FLAG_LEVEL1

Interrupt allocation flags.

These flags can be used to specify which interrupt qualities the code calling esp_intr_alloc* needs. Accept a Level 1 interrupt vector (lowest priority)

ESP_INTR_FLAG_LEVEL2

Accept a Level 2 interrupt vector.

ESP_INTR_FLAG_LEVEL3

Accept a Level 3 interrupt vector.

ESP_INTR_FLAG_LEVEL4

Accept a Level 4 interrupt vector.

ESP_INTR_FLAG_LEVEL5

Accept a Level 5 interrupt vector.

ESP_INTR_FLAG_LEVEL6

Accept a Level 6 interrupt vector.

ESP_INTR_FLAG_NMI

Accept a Level 7 interrupt vector (highest priority)

ESP_INTR_FLAG_SHARED

Interrupt can be shared between ISRs.

ESP_INTR_FLAG_EDGE

Edge-triggered interrupt.

ESP_INTR_FLAG_IRAM

ISR can be called if cache is disabled.

ESP_INTR_FLAG_INTRDISABLED

Return with this interrupt disabled.

ESP_INTR_FLAG_LOWMED

Low and medium prio interrupts. These can be handled in C.

ESP_INTR_FLAG_HIGH

High level interrupts. Need to be handled in assembly.

ESP_INTR_FLAG_LEVELMASK

Mask for all level flags

ETS_INTERNAL_TIMER0_INTR_SOURCE

Platform timer 0 interrupt source.

The esp_intr_alloc* functions can allocate an int for all ETS_*_INTR_SOURCE interrupt sources that are routed through the interrupt mux. Apart from these sources, each core also has some internal sources that do not pass through the interrupt mux. To allocate an interrupt for these sources, pass these pseudo-sources to the functions.

ETS_INTERNAL_TIMER1_INTR_SOURCE

Platform timer 1 interrupt source.

ETS_INTERNAL_TIMER2_INTR_SOURCE

Platform timer 2 interrupt source.

ETS_INTERNAL_SW0_INTR_SOURCE

Software int source 1.

ETS_INTERNAL_SW1_INTR_SOURCE

Software int source 2.

ETS_INTERNAL_PROFILING_INTR_SOURCE

Int source for profiling.

ETS_INTERNAL_UNUSED_INTR_SOURCE

Interrupt is not assigned to any source.

ETS_INTERNAL_INTR_SOURCE_OFF

Provides SystemView with positive IRQ IDs, otherwise scheduler events are not shown properly

ESP_INTR_ENABLE(inum)

Enable interrupt by interrupt number

ESP_INTR_DISABLE(inum)

Disable interrupt by interrupt number

2.9.22 日志库

概述

日志库提供了三种设置日志级别的方式：

- **编译时**：在 menuconfig 中，使用选项*CONFIG_LOG_DEFAULT_LEVEL* 来设置日志级别。
- 另外，还可以选择在 menuconfig 中使用选项*CONFIG_LOG_MAXIMUM_LEVEL* 设置最高日志级别。这个选项默认被配置为默认级别，但这个选项也可以被配置为更高级别，将更多的可选日志编译到固件中。
- **运行时**：默认启用所有级别低于*CONFIG_LOG_DEFAULT_LEVEL* 的日志。`esp_log_level_set()` 函数可以为各个模块分别设置不同的日志级别，可通过人类可读的 ASCII 零终止字符串标签来识别不同的模块。
- **运行时**：启用*CONFIG_LOG_MASTER_LEVEL* 时，可以使用`esp_log_set_level_master()` 函数设置 主日志级别 (Master logging level)。该选项会为所有已编译的日志添加额外的日志级别检查。注意，使用此选项会增加应用程序大小。如果希望在运行时编译大量可选日志，同时避免在不需要日志输出时查找标签及其级别带来的性能损耗，此功能会非常有用。

以下是不同的日志级别：

- 错误（Error，最低级别）
- 警告 (Warning)
- 普通 (Info)
- 调试 (Debug)
- 冗余（Verbose，最高级别）

备注： 注意，函数 *esp_log_level_set()* 无法将日志级别设置为高于 *CONFIG_LOG_MAXIMUM_LEVEL* 指定的级别。如需在编译时将特定文件的日志级别提高到此最高级别以上，请使用 *LOG_LOCAL_LEVEL* 宏（详细信息见下文）。

如何使用日志库

在使用日志功能的所有 C 文件中，将 TAG 变量定义如下：

```
static const char* TAG = "MyModule";
```

然后使用一个日志宏进行输出，例如：

```
ESP_LOGW(TAG, "Baud rate error %.1f%%. Requested: %d baud, actual: %d baud", error␣
↪* 100, baud_req, baud_real);
```

可使用下列宏来定义不同的日志级别：

- ESP_LOGE - 错误（最低级别）
- ESP_LOGW - 警告
- ESP_LOGI - 普通
- ESP_LOGD - 调试
- ESP_LOGV - 冗余（最高级别）

此外，上述宏还有对应的 ESP_EARLY_LOGx 版本，如 *ESP_EARLY_LOGE*。这些版本的宏必须在堆分配器和系统调用初始化之前，在早期启动代码中显式使用。通常情况下，编译引导加载程序时也可以使用普通的 ESP_LOGx 宏，但其最终实现与 ESP_EARLY_LOGx 宏相同。

上述宏还有对应的 ESP_DRAM_LOGx 版本，如 *ESP_DRAM_LOGE*。在禁用中断或无法访问 flash cache 的情况下需要输出日志时，可以使用这些版本的宏。但是，应尽量避免使用这些宏版本，因为在上述情况下输出日志可能会影响性能。

备注： 在关键部分中断被禁用，因此只能使用 ESP_DRAM_LOGx （首选）或 ESP_EARLY_LOGx 宏。尽管这样可以输出日志，但最好可以调整程序使其不用输出日志。

如需在文件或组件范围内覆盖默认的日志级别，请定义 LOG_LOCAL_LEVEL 宏。

在文件中，该宏应在包含 esp_log.h 文件前进行定义，例如：

```
#define LOG_LOCAL_LEVEL ESP_LOG_VERBOSE
#include "esp_log.h"
```

在组件中，该宏应在组件的 CMakeList 中进行定义：

```
target_compile_definitions(${COMPONENT_LIB} PUBLIC "-DLOG_LOCAL_LEVEL=ESP_LOG_
↪VERBOSE")
```

如需在运行时按模块配置日志输出，请按如下方式调用 *esp_log_level_set()* 函数：

```
esp_log_level_set("*", ESP_LOG_ERROR);          // 将所有组件的日志级别设置为错误␣
↪(ERROR) 级别
esp_log_level_set("wifi", ESP_LOG_WARN);        // 启用来自 WiFi 堆栈的警告 (WARN)␣
↪日志
esp_log_level_set("dhcpc", ESP_LOG_INFO);       // 启用来自 DHCP 客户端的普通 (INFO)␣
↪日志
```

备注： 上文介绍的"DRAM" 和"EARLY" 日志宏变型不支持按照模块设置日志级别。这些宏始终以 "默认" 级别记录日志，且只能在运行时调用 `esp_log_level("*", level)` 对日志级别进行更改。

即使已通过标签名称禁用日志输出，每个条目仍需约 10.9 微秒的处理时间。

主日志级别 要启用主日志级别功能，须启用*CONFIG_LOG_MASTER_LEVEL* 选项。该功能在调用*esp_log_write()* 之前为 ESP_LOGx 宏添加了额外的级别检查。这样就可以设置更高的*CONFIG_LOG_MAXIMUM_LEVEL*，并且不会在正常操作期间对性能造成影响（仅在有指示时）。应用程序可以全局设置主日志级别（*esp_log_set_level_master()*）以强制执行最高日志级别。高于此级别的 ESP_LOGx 宏将直接跳过，不会调用*esp_log_write()* 并进行标签查找。建议只在顶层应用程序中使用此功能，不要在共享组件中使用，因为这将覆盖所有使用该组件的用户的全局日志级别。默认情况下，启动时主日志级别是*CONFIG_LOG_DEFAULT_LEVEL*。

注意，由于此功能为所有 ESP_LOGx 宏添加了额外的检查，会导致应用程序的大小增加。

以下代码片段展示了主日志级别的运行方式。将主日志级别设置为 ESP_LOG_NONE 将在全局范围内禁用所有日志记录。*esp_log_level_set()* 目前不会影响日志记录。但在主日志级别释放后，日志将按照*esp_log_level_set()* 中的设置打印输出。

```
// 在启动时，主日志级别为 CONFIG_LOG_DEFAULT_LEVEL，并等于ESP_LOG_INFO
ESP_LOGI("lib_name", "用于打印的消息");           // 打印普通 (INFO) 级别消息
esp_log_level_set("lib_name", ESP_LOG_WARN);      // 启用 lib_name 的警告 (WARN)␣
↪日志

esp_log_set_level_master(ESP_LOG_NONE);           // 全局禁用所有日志。esp_log_
↪level_set 目前没有生效

ESP_LOGW("lib_name", "用于打印的消息");           // 主日志级别阻止了打印
esp_log_level_set("lib_name", ESP_LOG_INFO);      // 启用 lib_name 的 INFO 日志
ESP_LOGI("lib_name", "用于打印的消息");           // 主日志级别阻止了打印

esp_log_set_level_master(ESP_LOG_INFO);           // 全局启用所有 INFO 日志

ESP_LOGI("lib_name", "用于打印的消息");           // 打印一条 INFO 消息
```

通过 JTAG 将日志记录到主机 默认情况下，日志库使用类似 vprintf 的函数将格式化输出写入专用 UART。通过调用一个简单的 API，即可将所有日志通过 JTAG 输出，将日志输出速度提高数倍。如需了解详情，请参阅记录日志到主机。

线程安全 日志字符串首先被写入内存 buffer，然后发送到 UART 打印。日志调用是线程安全的，即不同线程的日志不会互相冲突。

应用示例

大多数 ESP-IDF 组件和示例都会使用日志库。如需查看有关日志功能的应用示例，请前往 ESP-IDF 的 examples 目录。与日志最相关的示例如下：

- system/ota
- storage/sd_card

- protocols/https_request

API 参考

Header File

- components/log/include/esp_log.h
- This header file can be included with:

```
#include "esp_log.h"
```

Functions

void **esp_log_set_level_master**(*esp_log_level_t* level)

Master log level.

Optional master log level to check against for ESP_LOGx macros before calling esp_log_write. Allows one to set a higher CONFIG_LOG_MAXIMUM_LEVEL but not impose a performance hit during normal operation (only when instructed). An application may set esp_log_set_level_master(level) to globally enforce a maximum log level. ESP_LOGx macros above this level will be skipped immediately, rather than calling esp_log_write and doing a cache hit.

The tradeoff is increased application size.

参数 **level** -- Master log level

esp_log_level_t **esp_log_get_level_master**(void)

Returns master log level.

返回 Master log level

void **esp_log_level_set**(const char *tag, *esp_log_level_t* level)

Set log level for given tag.

If logging for given component has already been enabled, changes previous setting.

备 注: Note that this function can not raise log level above the level set using CONFIG_LOG_MAXIMUM_LEVEL setting in menuconfig. To raise log level above the default one for a given file, define LOG_LOCAL_LEVEL to one of the ESP_LOG_* values, before including esp_log.h in this file.

参数

- **tag** -- Tag of the log entries to enable. Must be a non-NULL zero terminated string. Value "*" resets log level for all tags to the given value.
- **level** -- Selects log level to enable. Only logs at this and lower verbosity levels will be shown.

esp_log_level_t **esp_log_level_get**(const char *tag)

Get log level for a given tag, can be used to avoid expensive log statements.

参数 **tag** -- Tag of the log to query current level. Must be a non-NULL zero terminated string.
返回 The current log level for the given tag

vprintf_like_t **esp_log_set_vprintf**(*vprintf_like_t* func)

Set function used to output log entries.

By default, log output goes to UART0. This function can be used to redirect log output to some other destination, such as file or network. Returns the original log handler, which may be necessary to return output to the previous destination.

备注： Please note that function callback here must be re-entrant as it can be invoked in parallel from multiple thread context.

参数 **func** -- new Function used for output. Must have same signature as vprintf.
返回 func old Function used for output.

uint32_t **esp_log_timestamp**(void)

Function which returns timestamp to be used in log output.

This function is used in expansion of ESP_LOGx macros. In the 2nd stage bootloader, and at early application startup stage this function uses CPU cycle counter as time source. Later when FreeRTOS scheduler start running, it switches to FreeRTOS tick count.

For now, we ignore millisecond counter overflow.

返回 timestamp, in milliseconds

char ***esp_log_system_timestamp**(void)

Function which returns system timestamp to be used in log output.

This function is used in expansion of ESP_LOGx macros to print the system time as "HH:MM:SS.sss". The system time is initialized to 0 on startup, this can be set to the correct time with an SNTP sync, or manually with standard POSIX time functions.

Currently, this will not get used in logging from binary blobs (i.e. Wi-Fi & Bluetooth libraries), these will still print the RTOS tick time.

返回 timestamp, in "HH:MM:SS.sss"

uint32_t **esp_log_early_timestamp**(void)

Function which returns timestamp to be used in log output.

This function uses HW cycle counter and does not depend on OS, so it can be safely used after application crash.

返回 timestamp, in milliseconds

void **esp_log_write**(*esp_log_level_t* level, const char *tag, const char *format, ...)

Write message into the log.

This function is not intended to be used directly. Instead, use one of ESP_LOGE, ESP_LOGW, ESP_LOGI, ESP_LOGD, ESP_LOGV macros.

This function or these macros should not be used from an interrupt.

void **esp_log_writev**(*esp_log_level_t* level, const char *tag, const char *format, va_list args)

Write message into the log, va_list variant.

This function is provided to ease integration toward other logging framework, so that esp_log can be used as a log sink.

参见:

esp_log_write()

Macros

ESP_LOG_BUFFER_HEX_LEVEL(tag, buffer, buff_len, level)

Log a buffer of hex bytes at specified level, separated into 16 bytes each line.

参数

- **tag** -- description tag

- **buffer** -- Pointer to the buffer array
- **buff_len** -- length of buffer in bytes
- **level** -- level of the log

ESP_LOG_BUFFER_CHAR_LEVEL(tag, buffer, buff_len, level)

Log a buffer of characters at specified level, separated into 16 bytes each line. Buffer should contain only printable characters.

参数

- **tag** -- description tag
- **buffer** -- Pointer to the buffer array
- **buff_len** -- length of buffer in bytes
- **level** -- level of the log

ESP_LOG_BUFFER_HEXDUMP(tag, buffer, buff_len, level)

Dump a buffer to the log at specified level.

The dump log shows just like the one below:

```
W (195) log_example: 0x3ffb4280   45 53 50 33 32 20 69 73  20 67 72 65 61 74␣
↪2c 20  |ESP32 is great, |
W (195) log_example: 0x3ffb4290   77 6f 72 6b 69 6e 67 20  61 6c 6f 6e 67 20␣
↪77 69  |working along wi|
W (205) log_example: 0x3ffb42a0   74 68 20 74 68 65 20 49  44 46 2e 00       ␣
↪      |th the IDF..|
```

It is highly recommended to use terminals with over 102 text width.

参数

- **tag** -- description tag
- **buffer** -- Pointer to the buffer array
- **buff_len** -- length of buffer in bytes
- **level** -- level of the log

ESP_LOG_BUFFER_HEX(tag, buffer, buff_len)

Log a buffer of hex bytes at Info level.

参见:

esp_log_buffer_hex_level

参数

- **tag** -- description tag
- **buffer** -- Pointer to the buffer array
- **buff_len** -- length of buffer in bytes

ESP_LOG_BUFFER_CHAR(tag, buffer, buff_len)

Log a buffer of characters at Info level. Buffer should contain only printable characters.

参见:

esp_log_buffer_char_level

参数

- **tag** -- description tag
- **buffer** -- Pointer to the buffer array
- **buff_len** -- length of buffer in bytes

ESP_EARLY_LOGE(tag, format, ...)

macro to output logs in startup code, before heap allocator and syscalls have been initialized. Log at ESP_LOG_ERROR level.

参见:

printf,ESP_LOGE,ESP_DRAM_LOGE In the future, we want to become compatible with clang. Hence, we provide two versions of the following macros which are using variadic arguments. The first one is using the GNU extension ##__VA_ARGS__. The second one is using the C++20 feature **VA_OPT**(,). This allows users to compile their code with standard C++20 enabled instead of the GNU extension. Below C++20, we haven't found any good alternative to using ##__VA_ARGS__.

ESP_EARLY_LOGW(tag, format, ...)

macro to output logs in startup code at ESP_LOG_WARN level.

参见:

ESP_EARLY_LOGE,ESP_LOGE, printf

ESP_EARLY_LOGI(tag, format, ...)

macro to output logs in startup code at ESP_LOG_INFO level.

参见:

ESP_EARLY_LOGE,ESP_LOGE, printf

ESP_EARLY_LOGD(tag, format, ...)

macro to output logs in startup code at ESP_LOG_DEBUG level.

参见:

ESP_EARLY_LOGE,ESP_LOGE, printf

ESP_EARLY_LOGV(tag, format, ...)

macro to output logs in startup code at ESP_LOG_VERBOSE level.

参见:

ESP_EARLY_LOGE,ESP_LOGE, printf

_ESP_LOG_EARLY_ENABLED(log_level)

ESP_LOG_EARLY_IMPL(tag, format, log_level, log_tag_letter, ...)

ESP_LOGE(tag, format, ...)

ESP_LOGW(tag, format, ...)

ESP_LOGI(tag, format, ...)

ESP_LOGD(tag, format, ...)

ESP_LOGV(tag, format, ...)

ESP_LOG_LEVEL(level, tag, format, ...)

runtime macro to output logs at a specified level.

参见:

printf

参数

- **tag** -- tag of the log, which can be used to change the log level by `esp_log_level_set` at runtime.
- **level** -- level of the output log.
- **format** -- format of the output log. See `printf`
- `...` -- variables to be replaced into the log. See `printf`

ESP_LOG_LEVEL_LOCAL(level, tag, format, ...)

runtime macro to output logs at a specified level. Also check the level with `LOG_LOCAL_LEVEL`. If `CONFIG_LOG_MASTER_LEVEL` set, also check first against `esp_log_get_level_master()`.

参见:

`printf`, `ESP_LOG_LEVEL`

ESP_DRAM_LOGE(tag, format, ...)

Macro to output logs when the cache is disabled. Log at `ESP_LOG_ERROR` level.

Similar to

Usage: `ESP_DRAM_LOGE(DRAM_STR("my_tag"), "format", or`ESP_DRAM_LOGE(TAG, "format", ...)', where TAG is a char* that points to a str in the DRAM.

参见:

`ESP_EARLY_LOGE`, the log level cannot be changed per-tag, however esp_log_level_set("*", level) will set the default level which controls these log lines also.

参见:

`esp_rom_printf`,`ESP_LOGE`

备注: Unlike normal logging macros, it's possible to use this macro when interrupts are disabled or inside an ISR.

备注: Placing log strings in DRAM reduces available DRAM, so only use when absolutely essential.

ESP_DRAM_LOGW(tag, format, ...)

macro to output logs when the cache is disabled at `ESP_LOG_WARN` level.

参见:

`ESP_DRAM_LOGW`,`ESP_LOGW`, `esp_rom_printf`

ESP_DRAM_LOGI(tag, format, ...)

macro to output logs when the cache is disabled at `ESP_LOG_INFO` level.

参见:

`ESP_DRAM_LOGI`,`ESP_LOGI`, `esp_rom_printf`

ESP_DRAM_LOGD (tag, format, ...)

macro to output logs when the cache is disabled at `ESP_LOG_DEBUG` level.

参见:

`ESP_DRAM_LOGD`,`ESP_LOGD`, `esp_rom_printf`

ESP_DRAM_LOGV (tag, format, ...)

macro to output logs when the cache is disabled at `ESP_LOG_VERBOSE` level.

参见:

`ESP_DRAM_LOGV`,`ESP_LOGV`, `esp_rom_printf`

Type Definitions

typedef int (***vprintf_like_t**)(const char*, va_list)

Enumerations

enum **esp_log_level_t**

Log level.

Values:

enumerator **ESP_LOG_NONE**

No log output

enumerator **ESP_LOG_ERROR**

Critical errors, software module can not recover on its own

enumerator **ESP_LOG_WARN**

Error conditions from which recovery measures have been taken

enumerator **ESP_LOG_INFO**

Information messages which describe normal flow of events

enumerator **ESP_LOG_DEBUG**

Extra information which is not necessary for normal use (values, pointers, sizes, etc).

enumerator **ESP_LOG_VERBOSE**

Bigger chunks of debugging information, or frequent messages which can potentially flood the output.

2.9.23 杂项系统 API

软件复位

函数*esp_restart()* 用于执行芯片的软件复位。调用此函数时，程序停止执行，两个 CPU 均复位，应用程序由 bootloader 加载并重启。

函数*esp_register_shutdown_handler()* 用于注册复位前会自动调用的例程（复位过程由*esp_restart()* 函数触发），这与 `atexit` POSIX 函数的功能类似。

复位原因

ESP-IDF 应用程序启动或复位的原因有多种。调用*esp_reset_reason()* 函数可获取最近一次复位的原因。复位的所有可能原因，请查看*esp_reset_reason_t* 中的描述。

堆内存

ESP-IDF 中有两个与堆内存相关的函数：

- 函数*esp_get_free_heap_size()* 用于查询当前可用的堆内存大小。
- 函数*esp_get_minimum_free_heap_size()* 用于查询整个过程中可用的最小堆内存大小（例如应用程序生命周期内可用的最小堆内存大小）。

请注意，ESP-IDF 支持功能不同的的多个堆。上文中函数返回的堆内存大小可使用 `malloc` 函数族来进行分配。有关堆内存的更多信息，请参阅堆内存分配。

MAC 地址

以下 API 用于查询和自定义支持的网络接口（如 Wi-Fi、蓝牙、以太网）的 MAC 地址。

要获取特定接口（如 Wi-Fi、蓝牙、以太网）的 MAC 地址，请调用函数*esp_read_mac()*。

在 ESP-IDF 中，各个网络接口的 MAC 地址是根据单个 **基准 MAC 地址 (Base MAC address)** 计算出来的。默认情况下使用乐鑫指定的基准 MAC 地址，该基准地址在产品生产过程中已预烧录至 ESP32-P4 eFuse。

接口	MAC 地址（默认 4 个全局地址）	MAC 地址（2 个全局地址）
Wi-Fi Station	base_mac	base_mac
Wi-Fi SoftAP	base_mac 最后一组字节后加 1	本地 *MAC*（由 Wi-Fi Station MAC 生成）
蓝牙	base_mac 最后一组字节后加 2	base_mac 最后一组字节后加 1
以太网	base_mac 最后一组字节后加 3	本地 *MAC*（由蓝牙 MAC 生成）

备注： 配置选项 配置了乐鑫提供的全局 MAC 地址的数量。

自定义接口 MAC 有时用户可能需要自定义 MAC 地址，这些地址并不由基准 MAC 地址生成。如需设置自定义接口 MAC 地址，请使用*esp_iface_mac_addr_set()* 函数。该函数用于覆盖由基准 MAC 地址设置（或尚未设置）的接口 MAC 地址。一旦设置某个接口 MAC 地址，即使更改基准 MAC 地址，也不会对其产生影响。

自定义基准 MAC 乐鑫已将默认的基准 MAC 地址预烧录至 eFuse BLK1 中。如需设置自定义基准 MAC 地址，请在初始化任一网络接口或调用*esp_read_mac()* 函数前调用*esp_base_mac_addr_set()* 函数。自定义基准 MAC 地址可以存储在任何支持的存储设备中（例如 flash、NVS）。

分配自定义基准 MAC 地址时，应避免 MAC 地址重叠。请根据上面的表格配置选项*CONFIG_ESP32P4_UNIVERSAL_MAC_ADDRESSES*，设置可从自定义基准 MAC 地址生成的有效全局 MAC 地址。

备注： 也可以调用函数*esp_netif_set_mac()*，在网络初始化后设置网络接口使用的特定 MAC。但建议使用此处介绍的自定义基准 MAC 地址的方法，以避免原始 MAC 地址在更改前短暂出现在网络上。

eFuse 中的自定义 MAC 地址 ESP-IDF 提供了*esp_efuse_mac_get_custom()* 函数，从 eFuse 读取自定义 MAC 地址时，调用该函数将从 eFuse BLK3 加载 MAC 地址。用户也可以调用*esp_read_mac()* 函数，此时需使用 ESP_MAC_EFUSE_CUSTOM 参数。*esp_efuse_mac_get_custom()* 函数假定自定义基准 MAC 地址的存储格式如下：

字段	比特数	比特范围
MAC address	48	200:248

备注： eFuse BLK3 在烧写时使用 RS 编码，这意味着必须同时烧写该块中的所有 eFuse 字段。

调用*esp_efuse_mac_get_custom()* 或*esp_read_mac()* 函数获得自定义 eFuse MAC 地址后，请将此 MAC 地址设置为基准 MAC 地址。有以下两种方法：

1. 使用原有 API：调用*esp_base_mac_addr_set()*。
2. 使用新 API：调用*esp_iface_mac_addr_set()*，此时需使用 ESP_MAC_BASE 参数。

本地 MAC 地址和全局 MAC 地址 在 ESP32-P4 中，乐鑫已预烧录足够数量的有效乐鑫全局 MAC 地址，供所有内部接口使用。上文中的表格已经介绍了如何根据基准 MAC 地址计算出具体接口的 MAC 地址。

当使用自定义 MAC 地址时，可能并非所有接口都能被分配到一个全局 MAC 地址。此时，接口会被分配一个本地 MAC 地址。请注意，这些地址仅用于单个本地网络。

本地 MAC 地址和全局 MAC 地址的定义，请参见 此处。

内部调用函数*esp_derive_local_mac()*，可从全局 MAC 地址生成本地 MAC 地址。具体流程如下：

1. 在全局 MAC 地址的第一个字节组中设置 U/L 位（位值为 0x2），创建本地 MAC 地址。
2. 如果该位已存在于全局 MAC 地址中（即现有的“全局”MAC 地址实际上已经是本地 MAC 地址），则本地 MAC 地址的第一个字节组与 0x4 异或。

芯片版本

esp_chip_info() 函数用于填充*esp_chip_info_t* 结构体中的芯片信息，包括芯片版本、CPU 数量和芯片中已启用功能的位掩码。

SDK 版本

调用函数*esp_get_idf_version()* 可返回一个字符串，该字符串包含了用于编译应用程序的 ESP-IDF 版本，与构建系统中通过 IDF_VER 变量所获得的值相同。该版本字符串的格式即 git describe 命令的运行结果。

也有其它的版本宏可用于在构建过程中获取 ESP-IDF 版本，它们可根据 ESP-IDF 版本启用或禁用部分程序。

- *ESP_IDF_VERSION_MAJOR*、*ESP_IDF_VERSION_MINOR* 和*ESP_IDF_VERSION_PATCH* 分别被定义为代表主要版本、次要版本和补丁版本的整数。
- *ESP_IDF_VERSION_VAL* 和*ESP_IDF_VERSION* 可在确认版本时使用：

```
#include "esp_idf_version.h"

#if ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)
    // 启用 ESP-IDF v4.0 中的功能
#endif
```

应用程序版本

应用程序版本存储在*esp_app_desc_t* 结构体中。该结构体位于 DROM 扇区，有一个从二进制文件头部计算的固定偏移值。该结构体位于*esp_image_header_t* 和*esp_image_segment_header_t* 结构体之后。字段 Version 类型为字符串，最大长度为 32 字节。

若需手动设置版本，需要在项目的 `CMakeLists.txt` 文件中设置 `PROJECT_VER` 变量，即在 `CMakeLists.txt` 文件中，在包含 `project.cmake` 之前添加 `set(PROJECT_VER "0.1.0.1")`。

如果设置了*CONFIG_APP_PROJECT_VER_FROM_CONFIG* 选项，则将使用*CONFIG_APP_PROJECT_VER* 的值。否则，如果在项目中未设置 `PROJECT_VER` 变量，则该变量将从 `$(PROJECT_PATH)/version.txt` 文件（若有）中检索，或使用 git 命令 `git describe` 检索。如果两者都不可用，则 `PROJECT_VER` 将被设置为“1”。应用程序可通过调用*esp_app_get_description()* 或*esp_ota_get_partition_description()* 函数来获取应用程序的版本信息。

API 参考

Header File

- components/esp_system/include/esp_system.h
- This header file can be included with:

```
#include "esp_system.h"
```

Functions

esp_err_t **esp_register_shutdown_handler**(*shutdown_handler_t* handle)

Register shutdown handler.

This function allows you to register a handler that gets invoked before the application is restarted using esp_restart function.

参数 **handle** -- function to execute on restart

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if the handler has already been registered
- ESP_ERR_NO_MEM if no more shutdown handler slots are available

esp_err_t **esp_unregister_shutdown_handler**(*shutdown_handler_t* handle)

Unregister shutdown handler.

This function allows you to unregister a handler which was previously registered using esp_register_shutdown_handler function.

- ESP_OK on success
- ESP_ERR_INVALID_STATE if the given handler hasn't been registered before

void **esp_restart**(void)

Restart PRO and APP CPUs.

This function can be called both from PRO and APP CPUs. After successful restart, CPU reset reason will be SW_CPU_RESET. Peripherals (except for Wi-Fi, BT, UART0, SPI1, and legacy timers) are not reset. This function does not return.

esp_reset_reason_t **esp_reset_reason**(void)

Get reason of last reset.

返回 See description of esp_reset_reason_t for explanation of each value.

uint32_t **esp_get_free_heap_size**(void)

Get the size of available heap.

备注: Note that the returned value may be larger than the maximum contiguous block which can be allocated.

返回 Available heap size, in bytes.

uint32_t **esp_get_free_internal_heap_size**(void)

Get the size of available internal heap.

备注: Note that the returned value may be larger than the maximum contiguous block which can be allocated.

返回 Available internal heap size, in bytes.

uint32_t **esp_get_minimum_free_heap_size**(void)

Get the minimum heap that has ever been available.

返回 Minimum free heap ever available

void **esp_system_abort**(const char *details)

Trigger a software abort.

参数 **details** -- Details that will be displayed during panic handling.

Type Definitions

typedef void (***shutdown_handler_t**)(void)

Shutdown handler type

Enumerations

enum **esp_reset_reason_t**

Reset reasons.

Values:

enumerator **ESP_RST_UNKNOWN**

Reset reason can not be determined.

enumerator **ESP_RST_POWERON**

Reset due to power-on event.

enumerator **ESP_RST_EXT**

Reset by external pin (not applicable for ESP32)

enumerator **ESP_RST_SW**

Software reset via esp_restart.

enumerator **ESP_RST_PANIC**
Software reset due to exception/panic.

enumerator **ESP_RST_INT_WDT**
Reset (software or hardware) due to interrupt watchdog.

enumerator **ESP_RST_TASK_WDT**
Reset due to task watchdog.

enumerator **ESP_RST_WDT**
Reset due to other watchdogs.

enumerator **ESP_RST_DEEPSLEEP**
Reset after exiting deep sleep mode.

enumerator **ESP_RST_BROWNOUT**
Brownout reset (software or hardware)

enumerator **ESP_RST_SDIO**
Reset over SDIO.

enumerator **ESP_RST_USB**
Reset by USB peripheral.

enumerator **ESP_RST_JTAG**
Reset by JTAG.

enumerator **ESP_RST_EFUSE**
Reset due to efuse error.

enumerator **ESP_RST_PWR_GLITCH**
Reset due to power glitch detected.

enumerator **ESP_RST_CPU_LOCKUP**
Reset due to CPU lock up (double exception)

Header File

- components/esp_common/include/esp_idf_version.h
- This header file can be included with:

```
#include "esp_idf_version.h"
```

Functions

const char *__esp_get_idf_version__(void)
Return full IDF version string, same as 'git describe' output.

备注: If you are printing the ESP-IDF version in a log file or other information, this function provides more information than using the numerical version macros. For example, numerical version macros don't differentiate between development, pre-release and release versions, but the output of this function does.

返回 constant string from IDF_VER

Macros

ESP_IDF_VERSION_MAJOR

Major version number (X.x.x)

ESP_IDF_VERSION_MINOR

Minor version number (x.X.x)

ESP_IDF_VERSION_PATCH

Patch version number (x.x.X)

ESP_IDF_VERSION_VAL(major, minor, patch)

Macro to convert IDF version number into an integer

To be used in comparisons, such as ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)

ESP_IDF_VERSION

Current IDF version, as an integer

To be used in comparisons, such as ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)

Header File

- components/esp_hw_support/include/esp_mac.h
- This header file can be included with:

```
#include "esp_mac.h"
```

Functions

esp_err_t **esp_base_mac_addr_set**(const uint8_t *mac)

Set base MAC address with the MAC address which is stored in BLK3 of EFUSE or external storage e.g. flash and EEPROM.

Base MAC address is used to generate the MAC addresses used by network interfaces.

If using a custom base MAC address, call this API before initializing any network interfaces. Refer to the ESP-IDF Programming Guide for details about how the Base MAC is used.

备注: Base MAC must be a unicast MAC (least significant bit of first byte must be zero).

备注: If not using a valid OUI, set the "locally administered" bit (bit value 0x02 in the first byte) to avoid collisions.

参数 **mac** -- base MAC address, length: 6 bytes. length: 6 bytes for MAC-48
返回 ESP_OK on success ESP_ERR_INVALID_ARG If mac is NULL or is not a unicast MAC

esp_err_t **esp_base_mac_addr_get**(uint8_t *mac)

Return base MAC address which is set using esp_base_mac_addr_set.

备注: If no custom Base MAC has been set, this returns the pre-programmed Espressif base MAC address.

参数 **mac** -- base MAC address, length: 6 bytes. length: 6 bytes for MAC-48
返回 ESP_OK on success ESP_ERR_INVALID_ARG mac is NULL ESP_ERR_INVALID_MAC base MAC address has not been set

esp_err_t **esp_efuse_mac_get_custom**(uint8_t *mac)

Return base MAC address which was previously written to BLK3 of EFUSE.

Base MAC address is used to generate the MAC addresses used by the networking interfaces. This API returns the custom base MAC address which was previously written to EFUSE BLK3 in a specified format.

Writing this EFUSE allows setting of a different (non-Espressif) base MAC address. It is also possible to store a custom base MAC address elsewhere, see esp_base_mac_addr_set() for details.

备注: This function is currently only supported on ESP32.

参数 **mac** -- base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4, if CONFIG_SOC_IEEE802154_SUPPORTED=y)
返回 ESP_OK on success ESP_ERR_INVALID_ARG mac is NULL ESP_ERR_INVALID_MAC CUSTOM_MAC address has not been set, all zeros (for esp32-xx) ESP_ERR_INVALID_VERSION An invalid MAC version field was read from BLK3 of EFUSE (for esp32) ESP_ERR_INVALID_CRC An invalid MAC CRC was read from BLK3 of EFUSE (for esp32)

esp_err_t **esp_efuse_mac_get_default**(uint8_t *mac)

Return base MAC address which is factory-programmed by Espressif in EFUSE.

参数 **mac** -- base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4, if CONFIG_SOC_IEEE802154_SUPPORTED=y)
返回 ESP_OK on success ESP_ERR_INVALID_ARG mac is NULL

esp_err_t **esp_read_mac**(uint8_t *mac, *esp_mac_type_t* type)

Read base MAC address and set MAC address of the interface.

This function first get base MAC address using esp_base_mac_addr_get(). Then calculates the MAC address of the specific interface requested, refer to ESP-IDF Programming Guide for the algorithm.

The MAC address set by the esp_iface_mac_addr_set() function will not depend on the base MAC address.

参数

- **mac** -- base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4, if CONFIG_SOC_IEEE802154_SUPPORTED=y)
- **type** -- Type of MAC address to return

返回 ESP_OK on success

esp_err_t **esp_derive_local_mac**(uint8_t *local_mac, const uint8_t *universal_mac)

Derive local MAC address from universal MAC address.

This function copies a universal MAC address and then sets the ”locally

administered” bit (bit 0x2) in the first octet, creating a locally administered MAC address.

If the universal MAC address argument is already a locally administered MAC address, then the first octet is XORed with 0x4 in order to create a different locally administered MAC address.

参数

- **local_mac** -- base MAC address, length: 6 bytes. length: 6 bytes for MAC-48
- **universal_mac** -- Source universal MAC address, length: 6 bytes.

返回 ESP_OK on success

esp_err_t **esp_iface_mac_addr_set**(const uint8_t *mac, *esp_mac_type_t* type)

Set custom MAC address of the interface. This function allows you to overwrite the MAC addresses of the interfaces set by the base MAC address.

参数

- **mac** -- MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for ESP_MAC_IEEE802154 type, if CONFIG_SOC_IEEE802154_SUPPORTED=y)
- **type** -- Type of MAC address

返回 ESP_OK on success

size_t **esp_mac_addr_len_get**(*esp_mac_type_t* type)

Return the size of the MAC type in bytes.

If CONFIG_SOC_IEEE802154_SUPPORTED is set then for these types:

- ESP_MAC_IEEE802154 is 8 bytes.
- ESP_MAC_BASE, ESP_MAC_EFUSE_FACTORY and ESP_MAC_EFUSE_CUSTOM the MAC size is 6 bytes.
- ESP_MAC_EFUSE_EXT is 2 bytes. If CONFIG_SOC_IEEE802154_SUPPORTED is not set then for all types it returns 6 bytes.

参数 **type** -- Type of MAC address

返回 0 MAC type not found (not supported) 6 bytes for MAC-48. 8 bytes for EUI-64.

Macros

MAC2STR(a)

MACSTR

Enumerations

enum **esp_mac_type_t**

Values:

enumerator **ESP_MAC_WIFI_STA**

MAC for WiFi Station (6 bytes)

enumerator **ESP_MAC_WIFI_SOFTAP**

MAC for WiFi Soft-AP (6 bytes)

enumerator **ESP_MAC_BT**

MAC for Bluetooth (6 bytes)

enumerator **ESP_MAC_ETH**

MAC for Ethernet (6 bytes)

enumerator **ESP_MAC_IEEE802154**

if CONFIG_SOC_IEEE802154_SUPPORTED=y, MAC for IEEE802154 (8 bytes)

enumerator **ESP_MAC_BASE**

Base MAC for that used for other MAC types (6 bytes)

enumerator **ESP_MAC_EFUSE_FACTORY**

MAC_FACTORY eFuse which was burned by Espressif in production (6 bytes)

enumerator **ESP_MAC_EFUSE_CUSTOM**

MAC_CUSTOM eFuse which was can be burned by customer (6 bytes)

enumerator **ESP_MAC_EFUSE_EXT**

if CONFIG_SOC_IEEE802154_SUPPORTED=y, MAC_EXT eFuse which is used as an extender for IEEE802154 MAC (2 bytes)

Header File

- components/esp_hw_support/include/esp_chip_info.h
- This header file can be included with:

```
#include "esp_chip_info.h"
```

Functions

void **esp_chip_info**(*esp_chip_info_t* *out_info)

Fill an *esp_chip_info_t* structure with information about the chip.

参数 out_info -- **[out]** structure to be filled

Structures

struct **esp_chip_info_t**

The structure represents information about the chip.

Public Members

esp_chip_model_t **model**

chip model, one of esp_chip_model_t

uint32_t **features**

bit mask of CHIP_FEATURE_x feature flags

uint16_t **revision**

chip revision number (in format MXX; where M - wafer major version, XX - wafer minor version)

uint8_t **cores**

number of CPU cores

Macros

CHIP_FEATURE_EMB_FLASH

Chip has embedded flash memory.

CHIP_FEATURE_WIFI_BGN

Chip has 2.4GHz WiFi.

CHIP_FEATURE_BLE

Chip has Bluetooth LE.

CHIP_FEATURE_BT

Chip has Bluetooth Classic.

CHIP_FEATURE_IEEE802154

Chip has IEEE 802.15.4.

CHIP_FEATURE_EMB_PSRAM

Chip has embedded psram.

Enumerations

enum **esp_chip_model_t**

Chip models.

Values:

enumerator **CHIP_ESP32**

ESP32.

enumerator **CHIP_ESP32S2**

ESP32-S2.

enumerator **CHIP_ESP32S3**

ESP32-S3.

enumerator **CHIP_ESP32C3**

ESP32-C3.

enumerator **CHIP_ESP32C2**

ESP32-C2.

enumerator **CHIP_ESP32C6**

ESP32-C6.

enumerator **CHIP_ESP32H2**

ESP32-H2.

enumerator **CHIP_ESP32P4**

ESP32-P4.

enumerator **CHIP_ESP32C61**

ESP32-C61.

enumerator **CHIP_POSIX_LINUX**

The code is running on POSIX/Linux simulator.

Header File

- components/esp_hw_support/include/esp_cpu.h
- This header file can be included with:

```
#include "esp_cpu.h"
```

Functions

void **esp_cpu_stall**(int core_id)

Stall a CPU core.

参数 **core_id** -- The core's ID

void **esp_cpu_unstall**(int core_id)

Resume a previously stalled CPU core.

参数 **core_id** -- The core's ID

void **esp_cpu_reset**(int core_id)

Reset a CPU core.

参数 **core_id** -- The core's ID

void **esp_cpu_wait_for_intr**(void)

Wait for Interrupt.

This function causes the current CPU core to execute its Wait For Interrupt (WFI or equivalent) instruction. After executing this function, the CPU core will stop execution until an interrupt occurs.

int **esp_cpu_get_core_id**(void)

Get the current core's ID.

This function will return the ID of the current CPU (i.e., the CPU that calls this function).

返回 The current core's ID [0..SOC_CPU_CORES_NUM - 1]

void ***esp_cpu_get_sp**(void)

Read the current stack pointer address.

返回 Stack pointer address

esp_cpu_cycle_count_t **esp_cpu_get_cycle_count**(void)

Get the current CPU core's cycle count.

Each CPU core maintains an internal counter (i.e., cycle count) that increments every CPU clock cycle.

返回 Current CPU's cycle count, 0 if not supported.

void **esp_cpu_set_cycle_count**(*esp_cpu_cycle_count_t* cycle_count)

Set the current CPU core's cycle count.

Set the given value into the internal counter that increments every CPU clock cycle.

参数 **cycle_count** -- CPU cycle count

void ***esp_cpu_pc_to_addr**(uint32_t pc)

Convert a program counter (PC) value to address.

If the architecture does not store the true virtual address in the CPU's PC or return addresses, this function will convert the PC value to a virtual address. Otherwise, the PC is just returned

参数 **pc** -- PC value
返回 Virtual address

void **esp_cpu_intr_get_desc**(int core_id, int intr_num, *esp_cpu_intr_desc_t* *intr_desc_ret)

Get a CPU interrupt's descriptor.

Each CPU interrupt has a descriptor describing the interrupt's capabilities and restrictions. This function gets the descriptor of a particular interrupt on a particular CPU.

参数

- **core_id** -- **[in]** The core's ID
- **intr_num** -- **[in]** Interrupt number
- **intr_desc_ret** -- **[out]** The interrupt's descriptor

void **esp_cpu_intr_set_ivt_addr**(const void *ivt_addr)

Set the base address of the current CPU's Interrupt Vector Table (IVT)

参数 **ivt_addr** -- Interrupt Vector Table's base address

void **esp_cpu_intr_set_mtvt_addr**(const void *mtvt_addr)

Set the base address of the current CPU's Interrupt Vector Table (MTVT)

备注: The MTVT table is only applicable when CLIC is supported

参数 **mtvt_addr** -- Interrupt Vector Table's base address

void **esp_cpu_intr_set_type**(int intr_num, *esp_cpu_intr_type_t* intr_type)

Set the interrupt type of a particular interrupt.

Set the interrupt type (Level or Edge) of a particular interrupt on the current CPU.

参数

- **intr_num** -- Interrupt number (from 0 to 31)
- **intr_type** -- The interrupt's type

esp_cpu_intr_type_t **esp_cpu_intr_get_type**(int intr_num)

Get the current configured type of a particular interrupt.

Get the currently configured type (i.e., level or edge) of a particular interrupt on the current CPU.

参数 **intr_num** -- Interrupt number (from 0 to 31)
返回 Interrupt type

void **esp_cpu_intr_set_priority**(int intr_num, int intr_priority)

Set the priority of a particular interrupt.

Set the priority of a particular interrupt on the current CPU.

参数

- **intr_num** -- Interrupt number (from 0 to 31)
- **intr_priority** -- The interrupt's priority

int **esp_cpu_intr_get_priority**(int intr_num)

Get the current configured priority of a particular interrupt.

Get the currently configured priority of a particular interrupt on the current CPU.

参数 **intr_num** -- Interrupt number (from 0 to 31)
返回 Interrupt's priority

bool **esp_cpu_intr_has_handler**(int intr_num)

Check if a particular interrupt already has a handler function.

Check if a particular interrupt on the current CPU already has a handler function assigned.

备注: This function simply checks if the IVT of the current CPU already has a handler assigned.

参数 **intr_num** -- Interrupt number (from 0 to 31)
返回 True if the interrupt has a handler function, false otherwise.

void **esp_cpu_intr_set_handler**(int intr_num, *esp_cpu_intr_handler_t* handler, void *handler_arg)

Set the handler function of a particular interrupt.

Assign a handler function (i.e., ISR) to a particular interrupt on the current CPU.

备注: This function simply sets the handler function (in the IVT) and does not actually enable the interrupt.

参数
- **intr_num** -- Interrupt number (from 0 to 31)
- **handler** -- Handler function
- **handler_arg** -- Argument passed to the handler function

void ***esp_cpu_intr_get_handler_arg**(int intr_num)

Get a handler function's argument of.

Get the argument of a previously assigned handler function on the current CPU.

参数 **intr_num** -- Interrupt number (from 0 to 31)
返回 The the argument passed to the handler function

void **esp_cpu_intr_enable**(uint32_t intr_mask)

Enable particular interrupts on the current CPU.

参数 **intr_mask** -- Bit mask of the interrupts to enable

void **esp_cpu_intr_disable**(uint32_t intr_mask)

Disable particular interrupts on the current CPU.

参数 **intr_mask** -- Bit mask of the interrupts to disable

uint32_t **esp_cpu_intr_get_enabled_mask**(void)

Get the enabled interrupts on the current CPU.

返回 Bit mask of the enabled interrupts

void **esp_cpu_intr_edge_ack**(int intr_num)

Acknowledge an edge interrupt.

参数 **intr_num** -- Interrupt number (from 0 to 31)

void **esp_cpu_configure_region_protection**(void)

Configure the CPU to disable access to invalid memory regions.

esp_err_t **esp_cpu_set_breakpoint**(int bp_num, const void *bp_addr)

Set and enable a hardware breakpoint on the current CPU.

备注: This function is meant to be called by the panic handler to set a breakpoint for an attached debugger during a panic.

备注: Overwrites previously set breakpoint with same breakpoint number.

参数
- **bp_num** -- Hardware breakpoint number [0..SOC_CPU_BREAKPOINTS_NUM - 1]
- **bp_addr** -- Address to set a breakpoint on

返回 ESP_OK if breakpoint is set. Failure otherwise

esp_err_t **esp_cpu_clear_breakpoint** (int bp_num)

Clear a hardware breakpoint on the current CPU.

备注: Clears a breakpoint regardless of whether it was previously set

参数 **bp_num** -- Hardware breakpoint number [0..SOC_CPU_BREAKPOINTS_NUM - 1]
返回 ESP_OK if breakpoint is cleared. Failure otherwise

esp_err_t **esp_cpu_set_watchpoint** (int wp_num, const void *wp_addr, size_t size, *esp_cpu_watchpoint_trigger_t* trigger)

Set and enable a hardware watchpoint on the current CPU.

Set and enable a hardware watchpoint on the current CPU, specifying the memory range and trigger operation. Watchpoints will break/panic the CPU when the CPU accesses (according to the trigger type) on a certain memory range.

备注: Overwrites previously set watchpoint with same watchpoint number. On RISC-V chips, this API uses method0(Exact matching) and method1(NAPOT matching) according to the riscv-debug-spec-0.13 specification for address matching. If the watch region size is 1byte, it uses exact matching (method 0). If the watch region size is larger than 1byte, it uses NAPOT matching (method 1). This mode requires the watching region start address to be aligned to the watching region size.

参数

- **wp_num** -- Hardware watchpoint number [0..SOC_CPU_WATCHPOINTS_NUM - 1]
- **wp_addr** -- Watchpoint's base address, must be naturally aligned to the size of the region
- **size** -- Size of the region to watch. Must be one of 2^n and in the range of [1 ... SOC_CPU_WATCHPOINT_MAX_REGION_SIZE]
- **trigger** -- Trigger type

返回 ESP_ERR_INVALID_ARG on invalid arg, ESP_OK otherwise

esp_err_t **esp_cpu_clear_watchpoint** (int wp_num)

Clear a hardware watchpoint on the current CPU.

备注: Clears a watchpoint regardless of whether it was previously set

参数 **wp_num** -- Hardware watchpoint number [0..SOC_CPU_WATCHPOINTS_NUM - 1]
返回 ESP_OK if watchpoint was cleared. Failure otherwise.

bool **esp_cpu_dbgr_is_attached** (void)

Check if the current CPU has a debugger attached.

返回 True if debugger is attached, false otherwise

void **esp_cpu_dbgr_break** (void)

Trigger a call to the current CPU's attached debugger.

intptr_t **esp_cpu_get_call_addr** (intptr_t return_address)

Given the return address, calculate the address of the preceding call instruction This is typically used to answer the question "where was the function called from?".

参数 **return_address** -- The value of the return address register. Typically set to the value of __builtin_return_address(0).
返回 Address of the call instruction preceding the return address.

bool **esp_cpu_compare_and_set**(volatile uint32_t *addr, uint32_t compare_value, uint32_t new_value)

Atomic compare-and-set operation.

参数

- **addr** -- Address of atomic variable
- **compare_value** -- Value to compare the atomic variable to
- **new_value** -- New value to set the atomic variable to

返回 Whether the atomic variable was set or not

void **esp_cpu_branch_prediction_enable**(void)

Enable branch prediction.

void **esp_cpu_branch_prediction_disable**(void)

Disable branch prediction.

Structures

struct **esp_cpu_intr_desc_t**

CPU interrupt descriptor.

Each particular CPU interrupt has an associated descriptor describing that particular interrupt's characteristics. Call esp_cpu_intr_get_desc() to get the descriptors of a particular interrupt.

Public Members

int **priority**

Priority of the interrupt if it has a fixed priority, (-1) if the priority is configurable.

esp_cpu_intr_type_t **type**

Whether the interrupt is an edge or level type interrupt, ESP_CPU_INTR_TYPE_NA if the type is configurable.

uint32_t **flags**

Flags indicating extra details.

Macros

ESP_CPU_INTR_DESC_FLAG_SPECIAL

Interrupt descriptor flags of *esp_cpu_intr_desc_t*.

The interrupt is a special interrupt (e.g., a CPU timer interrupt)

ESP_CPU_INTR_DESC_FLAG_RESVD

The interrupt is reserved for internal use

Type Definitions

typedef uint32_t **esp_cpu_cycle_count_t**

CPU cycle count type.

This data type represents the CPU's clock cycle count

typedef void (***esp_cpu_intr_handler_t**)(void *arg)

CPU interrupt handler type.

Enumerations

enum **esp_cpu_intr_type_t**

CPU interrupt type.

Values:

enumerator **ESP_CPU_INTR_TYPE_LEVEL**

enumerator **ESP_CPU_INTR_TYPE_EDGE**

enumerator **ESP_CPU_INTR_TYPE_NA**

enum **esp_cpu_watchpoint_trigger_t**

CPU watchpoint trigger type.

Values:

enumerator **ESP_CPU_WATCHPOINT_LOAD**

enumerator **ESP_CPU_WATCHPOINT_STORE**

enumerator **ESP_CPU_WATCHPOINT_ACCESS**

Header File

- components/esp_app_format/include/esp_app_desc.h
- This header file can be included with:

```
#include "esp_app_desc.h"
```

- This header file is a part of the API provided by the `esp_app_format` component. To declare that your component depends on `esp_app_format`, add the following to your CMakeLists.txt:

```
REQUIRES esp_app_format
```

or

```
PRIV_REQUIRES esp_app_format
```

Functions

const *esp_app_desc_t* ***esp_app_get_description**(void)

Return esp_app_desc structure. This structure includes app version.

Return description for running app.

返回 Pointer to esp_app_desc structure.

int **esp_app_get_elf_sha256**(char *dst, size_t size)

Fill the provided buffer with SHA256 of the ELF file, formatted as hexadecimal, null-terminated. If the buffer size is not sufficient to fit the entire SHA256 in hex plus a null terminator, the largest possible number of bytes will be written followed by a null.

参数

- **dst** -- Destination buffer
- **size** -- Size of the buffer

返回 Number of bytes written to dst (including null terminator)

char *esp_app_get_elf_sha256_str(void)

Return SHA256 of the ELF file which is already formatted as hexadecimal, null-terminated included. Can be used in panic handler or core dump during when cache is disabled. The length is defined by CONFIG_APP_RETRIEVE_LEN_ELF_SHA option.

返回 Hexadecimal SHA256 string

Structures

struct esp_app_desc_t

Description about application.

Public Members

uint32_t magic_word

Magic word ESP_APP_DESC_MAGIC_WORD

uint32_t secure_version

Secure version

uint32_t reserv1[2]

reserv1

char version[32]

Application version

char project_name[32]

Project name

char time[16]

Compile time

char date[16]

Compile date

char idf_ver[32]

Version IDF

uint8_t app_elf_sha256[32]

sha256 of elf file

uint16_t min_efuse_blk_rev_full

Minimal eFuse block revision supported by image, in format: major * 100 + minor

uint16_t max_efuse_blk_rev_full

Maximal eFuse block revision supported by image, in format: major * 100 + minor

uint32_t reserv2[19]

reserv2

Macros

ESP_APP_DESC_MAGIC_WORD

The magic word for the esp_app_desc structure that is in DROM.

2.9.24 空中升级 (OTA)

OTA 流程概览

OTA 升级机制可以让设备在固件正常运行时根据接收数据（如通过 Wi-Fi、蓝牙或以太网）进行自我更新。

要运行 OTA 机制，需配置设备的分区表，该分区表至少包括两个 OTA 应用程序分区（即 `ota_0` 和 `ota_1`）和一个 OTA 数据分区。

OTA 功能启动后，向当前未用于启动的 OTA 应用分区写入新的应用固件镜像。镜像验证后，OTA 数据分区更新，指定在下一次启动时使用该镜像。

OTA 数据分区

所有使用 OTA 功能项目，其分区表必须包含一个 OTA 数据分区（类型为 `data`，子类型为 `ota`）。

工厂启动设置下，OTA 数据分区中应没有数据（所有字节擦写成 0xFF）。如果分区表中有工厂应用程序，ESP-IDF 软件引导加载程序会启动工厂应用程序。如果分区表中没有工厂应用程序，则启动第一个可用的 OTA 分区（通常是 `ota_0`）。

第一次 OTA 升级后，OTA 数据分区更新，指定下一次启动哪个 OTA 应用程序分区。

OTA 数据分区的容量是 2 个 flash 扇区的大小（0x2000 字节），防止写入时电源故障引发问题。两个扇区单独擦除、写入匹配数据，若存在不一致，则用计数器字段判定哪个扇区为最新数据。

应用程序回滚

应用程序回滚的主要目的是确保设备在更新后正常工作。如果新版应用程序出现严重错误，该功能可使设备回滚到之前正常运行的应用版本。在使能回滚并且 OTA 升级应用程序至新版本后，可能出现的结果如下：

- 应用程序运行正常，*esp_ota_mark_app_valid_cancel_rollback()* 将正在运行的应用程序状态标记为 `ESP_OTA_IMG_VALID`，启动此应用程序无限制。
- 应用程序出现严重错误，无法继续工作，必须回滚到此前的版本，*esp_ota_mark_app_invalid_rollback_and_reboot()* 将正在运行的版本标记为 `ESP_OTA_IMG_INVALID` 然后复位。引导加载程序不会选取此版本，而是启动此前正常运行的版本。
- 如果*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 使能，则无需调用函数便可复位，回滚至之前的应用版本。

可使用以下代码检测 OTA 更新后应用程序的首次启动。首次启动时，应用程序会检查其状态并执行检测。如果检测成功，应用程序调用*esp_ota_mark_app_valid_cancel_rollback()* 函数，确认应用运行成功。如果检测失败，应用程序调用*esp_ota_mark_app_invalid_rollback_and_reboot()* 函数，回滚至之前的应用版本。

如果应用程序由于中止、重启或掉电无法启动或运行上述代码，引导加载程序在下一次启动尝试中会将该应用程序的状态标记为 `ESP_OTA_IMG_INVALID`，并回滚至之前的应用版本。

```
const esp_partition_t *running = esp_ota_get_running_partition();
esp_ota_img_states_t ota_state;
if (esp_ota_get_state_partition(running, &ota_state) == ESP_OK) {
    if (ota_state == ESP_OTA_IMG_PENDING_VERIFY) {
        // run diagnostic function ...
        bool diagnostic_is_ok = diagnostic();
        if (diagnostic_is_ok) {
            ESP_LOGI(TAG, "Diagnostics completed successfully! Continuing␣
↪execution ...");
            esp_ota_mark_app_valid_cancel_rollback();
        } else {
            ESP_LOGE(TAG, "Diagnostics failed! Start rollback to the previous␣
↪version ...");
            esp_ota_mark_app_invalid_rollback_and_reboot();
        }
    }
}
```

请查看 system/ota/native_ota_example 获取包含上述代码片段的完整示例。

备注： 应用程序的状态不是写到程序的二进制镜像，而是写到 otadata 分区。该分区有一个 ota_seq 计数器，该计数器是 OTA 应用分区的指针，指向下次启动时选取应用所在的分区 (ota_0, ota_1, ...)。

应用程序 OTA 状态 状态控制了选取启动应用程序的过程：

状态	引导加载程序选取启动应用程序的限制
ESP_OTA_IMG_VALID	没有限制，可以选取。
ESP_OTA_IMG_UNDEFINED	没有限制，可以选取。
ESP_OTA_IMG_INVALID	不会选取。
ESP_OTA_IMG_ABORTED	不会选取。
ESP_OTA_IMG_NEW	如使能*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*，则仅会选取一次。在引导加载程序中，状态立即变为 ESP_OTA_IMG_PENDING_VERIFY。
ESP_OTA_IMG_PENDING_VERIFY	如使能*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*，则不会选取，状态变为 ESP_OTA_IMG_ABORTED。

如果*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 没有使能（默认情况），则*esp_ota_mark_app_valid_cancel_rollback()* 和*esp_ota_mark_app_invalid_rollback_and_reboot()* 为可选功能，ESP_OTA_IMG_NEW 和 ESP_OTA_IMG_PENDING_VERIFY 不会使用。

Kconfig 中的*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 可以帮助用户追踪新版应用程序的第一次启动。应用程序需调用*esp_ota_mark_app_valid_cancel_rollback()* 函数确认可以运行，否则将会在重启时回滚至旧版本。该功能可让用户在启动阶段控制应用程序的可操作性。新版应用程序仅有一次机会尝试是否能成功启动。

回滚过程 *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 使能时，回滚过程如下：

- 新版应用程序下载成功，*esp_ota_set_boot_partition()* 函数将分区设为可启动，状态设为 ESP_OTA_IMG_NEW。该状态表示应用程序为新版本，第一次启动需要监测。
- 重新启动*esp_restart()*。
- 引导加载程序检查 ESP_OTA_IMG_PENDING_VERIFY 状态，如有设置，则将其写入 ESP_OTA_IMG_ABORTED。
- 引导加载程序选取一个新版应用程序来引导，这样应用程序状态就不会设置为 ESP_OTA_IMG_INVALID 或 ESP_OTA_IMG_ABORTED。
- 引导加载程序检查所选取的新版应用程序，若状态设置为 ESP_OTA_IMG_NEW，则写入 ESP_OTA_IMG_PENDING_VERIFY。该状态表示，需确认应用程序的可操作性，如不确认，发

生重启，则状态会重写为 ESP_OTA_IMG_ABORTED （见上文），该应用程序不可再启动，将回滚至上一版本。
- 新版应用程序启动，应进行自测。
- 若通过自测，则必须调用函数*esp_ota_mark_app_valid_cancel_rollback()*，因为新版应用程序在等待确认其可操作性（ESP_OTA_IMG_PENDING_VERIFY 状态）。
- 若未通过自测，则调用函数*esp_ota_mark_app_invalid_rollback_and_reboot()*，回滚至之前能正常工作的应用程序版本，同时将无效的新版本应用程序设置为 ESP_OTA_IMG_INVALID。
- 如果新版应用程序可操作性没有确认，则状态一直为 ESP_OTA_IMG_PENDING_VERIFY。下一次启动时，状态变更为 ESP_OTA_IMG_ABORTED，阻止其再次启动，之后回滚到之前的版本。

意外复位 如果在新版应用第一次启动时发生断电或意外崩溃，则会回滚至之前正常运行的版本。

建议：尽快完成自测，防止因断电回滚。

只有 OTA 分区可以回滚。工厂分区不会回滚。

启动无效/中止的应用程序 用户可以启动此前设置为 ESP_OTA_IMG_INVALID 或 ESP_OTA_IMG_ABORTED 的应用程序：

- 获取最后一个无效应用分区*esp_ota_get_last_invalid_partition()*。
- 将获取的分区传递给*esp_ota_set_boot_partition()*，更新 otadata。
- 重启*esp_restart()*。引导加载程序会启动指定应用程序。

要确定是否在应用程序启动时进行自测，可以调用*esp_ota_get_state_partition()* 函数。如果结果为 ESP_OTA_IMG_PENDING_VERIFY，则需要自测，后续确认应用程序的可操作性。

如何设置状态 下文简单描述了如何设置应用程序状态：

- ESP_OTA_IMG_VALID 由函数*esp_ota_mark_app_valid_cancel_rollback()* 设置。
- 如果*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 没有使能，ESP_OTA_IMG_UNDEFINED 由函数*esp_ota_set_boot_partition()* 设置。
- 如果*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 使能，ESP_OTA_IMG_NEW 由函数*esp_ota_set_boot_partition()* 设置。
- ESP_OTA_IMG_INVALID 由函数*esp_ota_mark_app_invalid_rollback_and_reboot()* 设置。
- 如果应用程序的可操作性无法确认，发生重启（*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 使能），则设置 ESP_OTA_IMG_ABORTED。
- 如果*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 使能，选取的应用程序状态为 ESP_OTA_IMG_NEW，则在引导加载程序中设置 ESP_OTA_IMG_PENDING_VERIFY。

防回滚

防回滚机制可以防止回滚到安全版本号低于芯片 eFuse 中烧录程序的应用程序版本。

设置*CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*，启动防回滚机制。在引导加载程序中选取可启动的应用程序，会额外检查芯片和应用程序镜像的安全版本号。可启动固件中的应用安全版本号必须等于或高于芯片中的应用安全版本号。

CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK 和*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* 一起使用。此时，只有安全版本号等于或高于芯片中的应用安全版本号时才会回滚。

典型的防回滚机制

- 新发布的固件解决了此前版本的安全问题。
- 开发者在确保固件可以运行之后，增加安全版本号，发布固件。
- 下载新版应用程序。
- 运行函数*esp_ota_set_boot_partition()*，将新版应用程序设为可启动。如果新版应用程序的安全版本号低于芯片中的应用安全版本号，新版应用程序会被擦除，无法更新到新固件。

- 重新启动。
- 在引导加载程序中选取安全版本号等于或高于芯片中应用安全版本号的应用程序。如果 otadata 处于初始阶段，通过串行通道加载了安全版本号高于芯片中应用安全版本号的固件，则引导加载程序中 eFuse 的安全版本号会立即更新。
- 新版应用程序启动，之后进行可操作性检测，如果通过检测，则调用函数*esp_ota_mark_app_valid_cancel_rollback()*，将应用程序标记为 ESP_OTA_IMG_VALID，更新芯片中应用程序的安全版本号。注意，如果调用函数*esp_ota_mark_app_invalid_rollback_and_reboot()*，可能会因为设备中没有可启动的应用程序而回滚失败，返回 ESP_ERR_OTA_ROLLBACK_FAILED 错误，应用程序状态一直为 ESP_OTA_IMG_PENDING_VERIFY。
- 如果运行的应用程序处于 ESP_OTA_IMG_VALID 状态，则可再次更新。

建议：

如果想避免因服务器应用程序的安全版本号低于运行的应用程序，造成不必要的下载和擦除，必须从镜像的第一个包中获取 new_app_info.secure_version，和 eFuse 的安全版本号比较。如果 esp_efuse_check_secure_version(new_app_info.secure_version) 函数为真，则下载继续，反之则中断。

```
....
bool image_header_was_checked = false;
while (1) {
    int data_read = esp_http_client_read(client, ota_write_data, BUFFSIZE);
    ...
    if (data_read > 0) {
        if (image_header_was_checked == false) {
            esp_app_desc_t new_app_info;
            if (data_read > sizeof(esp_image_header_t) + sizeof(esp_image_segment_
↪header_t) + sizeof(esp_app_desc_t)) {
                // check current version with downloading
                if (esp_efuse_check_secure_version(new_app_info.secure_version) ==␣
↪false) {
                  ESP_LOGE(TAG, "This a new app can not be downloaded due to a␣
↪secure version is lower than stored in efuse.");
                  http_cleanup(client);
                  task_fatal_error();
                }

                image_header_was_checked = true;

                esp_ota_begin(update_partition, OTA_SIZE_UNKNOWN, &update_handle);
            }
        }
        esp_ota_write( update_handle, (const void *)ota_write_data, data_read);
    }
}
...
```

限制：

- secure_version 字段最多有 16 位。也就是说，防回滚最多可以做 16 次。用户可以使用*CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD* 减少该 eFuse 字段的长度。
- 防回滚不支持工厂和测试分区，因此分区表中不应有设置为 工厂或 测试的分区。

security_version:

- 存储在应用程序镜像中的 esp_app_desc 里。版本号用*CONFIG_BOOTLOADER_APP_SECURE_VERSION* 设置。

没有安全启动的安全 OTA 升级

即便硬件安全启动没有使能，也可验证已签名的 OTA 升级。可通过设置*CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT* 和*CONFIG_SECURE_SIGNED_ON_UPDATE_NO_SECURE_BOOT* 实现。

OTA 工具 `otatool.py`

`app_update` 组件中有 app_update/otatool.py 工具，用于在目标设备上完成下列 OTA 分区相关操作：

- 读取 otadata 分区 (read_otadata)
- 擦除 otadata 分区，将设备复位至工厂应用程序 (erase_otadata)
- 切换 OTA 分区 (switch_ota_partition)
- 擦除 OTA 分区 (erase_ota_partition)
- 写入 OTA 分区 (write_ota_partition)
- 读取 OTA 分区 (read_ota_partition)

用户若想通过编程方式完成相关操作，可从另一个 Python 脚本导入并使用该 OTA 工具，或者从 Shell 脚本调用该 OTA 工具。前者可使用工具的 Python API，后者可使用命令行界面。

Python API 首先，确保已导入 `otatool` 模块。

```
import sys
import os

idf_path = os.environ["IDF_PATH"]  # 从环境中获取 IDF_PATH 的值
otatool_dir = os.path.join(idf_path, "components", "app_update")  # otatool.py␣
↪位于 $IDF_PATH/components/app_update 下

sys.path.append(otatool_dir)  # 使能 Python 寻找 otatool 模块
from otatool import *  # 导入 otatool 模块内的所有名称
```

要使用 OTA 工具的 Python API，第一步是创建 *OtatoolTarget* 对象：

```
# 创建 parttool.py 的目标设备，并将目标设备连接到串行端口 /dev/ttyUSB1
target = OtatoolTarget("/dev/ttyUSB1")
```

现在，可使用创建的 *OtatoolTarget* 在目标设备上完成操作：

```
# 擦除 otadata，将设备复位至工厂应用程序
target.erase_otadata()

# 擦除 OTA 应用程序分区 0
target.erase_ota_partition(0)

# 将启动分区切换至 OTA 应用程序分区 1
target.switch_ota_partition(1)

# 读取 OTA 分区 'ota_3'，将内容保存至文件 'ota_3.bin'
target.read_ota_partition("ota_3", "ota_3.bin")
```

要操作的 OTA 分区通过应用程序分区序号或分区名称指定。

更多关于 Python API 的信息，请查看 OTA 工具的代码注释。

命令行界面 `otatool.py` 的命令行界面具有如下结构：

```
otatool.py [command-args] [subcommand] [subcommand-args]

- command-args - 执行主命令 (otatool.py) 所需的实际参数，多与目标设备有关
- subcommand - 要执行的操作
- subcommand-args - 所选操作的实际参数
```

```
# 擦除 otadata，将设备复位至工厂应用程序
otatool.py --port "/dev/ttyUSB1" erase_otadata

# 擦除 OTA 应用程序分区 0
otatool.py --port "/dev/ttyUSB1" erase_ota_partition --slot 0

# 将启动分区切换至 OTA 应用程序分区 1
otatool.py --port "/dev/ttyUSB1" switch_ota_partition --slot 1

# 读取 OTA 分区 'ota_3'，将内容保存至文件 'ota_3.bin'
otatool.py --port "/dev/ttyUSB1" read_ota_partition --name=ota_3 --output=ota_3.bin
```

更多信息可用 `--help` 指令查看：

```
# 显示可用的子命令和主命令描述
otatool.py --help

# 显示子命令的描述
otatool.py [subcommand] --help
```

相关文档

- 分区表
- 分区 *API*
- *SPI flash API*
- *ESP HTTPS OTA* 升级

应用程序示例

端对端的 OTA 固件升级示例请参考 system/ota。

API 参考

Header File

- components/app_update/include/esp_ota_ops.h
- This header file can be included with:

  ```
  #include "esp_ota_ops.h"
  ```

- This header file is a part of the API provided by the `app_update` component. To declare that your component depends on `app_update`, add the following to your CMakeLists.txt:

  ```
  REQUIRES app_update
  ```

  or

  ```
  PRIV_REQUIRES app_update
  ```

Functions

const *esp_app_desc_t* ***esp_ota_get_app_description**(void)

Return esp_app_desc structure. This structure includes app version.

Return description for running app.

备注： This API is present for backward compatibility reasons. Alternative function with the same functionality is `esp_app_get_description`

返回 Pointer to esp_app_desc structure.

int **esp_ota_get_app_elf_sha256**(char *dst, size_t size)

Fill the provided buffer with SHA256 of the ELF file, formatted as hexadecimal, null-terminated. If the buffer size is not sufficient to fit the entire SHA256 in hex plus a null terminator, the largest possible number of bytes will be written followed by a null.

备注： This API is present for backward compatibility reasons. Alternative function with the same functionality is `esp_app_get_elf_sha256`

参数

- **dst** -- Destination buffer
- **size** -- Size of the buffer

返回 Number of bytes written to dst (including null terminator)

esp_err_t **esp_ota_begin**(const *esp_partition_t* *partition, size_t image_size, *esp_ota_handle_t* *out_handle)

Commence an OTA update writing to the specified partition.

The specified partition is erased to the specified image size.

If image size is not yet known, pass OTA_SIZE_UNKNOWN which will cause the entire partition to be erased.

On success, this function allocates memory that remains in use until esp_ota_end() is called with the returned handle.

Note: If the rollback option is enabled and the running application has the ESP_OTA_IMG_PENDING_VERIFY state then it will lead to the ESP_ERR_OTA_ROLLBACK_INVALID_STATE error. Confirm the running app before to run download a new app, use esp_ota_mark_app_valid_cancel_rollback() function for it (this should be done as early as possible when you first download a new application).

参数

- **partition** -- Pointer to info for partition which will receive the OTA update. Required.
- **image_size** -- Size of new OTA app image. Partition will be erased in order to receive this size of image. If 0 or OTA_SIZE_UNKNOWN, the entire partition is erased.
- **out_handle** -- On success, returns a handle which should be used for subsequent esp_ota_write() and esp_ota_end() calls.

返回

- ESP_OK: OTA operation commenced successfully.
- ESP_ERR_INVALID_ARG: partition or out_handle arguments were NULL, or partition doesn't point to an OTA app partition.
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_OTA_PARTITION_CONFLICT: Partition holds the currently running firmware, cannot update in place.
- ESP_ERR_NOT_FOUND: Partition argument not found in partition table.
- ESP_ERR_OTA_SELECT_INFO_INVALID: The OTA data partition contains invalid data.
- ESP_ERR_INVALID_SIZE: Partition doesn't fit in configured flash size.

- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- ESP_ERR_OTA_ROLLBACK_INVALID_STATE: If the running app has not confirmed state. Before performing an update, the application must be valid.

esp_err_t **esp_ota_write**(*esp_ota_handle_t* handle, const void *data, size_t size)

Write OTA update data to partition.

This function can be called multiple times as data is received during the OTA operation. Data is written sequentially to the partition.

参数

- **handle** -- Handle obtained from esp_ota_begin
- **data** -- Data buffer to write
- **size** -- Size of data buffer in bytes.

返回

- ESP_OK: Data was written to flash successfully, or size = 0
- ESP_ERR_INVALID_ARG: handle is invalid.
- ESP_ERR_OTA_VALIDATE_FAILED: First byte of image contains invalid app image magic byte.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- ESP_ERR_OTA_SELECT_INFO_INVALID: OTA data partition has invalid contents

esp_err_t **esp_ota_write_with_offset**(*esp_ota_handle_t* handle, const void *data, size_t size, uint32_t offset)

Write OTA update data to partition at an offset.

This function can write data in non-contiguous manner. If flash encryption is enabled, data should be 16 bytes aligned.

备注： While performing OTA, if the packets arrive out of order, esp_ota_write_with_offset() can be used to write data in non-contiguous manner. Use of esp_ota_write_with_offset() in combination with esp_ota_write() is not recommended.

参数

- **handle** -- Handle obtained from esp_ota_begin
- **data** -- Data buffer to write
- **size** -- Size of data buffer in bytes
- **offset** -- Offset in flash partition

返回

- ESP_OK: Data was written to flash successfully.
- ESP_ERR_INVALID_ARG: handle is invalid.
- ESP_ERR_OTA_VALIDATE_FAILED: First byte of image contains invalid app image magic byte.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- ESP_ERR_OTA_SELECT_INFO_INVALID: OTA data partition has invalid contents

esp_err_t **esp_ota_end**(*esp_ota_handle_t* handle)

Finish OTA update and validate newly written app image.

备注： After calling esp_ota_end(), the handle is no longer valid and any memory associated with it is freed (regardless of result).

参数 **handle** -- Handle obtained from esp_ota_begin().
返回

- ESP_OK: Newly written OTA app image is valid.
- ESP_ERR_NOT_FOUND: OTA handle was not found.
- ESP_ERR_INVALID_ARG: Handle was never written to.
- ESP_ERR_OTA_VALIDATE_FAILED: OTA image is invalid (either not a valid app image, or - if secure boot is enabled - signature failed to verify.)
- ESP_ERR_INVALID_STATE: If flash encryption is enabled, this result indicates an internal error writing the final encrypted bytes to flash.

esp_err_t **esp_ota_abort**(*esp_ota_handle_t* handle)

Abort OTA update, free the handle and memory associated with it.

参数 **handle** -- obtained from esp_ota_begin().
返回

- ESP_OK: Handle and its associated memory is freed successfully.
- ESP_ERR_NOT_FOUND: OTA handle was not found.

esp_err_t **esp_ota_set_boot_partition**(const *esp_partition_t* *partition)

Configure OTA data for a new boot partition.

备注: If this function returns ESP_OK, calling esp_restart() will boot the newly configured app partition.

参数 **partition** -- Pointer to info for partition containing app image to boot.
返回

- ESP_OK: OTA data updated, next reboot will use specified partition.
- ESP_ERR_INVALID_ARG: partition argument was NULL or didn't point to a valid OTA partition of type "app".
- ESP_ERR_OTA_VALIDATE_FAILED: Partition contained invalid app image. Also returned if secure boot is enabled and signature validation failed.
- ESP_ERR_NOT_FOUND: OTA data partition not found.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash erase or write failed.

const *esp_partition_t* ***esp_ota_get_boot_partition**(void)

Get partition info of currently configured boot app.

If esp_ota_set_boot_partition() has been called, the partition which was set by that function will be returned.

If esp_ota_set_boot_partition() has not been called, the result is usually the same as esp_ota_get_running_partition(). The two results are not equal if the configured boot partition does not contain a valid app (meaning that the running partition will be an app that the bootloader chose via fallback).

If the OTA data partition is not present or not valid then the result is the first app partition found in the partition table. In priority order, this means: the factory app, the first OTA app slot, or the test app partition.

Note that there is no guarantee the returned partition is a valid app. Use esp_image_verify(ESP_IMAGE_VERIFY, ...) to verify if the returned partition contains a bootable image.

返回 Pointer to info for partition structure, or NULL if partition table is invalid or a flash read operation failed. Any returned pointer is valid for the lifetime of the application.

const *esp_partition_t* ***esp_ota_get_running_partition**(void)

Get partition info of currently running app.

This function is different to esp_ota_get_boot_partition() in that it ignores any change of selected boot partition caused by esp_ota_set_boot_partition(). Only the app whose code is currently running will have its partition information returned.

The partition returned by this function may also differ from esp_ota_get_boot_partition() if the configured boot partition is somehow invalid, and the bootloader fell back to a different app partition at boot.

返回 Pointer to info for partition structure, or NULL if no partition is found or flash read operation failed. Returned pointer is valid for the lifetime of the application.

const *esp_partition_t* ***esp_ota_get_next_update_partition**(const *esp_partition_t* *start_from)

Return the next OTA app partition which should be written with a new firmware.

Call this function to find an OTA app partition which can be passed to esp_ota_begin().

Finds next partition round-robin, starting from the current running partition.

参数 **start_from** -- If set, treat this partition info as describing the current running partition. Can be NULL, in which case esp_ota_get_running_partition() is used to find the currently running partition. The result of this function is never the same as this argument.

返回 Pointer to info for partition which should be updated next. NULL result indicates invalid OTA data partition, or that no eligible OTA app slot partition was found.

esp_err_t **esp_ota_get_partition_description**(const *esp_partition_t* *partition, *esp_app_desc_t* *app_desc)

Returns esp_app_desc structure for app partition. This structure includes app version.

Returns a description for the requested app partition.

参数

- **partition** -- **[in]** Pointer to app partition. (only app partition)
- **app_desc** -- **[out]** Structure of info about app.

返回

- ESP_OK Successful.
- ESP_ERR_NOT_FOUND app_desc structure is not found. Magic word is incorrect.
- ESP_ERR_NOT_SUPPORTED Partition is not application.
- ESP_ERR_INVALID_ARG Arguments is NULL or if partition's offset exceeds partition size.
- ESP_ERR_INVALID_SIZE Read would go out of bounds of the partition.
- or one of error codes from lower-level flash driver.

esp_err_t **esp_ota_get_bootloader_description**(const *esp_partition_t* *bootloader_partition, *esp_bootloader_desc_t* *desc)

Returns the description structure of the bootloader.

参数

- **bootloader_partition** -- **[in]** Pointer to bootloader partition. If NULL, then the current bootloader is used (the default location). offset = CONFIG_BOOTLOADER_OFFSET_IN_FLASH, size = CONFIG_PARTITION_TABLE_OFFSET - CONFIG_BOOTLOADER_OFFSET_IN_FLASH,
- **desc** -- **[out]** Structure of info about bootloader.

返回

- ESP_OK Successful.
- ESP_ERR_NOT_FOUND Description structure is not found in the bootloader image. Magic byte is incorrect.
- ESP_ERR_INVALID_ARG Arguments is NULL.
- ESP_ERR_INVALID_SIZE Read would go out of bounds of the partition.
- or one of error codes from lower-level flash driver.

uint8_t **esp_ota_get_app_partition_count**(void)

Returns number of ota partitions provided in partition table.

返回

- Number of OTA partitions

esp_err_t **esp_ota_mark_app_valid_cancel_rollback**(void)

This function is called to indicate that the running app is working well.

返回

- ESP_OK: if successful.

esp_err_t **esp_ota_mark_app_invalid_rollback_and_reboot**(void)

This function is called to roll back to the previously workable app with reboot.

If rollback is successful then device will reset else API will return with error code. Checks applications on a flash drive that can be booted in case of rollback. If the flash does not have at least one app (except the running app) then rollback is not possible.

返回

- ESP_FAIL: if not successful.
- ESP_ERR_OTA_ROLLBACK_FAILED: The rollback is not possible due to flash does not have any apps.

const *esp_partition_t* ***esp_ota_get_last_invalid_partition**(void)

Returns last partition with invalid state (ESP_OTA_IMG_INVALID or ESP_OTA_IMG_ABORTED).

返回 partition.

esp_err_t **esp_ota_get_state_partition**(const *esp_partition_t* *partition, esp_ota_img_states_t *ota_state)

Returns state for given partition.

参数

- **partition** -- **[in]** Pointer to partition.
- **ota_state** -- **[out]** state of partition (if this partition has a record in otadata).

返回

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: partition or ota_state arguments were NULL.
- ESP_ERR_NOT_SUPPORTED: partition is not ota.
- ESP_ERR_NOT_FOUND: Partition table does not have otadata or state was not found for given partition.

esp_err_t **esp_ota_erase_last_boot_app_partition**(void)

Erase previous boot app partition and corresponding otadata select for this partition.

When current app is marked to as valid then you can erase previous app partition.

返回

- ESP_OK: Successful, otherwise ESP_ERR.

bool **esp_ota_check_rollback_is_possible**(void)

Checks applications on the slots which can be booted in case of rollback.

These applications should be valid (marked in otadata as not UNDEFINED, INVALID or ABORTED and crc is good) and be able booted, and secure_version of app >= secure_version of efuse (if anti-rollback is enabled).

返回

- True: Returns true if the slots have at least one app (except the running app).
- False: The rollback is not possible.

esp_err_t **esp_ota_revoke_secure_boot_public_key**(*esp_ota_secure_boot_public_key_index_t* index)

Revokes the signature digest denoted by the given index. This should be called in the application only after the rollback logic otherwise the device may end up in unrecoverable state.

Relevant for Secure boot v2 on ESP32-S2, ESP32-S3, ESP32-C3, ESP32-C6, ESP32-H2 where up to 3 key digests can be stored (Key #N-1, Key #N, Key #N+1). When a key used to sign an app is invalidated, an OTA update is to be sent with an app signed with at least one of the other two keys which has not been revoked already. After successfully booting the OTA app should call this function to revoke Key #N-1.

参数 **index** -- - The index of the signature block to be revoked

返回

- ESP_OK: If revocation is successful.
- ESP_ERR_INVALID_ARG: If the index of the public key to be revoked is incorrect.

- ESP_FAIL: If secure boot v2 has not been enabled.

Macros

OTA_SIZE_UNKNOWN

Used for esp_ota_begin() if new image size is unknown

OTA_WITH_SEQUENTIAL_WRITES

Used for esp_ota_begin() if new image size is unknown and erase can be done in incremental manner (assuming write operation is in continuous sequence)

ESP_ERR_OTA_BASE

Base error code for ota_ops api

ESP_ERR_OTA_PARTITION_CONFLICT

Error if request was to write or erase the current running partition

ESP_ERR_OTA_SELECT_INFO_INVALID

Error if OTA data partition contains invalid content

ESP_ERR_OTA_VALIDATE_FAILED

Error if OTA app image is invalid

ESP_ERR_OTA_SMALL_SEC_VER

Error if the firmware has a secure version less than the running firmware.

ESP_ERR_OTA_ROLLBACK_FAILED

Error if flash does not have valid firmware in passive partition and hence rollback is not possible

ESP_ERR_OTA_ROLLBACK_INVALID_STATE

Error if current active firmware is still marked in pending validation state (ESP_OTA_IMG_PENDING_VERIFY), essentially first boot of firmware image post upgrade and hence firmware upgrade is not possible

Type Definitions

typedef uint32_t **esp_ota_handle_t**

Opaque handle for an application OTA update.

esp_ota_begin() returns a handle which is then used for subsequent calls to esp_ota_write() and esp_ota_end().

Enumerations

enum **esp_ota_secure_boot_public_key_index_t**

Secure Boot V2 public key indexes.

Values:

enumerator **SECURE_BOOT_PUBLIC_KEY_INDEX_0**

Points to the 0th index of the Secure Boot v2 public key

enumerator **SECURE_BOOT_PUBLIC_KEY_INDEX_1**

Points to the 1st index of the Secure Boot v2 public key

enumerator **SECURE_BOOT_PUBLIC_KEY_INDEX_2**

Points to the 2nd index of the Secure Boot v2 public key

OTA 升级失败排查

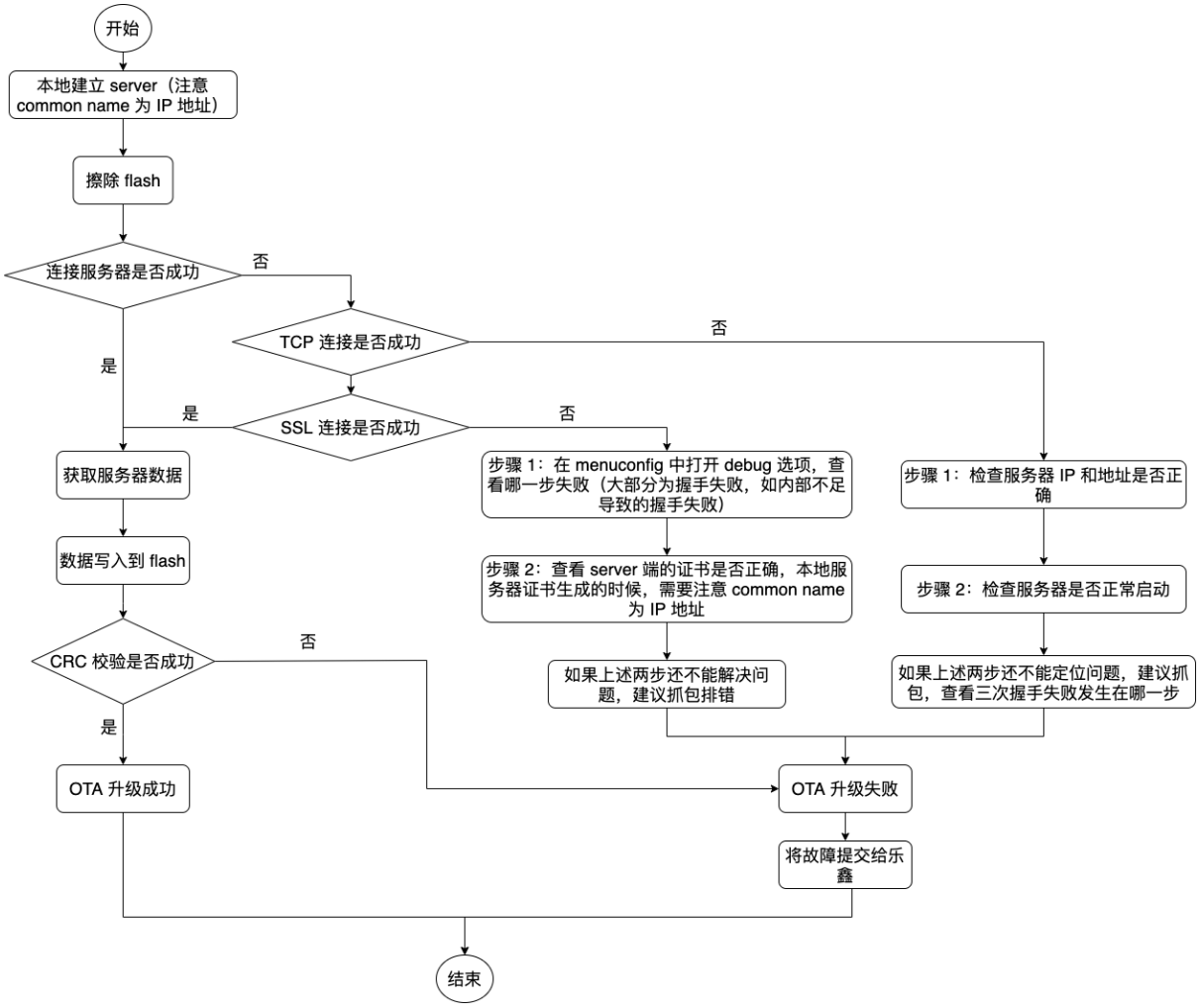


图 56: OTA 升级失败时如何排查（点击放大）

2.9.25 电源管理

概述

ESP-IDF 中集成的电源管理算法可以根据应用程序组件的需求，调整外围总线 (APB) 频率和 CPU 频率，并使芯片进入 Light-sleep 模式，尽可能减少运行应用程序的功耗。

应用程序组件可以通过创建和获取电源管理锁来控制功耗。

例如：

- 对于从 APB 获得时钟频率的外设，其驱动可以要求在使用该外设时，将 APB 频率设置为 80 MHz。
- RTOS 可以要求 CPU 在有任务准备开始运行时以最高配置频率工作。
- 一些外设可能需要中断才能启用，因此其驱动也会要求禁用 Light-sleep 模式。

请求较高的 APB 频率或 CPU 频率以及禁用 Light-sleep 模式会增加功耗，因此请将组件使用的电源管理锁降到最少。

电源管理配置

编译时可使用*CONFIG_PM_ENABLE* 选项启用电源管理功能。

启用电源管理功能将会增加中断延迟。额外延迟与多个因素有关，例如：CPU 频率、单/双核模式、是否需要进行频率切换等。CPU 频率为 240 MHz 且未启用频率调节时，最小额外延迟为 0.2 us；如果启用频率调节，且在中断入口将频率由 40 MHz 调节至 80 MHz，则最大额外延迟为 40 us。

通过调用`esp_pm_configure()` 函数可以在应用程序中启用动态调频 (DFS) 功能和自动 Light-sleep 模式。此函数的参数`esp_pm_config_t` 定义了频率调节的相关设置。在此参数结构中，需要初始化以下三个字段：

- `max_freq_mhz`：最大 CPU 频率 (MHz)，即获取 `ESP_PM_CPU_FREQ_MAX` 锁后所使用的频率。该字段通常设置为*CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ*。
- `min_freq_mhz`：最小 CPU 频率 (MHz)，即未持有电源管理锁时所使用的频率。
  - `light_sleep_enable`：没有获取任何管理锁时，决定系统是否需要自动进入 Light-sleep 状态 (`true`/`false`)。

如果在 menuconfig 中启用了*CONFIG_PM_DFS_INIT_AUTO* 选项，最大 CPU 频率将由*CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ* 设置决定，最小 CPU 频率将锁定为 XTAL 频率。

备注： 自动 Light-sleep 模式基于 FreeRTOS Tickless Idle 功能，因此如果在 menuconfig 中没有启用*CONFIG_FREERTOS_USE_TICKLESS_IDLE* 选项，在请求自动 Light-sleep 时，`esp_pm_configure()` 将会返回 *ESP_ERR_NOT_SUPPORTED* 错误。

备注： Light-sleep 状态下，外设设有时钟门控，不会产生来自 GPIO 和内部外设的中断。睡眠模式 文档中所提到的唤醒源可用于从 Light-sleep 状态触发唤醒。

例如，EXT0 和 EXT1 唤醒源可以通过 GPIO 唤醒芯片。

电源管理锁

应用程序可以通过获取或释放管理锁来控制电源管理算法。应用程序获取电源管理锁后，电源管理算法的操作将受到下面的限制。释放电源管理锁后，限制解除。

电源管理锁设有获取/释放计数器，如果已多次获取电源管理锁，则需要将电源管理锁释放相同次数以解除限制。

ESP32-P4 支持下表中三种电源管理锁。

电源管理锁	描述
`ESP_PM_CPU_FREQ_MAX`	请求使用`esp_pm_configure()` 将 CPU 频率设置为最大值。ESP32-P4 可以将该值设置为 360 MHz。
`ESP_PM_APB_FREQ_MAX`	请求将 APB 频率设置为最大值，ESP32-P4 支持的最大频率为 80 MHz。
`ESP_PM_NO_LIGHT_SLEEP`	禁止自动切换至 Light-sleep 模式。

ESP32-P4 电源管理算法

下表列出了启用动态调频时如何切换 CPU 频率和 APB 频率。可以使用*esp_pm_configure()*或*CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ* 指定 CPU 最大频率。

CPU 最高频率	电源管理锁获取情况	APB 频率和 CPU 频率
360	获取 ESP_PM_CPU_FREQ_MAX	CPU: 360 MHz APB: 90 Mhz
	获取 ESP_PM_APB_FREQ_MAX，未获得 ESP_PM_CPU_FREQ_MAX	CPU: 90 MHz APB: 90 Mhz
	无	使用*esp_pm_configure()*为二者设置最小值

如果没有获取任何管理锁，调用*esp_pm_configure()* 将启动 Light-sleep 模式。Light-sleep 模式持续时间由以下因素决定：

- 处于阻塞状态的 FreeRTOS 任务数（有限超时）
- 高分辨率定时器 API 注册的计数器数量

也可以设置 Light-sleep 模式在最近事件（任务解除阻塞，或计时器超时）之前的持续时间，在持续时间结束后再唤醒芯片。

为了跳过不必要的唤醒，可以将 skip_unhandled_events 选项设置为 true 来初始化 esp_timer。带有此标志的定时器不会唤醒系统，有助于减少功耗。

动态调频和外设驱动

启用动态调频后，APB 频率可在一个 RTOS 滴答周期内多次更改。有些外设不受 APB 频率变更的影响，但有些外设可能会出现问题。例如，Timer Group 外设定时器会继续计数，但定时器计数的速度将随 APB 频率的变更而变更。

时钟频率不受 APB 频率影响的外设时钟源通常有 REF_TICK, XTAL, RC_FAST (i.e., RTC_8M)。因此，为了保证外设在 DFS 期间的所有行为一致，建议在上述时钟中选择其一作为外设的时钟源。如果想要了解更多详情可以浏览每个外设”API 参考 > 外设 API“页面的“电源管理”章节。

目前以下外设驱动程序可感知动态调频，并在调频期间使用 ESP_PM_APB_FREQ_MAX 锁：

- SPI master
- I2C
- I2S（如果 APLL 锁在使用中，I2S 则会启用 ESP_PM_NO_LIGHT_SLEEP 锁）
- SDMMC

启用以下驱动程序时，将占用 ESP_PM_APB_FREQ_MAX 锁：

- **SPI slave**：从调用*spi_slave_initialize()* 至*spi_slave_free()* 期间。
- **GPTimer**：从调用*gptimer_enable()* 至*gptimer_disable()* 期间。
- **Ethernet**：从调用*esp_eth_driver_install()* 至*esp_eth_driver_uninstall()* 期间。
- **TWAI**：从调用*twai_driver_install()* 至*twai_driver_uninstall()* 期间 (只有在 TWAI 时钟源选择为 TWAI_CLK_SRC_APB 的时候生效)。
- **PCNT**：从调用*pcnt_unit_enable()* 至*pcnt_unit_disable()* 期间。
- **Sigma-delta**：从调用*sdm_channel_enable()* 至*sdm_channel_disable()* 期间。

- **MCPWM**: 从调用*mcpwm_timer_enable()* 至*mcpwm_timer_disable()* 期间，以及调用*mcpwm_capture_timer_enable()* 至*mcpwm_capture_timer_disable()* 期间。

以下外设驱动程序无法感知动态调频，应用程序需自己获取/释放管理锁：

- 旧版 PCNT 驱动
- 旧版 Sigma-delta 驱动
- 旧版定时器驱动 (Timer Group)
- 旧版 MCPWM 驱动

Light-sleep 外设下电

ESP32-P4 支持在 Light-sleep 时掉电外设的电源域.

如果在 menuconfig 中启用了*CONFIG_PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP*，在初始化外设时，驱动会将外设工作的寄存器上下文注册到休眠备份链表中，在进入休眠前，REG_DMA 外设会读取休眠备份链表中的配置，根据链表中的配置将外设的寄存器上下文备份至内存，REG_DMA 也会在唤醒时将上下文从内存恢复到外设寄存中。

目前 IDF 支持以下外设的 Light-sleep 上下文备份：

- INT_MTX
- TEE/APM
- IO_MUX / GPIO
- Timer Group 0 & Timer Group 1
- SPI0/1
- SYSTIMER
- All UARTs

以下外设尚未支持：

- ETM
- ASSIST_DEBUG
- Trace
- Crypto: AES/ECC/HMAC/RSA/SHA/DS/XTA_AES/ECDSA
- SPI2
- I2S
- PCNT
- USB-Serial-JTAG
- TWAI
- LEDC
- MCPWM
- SARADC
- SDIO
- PARL_IO

对于未支持 Light-sleep 上下文备份的外设，若启用了电源管理功能，应在外设工作时持有 ESP_PM_NO_LIGHT_SLEEP 锁以避免进入休眠导致外设工作上下文丢失。

备注： 当外设电源域在睡眠期间断电时，IO_MUX 和 GPIO 模块都处于下电状态，这意味着芯片引脚的状态不会受这些模块控制。要在休眠期间保持 IO 的状态，需要在配置 GPIO 状态前后调用*gpio_hold_dis()* 和*gpio_hold_en()*。此操作可确保 IO 配置被锁存，防止 IO 在睡眠期间浮空。

API 参考

Header File

- components/esp_pm/include/esp_pm.h
- This header file can be included with:

```
#include "esp_pm.h"
```

- This header file is a part of the API provided by the esp_pm component. To declare that your component depends on esp_pm, add the following to your CMakeLists.txt:

```
REQUIRES esp_pm
```

or

```
PRIV_REQUIRES esp_pm
```

Functions

esp_err_t **esp_pm_configure**(const void *config)

Set implementation-specific power management configuration.

参数 **config** -- pointer to implementation-specific configuration structure (e.g. esp_pm_config_esp32)

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the configuration values are not correct
- ESP_ERR_NOT_SUPPORTED if certain combination of values is not supported, or if CONFIG_PM_ENABLE is not enabled in sdkconfig

esp_err_t **esp_pm_get_configuration**(void *config)

Get implementation-specific power management configuration.

参数 **config** -- pointer to implementation-specific configuration structure (e.g. esp_pm_config_esp32)

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the pointer is null

esp_err_t **esp_pm_lock_create**(*esp_pm_lock_type_t* lock_type, int arg, const char *name, *esp_pm_lock_handle_t* *out_handle)

Initialize a lock handle for certain power management parameter.

When lock is created, initially it is not taken. Call esp_pm_lock_acquire to take the lock.

This function must not be called from an ISR.

备注: If the lock_type argument is not valid, it will cause an abort.

参数

- **lock_type** -- Power management constraint which the lock should control
- **arg** -- argument, value depends on lock_type, see esp_pm_lock_type_t
- **name** -- arbitrary string identifying the lock (e.g. "wifi" or "spi"). Used by the esp_pm_dump_locks function to list existing locks. May be set to NULL. If not set to NULL, must point to a string which is valid for the lifetime of the lock.
- **out_handle** -- **[out]** handle returned from this function. Use this handle when calling esp_pm_lock_delete, esp_pm_lock_acquire, esp_pm_lock_release. Must not be NULL.

返回

- ESP_OK on success
- ESP_ERR_NO_MEM if the lock structure can not be allocated

- ESP_ERR_INVALID_ARG if out_handle is NULL
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

esp_err_t **esp_pm_lock_acquire**(*esp_pm_lock_handle_t* handle)

Take a power management lock.

Once the lock is taken, power management algorithm will not switch to the mode specified in a call to esp_pm_lock_create, or any of the lower power modes (higher numeric values of 'mode').

The lock is recursive, in the sense that if esp_pm_lock_acquire is called a number of times, esp_pm_lock_release has to be called the same number of times in order to release the lock.

This function may be called from an ISR.

This function is not thread-safe w.r.t. calls to other esp_pm_lock_* functions for the same handle.

参数 **handle** -- handle obtained from esp_pm_lock_create function

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

esp_err_t **esp_pm_lock_release**(*esp_pm_lock_handle_t* handle)

Release the lock taken using esp_pm_lock_acquire.

Call to this functions removes power management restrictions placed when taking the lock.

Locks are recursive, so if esp_pm_lock_acquire is called a number of times, esp_pm_lock_release has to be called the same number of times in order to actually release the lock.

This function may be called from an ISR.

This function is not thread-safe w.r.t. calls to other esp_pm_lock_* functions for the same handle.

参数 **handle** -- handle obtained from esp_pm_lock_create function

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if lock is not acquired
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

esp_err_t **esp_pm_lock_delete**(*esp_pm_lock_handle_t* handle)

Delete a lock created using esp_pm_lock.

The lock must be released before calling this function.

This function must not be called from an ISR.

参数 **handle** -- handle obtained from esp_pm_lock_create function

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle argument is NULL
- ESP_ERR_INVALID_STATE if the lock is still acquired
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

esp_err_t **esp_pm_dump_locks**(FILE *stream)

Dump the list of all locks to stderr

This function dumps debugging information about locks created using esp_pm_lock_create to an output stream.

This function must not be called from an ISR. If esp_pm_lock_acquire/release are called while this function is running, inconsistent results may be reported.

参数 **stream** -- stream to print information to; use stdout or stderr to print to the console; use fmemopen/open_memstream to print to a string buffer.

返回

- ESP_OK on success

- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

Structures

struct **esp_pm_config_t**

Power management config.

Pass a pointer to this structure as an argument to esp_pm_configure function.

Public Members

int **max_freq_mhz**

Maximum CPU frequency, in MHz

int **min_freq_mhz**

Minimum CPU frequency to use when no locks are taken, in MHz

bool **light_sleep_enable**

Enter light sleep when no locks are taken

Type Definitions

typedef *esp_pm_config_t* **esp_pm_config_esp32_t**

backward compatibility newer chips no longer require this typedef

typedef *esp_pm_config_t* **esp_pm_config_esp32s2_t**

typedef *esp_pm_config_t* **esp_pm_config_esp32s3_t**

typedef *esp_pm_config_t* **esp_pm_config_esp32c3_t**

typedef *esp_pm_config_t* **esp_pm_config_esp32c2_t**

typedef *esp_pm_config_t* **esp_pm_config_esp32c6_t**

typedef struct esp_pm_lock ***esp_pm_lock_handle_t**

Opaque handle to the power management lock.

Enumerations

enum **esp_pm_lock_type_t**

Power management constraints.

Values:

enumerator **ESP_PM_CPU_FREQ_MAX**

Require CPU frequency to be at the maximum value set via esp_pm_configure. Argument is unused and should be set to 0.

enumerator **ESP_PM_APB_FREQ_MAX**

Require APB frequency to be at the maximum value supported by the chip. Argument is unused and should be set to 0.

enumerator **ESP_PM_NO_LIGHT_SLEEP**

Prevent the system from going into light sleep. Argument is unused and should be set to 0.

enumerator **ESP_PM_LOCK_MAX**

2.9.26 POSIX Thread

概述

ESP-IDF 基于 FreeRTOS，但提供了一系列与 POSIX 兼容的 API，以便轻松移植第三方代码，例如支持 pthread 常用的 `pthread` API。

在 ESP-IDF 中，pthread 对 FreeRTOS 中的等效功能进行了包装。pthread API 所需的运行内存或性能开销很低，但 pthread 或 FreeRTOS 的可用功能并非都可以通过 ESP-IDF 的 pthread 支持来实现。

添加标准 `pthread.h` 头文件后可以在 ESP-IDF 中使用 pthread，该头文件已包含在工具链 libc 中。还有另一个专用于 ESP-IDF 的头文件 `esp_pthread.h`，其中提供了一些额外的非 POSIX API，以便通过 pthread 使用一些 ESP-IDF 功能。

C++ 标准库中的 `std::thread`、`std::mutex`、`std::condition_variable` 等功能也是通过 pthread（利用 GCC libstdc++）实现的。因此，本文档提到的限制条件也同样适用于 C++ 标准库中等效功能。

RTOS 集成

与许多使用 pthread 的操作系统不同，ESP-IDF 是一个实时操作系统，具有实时调度程序。这意味着只有当一个更高优先级的任务准备就绪、线程在 OS 同步结构（如 mutex）上发生阻塞、或者线程调用 `sleep`、*`vTaskDelay()`*、`usleep` 等函数时，线程才会停止运行。

备注： 如果调用 C 标准库或 C++ sleep 函数，例如在 `unistd.h` 中定义的 `usleep`，那么只有当睡眠时间超过一个 *FreeRTOS 滴答周期* 时，任务才会阻塞并让出内核。如果时间较短，线程将处于忙等待状态，不会让步给另一个 RTOS 任务。

默认情况下，所有 pthread 具有相同的 RTOS 优先级，但可以通过调用*ESP-IDF 提供的扩展 API* 对此优先级进行更改。

标准功能

ESP-IDF 中实现了以下标准 API。

请参考 标准 pthread 文档 或 `pthread.h`，了解每个函数标准参数和行为的详细信息。下文还列出了 pthread API 与标准 API 相比的差异或局限性。

线程 API

- **`pthread_create()`**
  - `attr` 参数仅支持设置堆栈大小和分离状态，其他属性字段将被忽略。
  - 与 FreeRTOS 任务函数不同，`start_routine` 函数允许返回。如果函数返回，分离类型的线程会被自动删除。而默认的可连接类型线程将被挂起，直到调用 `pthread_join()`。

- pthread_join()
- pthread_detach()
- pthread_exit()
- sched_yield()
- **pthread_self()**
  - 如果从不是 pthread 的 FreeRTOS 任务中调用此函数，断言会失败。
- pthread_equal()

线程属性

- pthread_attr_init()
- **pthread_attr_destroy()**
  - 此函数不需要释放任何资源，而是将 attr 结构体重置为默认值。其实现与 pthread_attr_init() 相同。
- pthread_attr_getstacksize() / pthread_attr_setstacksize()
- pthread_attr_getdetachstate() / pthread_attr_setdetachstate()

Once

- pthread_once()

支持静态初始化常量 PTHREAD_ONCE_INIT。

备注： 在使用 pthread 或 FreeRTOS API 创建的任务中都可以调用此函数。

互斥锁 POSIX 互斥锁被实现为 FreeRTOS 互斥信号量（普通类型用于“快速”或“错误检查”互斥锁，递归类型用于“递归”互斥锁），因此与使用 xSemaphoreCreateMutex() 创建的互斥锁具有相同的优先级继承行为。

- pthread_mutex_init()
- pthread_mutex_destroy()
- pthread_mutex_lock()
- pthread_mutex_timedlock()
- pthread_mutex_trylock()
- pthread_mutex_unlock()
- pthread_mutexattr_init()
- pthread_mutexattr_destroy()
- pthread_mutexattr_gettype() / pthread_mutexattr_settype()

支持静态初始化常量 PTHREAD_MUTEX_INITIALIZER，但不支持其他互斥锁类型的非标准静态初始化常量。

备注： 在使用 pthread 或 FreeRTOS API 创建的任务中都可以调用这些函数。

条件变量

- **pthread_cond_init()**
  - attr 参数未实现，将被忽略。
- pthread_cond_destroy()
- pthread_cond_signal()
- pthread_cond_broadcast()
- pthread_cond_wait()
- pthread_cond_timedwait()

支持静态初始化常量 PTHREAD_COND_INITIALIZER。

- `pthread_cond_timedwait()` 超时的分辨率为 RTOS 滴答周期（参见*CONFIG_FREERTOS_HZ*）。在请求超时后，超时最多会延迟一个滴答周期。

备注： 在使用 pthread 或 FreeRTOS API 创建的任务中都可以调用这些函数。

信号量 ESP-IDF 中实现了 POSIX **未命名信号量**，下文介绍了可访问的 API。除非另有说明，否则 ESP-IDF 中未命名信号量的实现遵循 POSIX 标准规定的信号量。

- sem_init()
- sem_destroy()
  - `pshared` 被忽略。信号量始终可以在 FreeRTOS 任务之间共享。
- sem_post()
  - 如果信号量的值已经是 `SEM_VALUE_MAX`，则返回 `-1`，并将 `errno` 设置为 `EAGAIN`。
- sem_wait()
- sem_trywait()
- sem_timedwait()
  - 通过 abstime 传递的时间值将被向上舍入到下一个 FreeRTOS 时钟滴答。
  - 超时实际发生在被舍入到的滴答之后，下一个滴答之前。
  - 在计算超时后，任务有可能被立即抢占（可能性较小），从而延迟下一个阻塞操作系统调用的超时，延迟的时间等于抢占的持续时间。
- sem_getvalue()

读/写锁 ESP-IDF 中实现了 POSIX 读写锁规范的以下 API 函数：

- pthread_rwlock_init()
  - `attr` 参数未实现，将被忽略。
- pthread_rwlock_destroy()
- pthread_rwlock_rdlock()
- pthread_rwlock_tryrdlock()
- pthread_rwlock_wrlock()
- pthread_rwlock_trywrlock()
- pthread_rwlock_unlock()

支持静态初始化器常量 `PTHREAD_RWLOCK_INITIALIZER`。

备注： 在 pthread 或 FreeRTOS API 创建的任务中都可以调用此函数。

线程特定数据

- **`pthread_key_create()`**
  - 支持 `destr_function` 参数。如果线程函数正常退出并调用 `pthread_exit()`，此参数就会被调用，或者在使用 FreeRTOS 函数*`vTaskDelete()`* 直接删除了底层任务时被调用。
- `pthread_key_delete()`
- `pthread_setspecific()` / `pthread_getspecific()`

备注： 在 pthread 或 FreeRTOS API 创建的任务中都可以调用此函数。当从 FreeRTOS API 创建的任务中调用这些函数时，必须先启用*CONFIG_FREERTOS_TLSP_DELETION_CALLBACKS* 配置选项，以确保在删除任务之前清理线程数据。

备注： ESP-IDF 中还有其他的线程本地存储选项，包括性能更高的选项。参见*线程局部存储*。

未实现 API

pthread.h 头文件是一个标准头文件，包含了在 ESP-IDF 中未实现的额外 API 和功能，包括：

- 如果调用 pthread_cancel()，返回 ENOSYS。
- pthread_condattr_init() 如果被调用，返回 ENOSYS。

其他未列出的 pthread 函数未在 ESP-IDF 中实现，如果从 ESP-IDF 应用程序中直接引用，将产生编译器错误或链接器错误。如果希望 ESP-IDF 支持某个尚未实现的 API，请 在 GitHub 上发起功能请求 并提供详细信息。

ESP-IDF 扩展

在 esp_pthreads.h 头文件中定义的 API *esp_pthread_set_cfg()* 提供了自定义扩展，能够对后续 pthread_create() 的调用行为进行控制。目前提供以下配置：

- 如果调用 pthread_create() 时未指定默认堆栈大小，可设置新线程的默认堆栈大小（覆盖*CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT*）。
- 堆栈内存属性决定用于分配 pthread 堆栈的内存类型。该字段使用 ESP-IDF 堆属性标志，这一标志在 heap/include/esp_heap_caps.h 文件中定义。为了确保分配的内存能够通过 8 位地址访问 (MALLOC_CAP_8BIT)，用户必须设置相应的标志，此外也可添加其他自定义标志。用户应当确保选择了正确的堆栈内存属性。了解内存位置的更多信息，请参考内存属性 文档。
- 新线程的 RTOS 优先级（覆盖*CONFIG_PTHREAD_TASK_PRIO_DEFAULT*）。
- 新线程的内核亲和性/内核固定（覆盖*CONFIG_PTHREAD_TASK_CORE_DEFAULT*）。
- 新线程的 FreeRTOS 任务名称（覆盖*CONFIG_PTHREAD_TASK_NAME_DEFAULT*）

此配置的作用范围是调用线程或 FreeRTOS 任务，这意味着*esp_pthread_set_cfg()* 可以在不同的线程或任务中独立调用。如果在当前配置中设置了 inherit_cfg 标志，那么当一个线程递归调用 pthread_create() 时，任何新创建的线程都会继承该线程的配置，否则新线程将采用默认配置。

示例

- system/pthread 演示了如何使用 pthread API 创建线程。
- cxx/pthread 演示了如何通过线程使用 C++ 标准库函数。

API 参考

Header File

- components/pthread/include/esp_pthread.h
- This header file can be included with:

```
#include "esp_pthread.h"
```

- This header file is a part of the API provided by the pthread component. To declare that your component depends on pthread, add the following to your CMakeLists.txt:

```
REQUIRES pthread
```

or

```
PRIV_REQUIRES pthread
```

Functions

esp_pthread_cfg_t **esp_pthread_get_default_config**(void)

Creates a default pthread configuration based on the values set via menuconfig.

返回 A default configuration structure.

esp_err_t **esp_pthread_set_cfg**(const *esp_pthread_cfg_t* *cfg)

Configure parameters for creating pthread.

This API allows you to configure how the subsequent pthread_create() call will behave. This call can be used to setup configuration parameters like stack size, priority, configuration inheritance etc.

If the 'inherit' flag in the configuration structure is enabled, then the same configuration is also inherited in the thread subtree.

备注： If cfg->stack_alloc_caps is 0, it is automatically set to valid default stack memory capabilities. If cfg->stack_alloc_caps is non-zero, the developer is responsible for its correctenss. This function only checks that the capabilities are MALLOC_CAP_8BIT, the rest is unchecked.

备注： Passing non-NULL attributes to pthread_create() will override the stack_size parameter set using this API

参数 **cfg** -- The pthread config parameters

返回

- ESP_OK if configuration was successfully set
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_INVALID_ARG if stack_size is less than PTHREAD_STACK_MIN
- ESP_ERR_INVALID_ARG if stack_alloc_caps does not include MALLOC_CAP_8BIT

esp_err_t **esp_pthread_get_cfg**(*esp_pthread_cfg_t* *p)

Get current pthread creation configuration.

This will retrieve the current configuration that will be used for creating threads.

参数 **p** -- Pointer to the pthread config structure that will be updated with the currently configured parameters

返回

- ESP_OK if the configuration was available
- ESP_ERR_NOT_FOUND if a configuration wasn't previously set

esp_err_t **esp_pthread_init**(void)

Initialize pthread library.

Structures

struct **esp_pthread_cfg_t**

pthread configuration structure that influences pthread creation

Public Members

size_t **stack_size**

The stack size of the pthread

size_t **prio**

The thread's priority

bool **inherit_cfg**

Inherit this configuration further

const char ***thread_name**

The thread name.

int **pin_to_core**

The core id to pin the thread to. Has the same value range as xCoreId argument of xTaskCreatePinnedToCore.

uint32_t **stack_alloc_caps**

A bit mask of memory capabilities (MALLOC_CAPS*) to use when allocating the stack. The memory must be 8 bit accessible (MALLOC_CAP_8BIT). The developer is responsible for the correctenss of `stack_alloc_caps`.

Macros

PTHREAD_STACK_MIN

2.9.27 随机数发生器

ESP32-P4 中包含一个硬件随机数发生器 (RNG)，可以调用 API *esp_random()* 和*esp_fill_random()* 从中获取随机数值。

满足下列任意一个或多个条件时，硬件 RNG 会产生真随机数：

- RF 子系统已启用，即 Wi-Fi 或蓝牙已启用。
- 调用*bootloader_random_enable()* 启用了内部熵源，并且熵源尚未被*bootloader_random_disable()* 禁用。
- 在 ESP-IDF *二级引导程序* 运行时。这是因为默认的 ESP-IDF 引导加载程序启动时会调用*bootloader_random_enable()*，并在执行应用程序前调用*bootloader_random_disable()*。

当上述任一条件为真时，物理噪声样本会连续混合到内部硬件 RNG 状态中来提供熵。如需了解详情，请参阅 **ESP32-P4 技术参考手册 > 随机数发生器 (RNG)** [PDF] 章节。

如果上述条件都不满足，那么 RNG 的输出仅应被看作伪随机数。

启动

在启动过程中，ESP-IDF 引导加载程序暂时会启用一个非 RF 熵源（内部参考电压噪声），为首次生成的启动密钥提供熵。当应用程序开始执行后，一直到 Wi-Fi 或蓝牙初始化完成前，通常只有伪随机数可用。

如需在应用程序启动期间临时重启熵源，或为不使用 Wi-Fi 或蓝牙的应用程序临时重启熵源，请调用函数*bootloader_random_enable()* 重启内部熵源。在使用 ADC、或使用 Wi-Fi 或蓝牙前，必须调用函数*bootloader_random_disable()* 以禁用熵源。

备注： ESP-IDF 第二阶段引导加载程序在启动过程中启用的熵源会用熵来初始化内部 RNG 状态。但是，内部硬件 RNG 状态的大小并不足以提供连续的真随机数流。因此，在需要真随机数时必须启用连续的熵源。

备注： 如果应用程序需要真随机数源，但无法永久性地启用硬件熵源，可以考虑使用软件 DRBG（确定性随机数发生器）来实现，如 mbedTLS CTR-DRBG 或 HMAC-DRBG，并使用来自硬件 RNG 真随机数来获取初始熵。

二级熵源

ESP32-P4 RNG 包含一个基于异步 8 MHz 内部振荡器采样的二级熵源（详情请参阅技术参考手册）。该熵源在 ESP-IDF 中始终处于启用状态，并通过硬件持续混合到 RNG 状态中。在测试中，即使在不启用主熵源时，这个二级熵源也足以通过 Dieharder 随机数测试套件（测试输入数据是通过连续重置 ESP32-P4 生成短样本并将其拼接来创建的）。但是，目前只有在同时启用上文所述的主熵源时，才能保证产生真随机数。

API 参考

Header File

- components/esp_hw_support/include/esp_random.h
- This header file can be included with:

```
#include "esp_random.h"
```

Functions

uint32_t **esp_random**(void)

Get one random 32-bit word from hardware RNG.

If Wi-Fi or Bluetooth are enabled, this function returns true random numbers. In other situations, if true random numbers are required then consult the ESP-IDF Programming Guide "Random Number Generation" section for necessary prerequisites.

This function automatically busy-waits to ensure enough external entropy has been introduced into the hardware RNG state, before returning a new random number. This delay is very short (always less than 100 CPU cycles).

返回 Random value between 0 and UINT32_MAX

void **esp_fill_random**(void *buf, size_t len)

Fill a buffer with random bytes from hardware RNG.

备注： This function is implemented via calls to esp_random(), so the same constraints apply.

参数

- **buf** -- Pointer to buffer to fill with random numbers.
- **len** -- Length of buffer in bytes

Header File

- components/bootloader_support/include/bootloader_random.h
- This header file can be included with:

```
#include "bootloader_random.h"
```

- This header file is a part of the API provided by the `bootloader_support` component. To declare that your component depends on `bootloader_support`, add the following to your CMakeLists.txt:

```
REQUIRES bootloader_support
```

or

```
PRIV_REQUIRES bootloader_support
```

Functions

void **bootloader_random_enable**(void)

Enable an entropy source for RNG if RF subsystem is disabled.

The exact internal entropy source mechanism depends on the chip in use but all SoCs use the SAR ADC to continuously mix random bits (an internal noise reading) into the HWRNG. Consult the SoC Technical Reference Manual for more information.

Can also be called from app code, if true random numbers are required without initialized RF subsystem. This might be the case in early startup code of the application when the RF subsystem has not started yet or if the RF subsystem should not be enabled for power saving.

Consult ESP-IDF Programming Guide "Random Number Generation" section for details.

警告： This function is not safe to use if any other subsystem is accessing the RF subsystem or the ADC at the same time!

void **bootloader_random_disable**(void)

Disable entropy source for RNG.

Disables internal entropy source. Must be called after bootloader_random_enable() and before RF subsystem features, ADC, or I2S (ESP32 only) are initialized.

Consult the ESP-IDF Programming Guide "Random Number Generation" section for details.

void **bootloader_fill_random**(void *buffer, size_t length)

Fill buffer with 'length' random bytes.

备注： If this function is being called from app code only, and never from the bootloader, then it's better to call esp_fill_random().

参数

- **buffer** -- Pointer to buffer
- **length** -- This many bytes of random data will be copied to buffer

getrandom()

为方便移植，还提供了与 Linux 的 `getrandom()` 函数兼容的版本：

```
#include <sys/random.h>

ssize_t getrandom(void *buf, size_t buflen, unsigned int flags);
```

此函数通过内部调用*esp_fill_random()* 来实现。

`flags` 参数将被忽略。该函数始终是非阻塞的，但随机数的强度取决于本文档所述条件。

如果 `buf` 参数为 NULL，返回值为 -1，并将 `errno` 设置为 `EFAULT`。否则返回 `buflen`。

getentropy()

为了便于移植，还提供了与 Linux 的 `getentropy()` 函数兼容的版本：

```
#include <unistd.h>

int getentropy(void *buffer, size_t length);
```

此函数通过内部调用 `getrandom()` 实现。

随机数强度取决于本文档所述条件。

如果执行成功则返回 0，否则返回 -1，同时：

- 如果 `buffer` 参数为 NULL，`errno` 设置为 `EFAULT`。
- 如果 `length` 超过 256，`errno` 设置为 `EIO`。

2.9.28 睡眠模式

概述

ESP32-P4 具有 Light-sleep 和 Deep-sleep 两种睡眠节能模式。根据应用所使用的功能，还有一些细分的子睡眠模式。关于这些睡眠模式和其子模式，参见睡眠节能模式。另外，还可以配置一些断电选项来进一步减少功耗，请参见断电选项 。

睡眠模式有多种唤醒源。这些唤醒源也可以组合在一起，此时任何一个唤醒源都可以触发唤醒。唤醒源详细描述了这些唤醒源，以及配置 API。

断电选项和唤醒源的配置不是必要的，并且可以在进入睡眠模式前的任意时候进行。

最后，应用通过调用开始睡眠的 API 来使芯片进入其中一种睡眠模式，更多详情请参见进入睡眠模式。当唤醒条件满足，芯片就会从睡眠中被唤醒。关于如何获取唤醒原因，请参见检查睡眠唤醒原因 。关于醒来后如何处理唤醒源，请参见禁用睡眠模式唤醒源 。

睡眠节能模式

在 Light-sleep 模式下，数字外设、CPU、以及大部分 RAM 都使用时钟门控，同时其供电电压降低。退出该模式后，数字外设、CPU 和 RAM 恢复运行，并且内部状态将被保留。

在 Deep-sleep 模式下，CPU、大部分 RAM、以及所有由时钟 APB_CLK 驱动的数字外设都会被断电。芯片上继续处于供电状态的部分仅包括：

- RTC 控制器
- ULP 协处理器
- RTC 高速内存

唤醒源

通过 API `esp_sleep_enable_X_wakeup` 可启用唤醒源。唤醒源在芯片被唤醒后并不会被禁用，若你不再需要某些唤醒源，可通过 API *`esp_sleep_disable_wakeup_source()`* 将其禁用，详见禁用睡眠模式唤醒源。

以下是 ESP32-P4 所支持的唤醒源。

定时器 RTC 控制器中内嵌定时器，可用于在预定义的时间到达后唤醒芯片。时间精度为微秒，但其实际分辨率依赖于为 RTC_SLOW_CLK 所选择的时钟源。

关于 RTC 时钟选项的更多细节，请参考 **ESP32-P4 技术参考手册 > ULP 协处理器** [PDF]。

在这种唤醒模式下，无需为睡眠模式中的 RTC 外设或内存供电。

调用*`esp_sleep_enable_timer_wakeup()`* 函数可启用使用定时器唤醒睡眠模式。

触摸传感器 RTC IO 模块中包含这样一个逻辑——当发生触摸传感器中断时，触发唤醒。要启用此唤醒源，用户需要在芯片进入睡眠模式前配置触摸传感器中断功能。

可调用*esp_sleep_enable_touchpad_wakeup()* 函数来启用该唤醒源。

外部唤醒 (ext1) RTC 控制器中包含使用多个 RTC GPIO 触发唤醒的逻辑。从以下两个逻辑函数中任选其一，均可触发 ext1 唤醒：

- 当任意一个所选管脚为高电平时唤醒 (ESP_EXT1_WAKEUP_ANY_HIGH)
- 当任意一个所选管脚为低电平时唤醒 (ESP_EXT1_WAKEUP_ANY_LOW)

此唤醒源由 RTC 控制器实现。区别于 `ext0` 唤醒源，在 RTC 外设断电的情况下此唤醒源同样支持唤醒。虽然睡眠期间 RTC IO 所在的 RTC 外设电源域将会断电，但是 ESP-IDF 会自动在系统进入睡眠前锁定唤醒管脚的状态并在退出睡眠时解除锁定，所以仍然可为唤醒管脚配置内部上拉或下拉电阻:

```
esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_ON);
gpio_pullup_dis(gpio_num);
gpio_pulldown_en(gpio_num);
```

如果我们关闭 `RTC_PERIPH` 域，我们将使用 HOLD 功能在睡眠期间维持管脚上的上拉和下拉电阻。所选管脚的 HOLD 功能会在系统真正进入睡眠前被开启，这有助于进一步减小睡眠时的功耗:

```
rtc_gpio_pullup_dis(gpio_num);
rtc_gpio_pulldown_en(gpio_num);
```

如果某些芯片缺少 `RTC_PERIPH` 域，我们只能使用 HOLD 功能来在睡眠期间维持管脚上的上拉和下拉电阻:

```
gpio_pullup_dis(gpio_num);
gpio_pulldown_en(gpio_num);
```

可调用*esp_sleep_enable_ext1_wakeup_io()* 函数可用于增加 ext1 唤醒 IO 并设置相应的唤醒电平。

可调用*esp_sleep_disable_ext1_wakeup_io()* 函数可用于移除 ext1 唤醒 IO。

当前的 RTC 控制器也包含更强大的逻辑，允许配置的 IO 同时使用不同的唤醒电平。这可以通过*esp_sleep_enable_ext1_wakeup_io()* 函数来进行配置。

警告:

- 使用 EXT1 唤醒源时，用于唤醒的 IO pad 将被配置为 RTC IO。因此，在将该 pad 用作数字 GPIO 之前，请调用*rtc_gpio_deinit()* 函数对其进行重新配置。
- RTC 外设在默认情况下配置为断电，此时，唤醒 IO 在进入睡眠状态前将被设置为保持状态。因此，从 Light-sleep 状态唤醒芯片后，请调用 `rtc_gpio_hold_dis` 来禁用保持功能，以便对管脚进行重新配置。对于 Deep-sleep 唤醒，此问题已经在应用启动阶段解决。

ULP 协处理器唤醒 当芯片处于睡眠模式时，ULP 协处理器仍然运行，可用于轮询传感器、监视 ADC 或触摸传感器的值，并在检测到特殊事件时唤醒芯片。ULP 协处理器是 RTC 外设电源域的一部分，运行存储在 RTC 低速内存中的程序。如果这一唤醒源被请求，RTC 低速内存将会在睡眠期间保持供电状态。RTC 外设会在 ULP 协处理器开始运行程序前自动上电；一旦程序停止运行，RTC 外设会再次自动断电。

可调用*esp_sleep_enable_ulp_wakeup()* 函数来启用此唤醒源。

GPIO 唤醒（仅适用于 Light-sleep 模式） 除了上述 EXT0 和 EXT1 唤醒源之外，还有一种从外部唤醒 Light-sleep 模式的方法——使用函数*gpio_wakeup_enable()*。启用该唤醒源后，可将每个管脚单独配置为在高电平或低电平时唤醒。EXT0 和 EXT1 唤醒源只能用于 RTC IO，但此唤醒源既可以用于 RTC IO，可也用于数字 IO。

可调用*esp_sleep_enable_gpio_wakeup()* 函数来启用此唤醒源。

警告： 在进入 Light-sleep 模式前，请查看将要驱动的 GPIO 管脚的电源域。如果有管脚属于 VDD_SPI 电源域，必须将此电源域配置为在睡眠期间保持供电。

例如，在 ESP32-WROOM-32 开发板上，GPIO16 和 GPIO17 连接到 VDD_SPI 电源域。如果这两个管脚被配置为在睡眠期间保持高电平，则需将对应电源域配置为保持供电。为此，可以使用函数 *esp_sleep_pd_config()*：

```
esp_sleep_pd_config(ESP_PD_DOMAIN_VDDSDIO, ESP_PD_OPTION_ON);
```

备注： 在 Light-sleep 模式下，如果设置 Kconfig 选项 *CONFIG_PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP*，为了继续使用 *gpio_wakeup_enable()* 用于 GPIO 唤醒，需要先调用 *rtc_gpio_init()* 和 *rtc_gpio_set_direction()*，用于设置 RTC IO 为输入模式。

或者，可以使用直接调用 *esp_deep_sleep_enable_gpio_wakeup()* 用于 GPIO 唤醒，因为此时 digital IO 的电源域已经被关闭，这个情况类似于进入 Deep-sleep。

UART 唤醒（仅适用于 Light-sleep 模式） 当 ESP32-P4 从外部设备接收 UART 输入时，通常需要在输入数据可用时唤醒芯片。UART 外设支持在 RX 管脚上观测到一定数量的上升沿时，将芯片从 Light-sleep 模式中唤醒。调用 *uart_set_wakeup_threshold()* 函数可设置被观测上升沿的数量。请注意，触发唤醒的字符（及该字符前的所有字符）在唤醒后不会被 UART 接收，因此在发送数据之前，外部设备通常需要首先向 ESP32-P4 额外发送一个字符以触发唤醒。

可调用 *esp_sleep_enable_uart_wakeup()* 函数来启用此唤醒源。

使用 UART 唤醒之后，在芯片 Active 模式下需要让 UART 接受一些数据用来清零内部的唤醒指示信号。不然的话，下一次 UART 唤醒的触发将只需要比配置的阈值少两个上升沿的数量。

> **备注：** 在 Light-sleep 模式下，设置 Kconfig 选项 *CONFIG_PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP* 将使 UART 唤醒失效。

禁用睡眠模式唤醒源 调用 API *esp_sleep_disable_wakeup_source()* 可以禁用给定唤醒源的触发器，从而禁用该唤醒源。此外，如果将参数设置为 `ESP_SLEEP_WAKEUP_ALL`，该函数可用于禁用所有触发器。

断电选项

应用程序可以使用 API *esp_sleep_pd_config()* 强制 RTC 外设和 RTC 内存进入特定断电模式。在 Deep-sleep 模式下，你还可以通过隔离一些 IO 来进一步降低功耗。

RTC 外设和内存断电 默认情况下，调用函数 *esp_deep_sleep_start()* 和 *esp_light_sleep_start()* 后，所有唤醒源不需要的 RTC 电源域都会被断电。可调用函数 *esp_sleep_pd_config()* 来修改这一设置。

ESP32-P4 中只有 RTC 高速内存，因此，如果程序中的某些值被标记为 `RTC_DATA_ATTR`、`RTC_SLOW_ATTR` 或 `RTC_FAST_ATTR` 属性，那么所有这些值都将被存入 RTC 高速内存，默认情况下保持供电。如有需要，也可以使用函数 *esp_sleep_pd_config()* 对其进行修改。

flash 断电 默认情况下，调用函数 *esp_light_sleep_start()* 后，flash **不会**断电，因为在 sleep 过程中断电 flash 存在风险。具体而言，flash 断电需要时间，但是在此期间，系统有可能被唤醒，导致 flash 重新被上电。此时，断电尚未完成又重新上电的硬件行为有可能导致 flash 无法正常工作。

理论上讲，在 flash 完全断电后可以仅唤醒系统，然而现实情况是 flash 断电所需的时间很难预测。如果用户为 flash 供电电路添加了滤波电容，断电所需时间可能会更长。此外，即使可以预知 flash 彻底断电所需

的时间，有时也不能通过设置足够长的睡眠时间来确保 flash 断电的安全（比如，突发的异步唤醒源会使得实际的睡眠时间不可控）。

警告： 如果在 flash 的供电电路上添加了滤波电容，那么应当尽一切可能避免 flash 断电。

因为这些不可控的因素，ESP-IDF 很难保证 flash 断电的绝对安全。因此 ESP-IDF 不推荐用户断电 flash。对于一些功耗敏感型应用，可以通过设置 Kconfig 配置项*CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND* 来减少 Light-sleep 期间 flash 的功耗。这种方式在几乎所有场景下都要比断电 flash 更好，兼顾了安全性和功耗。

值得一提的是，PSRAM 也有一个类似的 Kconfig 配置项*CONFIG_ESP_SLEEP_PSRAM_LEAKAGE_WORKAROUND*。

考虑到有些用户能够充分评估断电 flash 的风险，并希望通过断电 flash 来获得更低的功耗，因此 ESP-IDF 提供了两种断电 flash 的机制：

- 设置 Kconfig 配置项*CONFIG_ESP_SLEEP_POWER_DOWN_FLASH* 将使 ESP-IDF 以一个严格的条件来断电 flash。严格的条件具体指的是，RTC timer 是唯一的唤醒源 **且**睡眠时间比 flash 彻底断电所需时间更长。
- 调用函数 `esp_sleep_pd_config(ESP_PD_DOMAIN_VDDSDIO, ESP_PD_OPTION_OFF)` 将使 ESP-IDF 以一个宽松的条件来断电 flash。宽松的条件具体指的是 RTC timer 唤醒源未被使能 **或**睡眠时间比 flash 彻底断电所需时间更长。

备注：

- Light-sleep 模式下，ESP-IDF 没有提供保证 flash 一定会被断电的机制。
- 不管用户的配置如何，函数*esp_deep_sleep_start()* 都会强制断电 flash。

配置 IO（仅适用于 Deep-sleep） 一些 ESP32-P4 IO 在默认情况下启用内部上拉或下拉电阻。如果这些管脚在 Deep-sleep 模式下中受外部电路驱动，电流流经这些上下拉电阻时，可能会增加电流消耗。

想要隔离这些管脚以避免额外的电流消耗，请调用 `rtc_gpio_isolate()` 函数。

例如，在 ESP32-WROVER 模组上，GPIO12 在外部上拉，但其在 ESP32 芯片中也有内部下拉。这意味着在 Deep-sleep 模式中，电流会流经这些外部和内部电阻，使电流消耗超出可能的最小值。

在函数*esp_deep_sleep_start()* 前增加以下代码即可避免额外电流消耗：

```
rtc_gpio_isolate(GPIO_NUM_12);
```

进入睡眠模式

应用程序通过 API *esp_light_sleep_start()* 或*esp_deep_sleep_start()* 进入 Light-sleep 或 Deep-sleep 模式。此时，系统将按照被请求的唤醒源和断电选项配置有关的 RTC 控制器参数。

允许在未配置唤醒源的情况下进入睡眠模式。在此情况下，芯片将一直处于睡眠模式，直到从外部被复位。

UART 输出处理 在进入睡眠模式之前，调用函数*esp_deep_sleep_start()* 会冲刷掉 UART FIFO 缓存。

当使用函数*esp_light_sleep_start()* 进入 Light-sleep 模式时，UART FIFO 将不会被冲刷。与之相反，UART 输出将被暂停，FIFO 中的剩余字符将在 Light-sleep 唤醒后被发送。

检查睡眠唤醒原因

`esp_sleep_get_wakeup_cause()` 函数可用于检测是何种唤醒源在睡眠期间被触发。

对于触摸传感器唤醒源，可以调用函数`esp_sleep_get_touchpad_wakeup_status()` 来确认触发唤醒的触摸管脚。

对于 ext1 唤醒源，可以调用函数`esp_sleep_get_ext1_wakeup_status()` 来确认触发唤醒的触摸管脚。

应用程序示例

- protocols/sntp 演示如何实现 Deep-sleep 模式的基本功能，周期性唤醒 ESP 模块，以从 NTP 服务器获取时间。
- system/deep_sleep 演示如何使用 Deep-sleep 唤醒触发器和 ULP 协处理器编程。
- system/light_sleep 演示如何使用 ESP32-P4 的唤醒源，如定时器，GPIO，触摸传感器等，触发 Light-sleep 唤醒。

API 参考

Header File

- components/esp_hw_support/include/esp_sleep.h
- This header file can be included with:

```
#include "esp_sleep.h"
```

Functions

esp_err_t **esp_sleep_disable_wakeup_source**(*esp_sleep_source_t* source)

Disable wakeup source.

This function is used to deactivate wake up trigger for source defined as parameter of the function.

See docs/sleep-modes.rst for details.

备 注: This function does not modify wake up configuration in RTC. It will be performed in esp_deep_sleep_start/esp_light_sleep_start function.

参数 **source** -- - number of source to disable of type esp_sleep_source_t

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if trigger was not active

esp_err_t **esp_sleep_enable_ulp_wakeup**(void)

Enable wakeup by ULP coprocessor.

备注: On ESP32, ULP wakeup source cannot be used when RTC_PERIPH power domain is forced, to be powered on (ESP_PD_OPTION_ON) or when ext0 wakeup source is used.

返回

- ESP_OK on success

- ESP_ERR_NOT_SUPPORTED if additional current by touch (CONFIG_RTC_EXT_CRYST_ADDIT_CURRENT) is enabled.
- ESP_ERR_INVALID_STATE if ULP co-processor is not enabled or if wakeup triggers conflict

esp_err_t **esp_sleep_enable_timer_wakeup**(uint64_t time_in_us)

Enable wakeup by timer.

参数 **time_in_us** -- time before wakeup, in microseconds

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if value is out of range (TBD)

esp_err_t **esp_sleep_enable_touchpad_wakeup**(void)

Enable wakeup by touch sensor.

备注: On ESP32, touch wakeup source can not be used when RTC_PERIPH power domain is forced to be powered on (ESP_PD_OPTION_ON) or when ext0 wakeup source is used.

备注: The FSM mode of the touch button should be configured as the timer trigger mode.

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if additional current by touch (CONFIG_RTC_EXT_CRYST_ADDIT_CURRENT) is enabled.
- ESP_ERR_INVALID_STATE if wakeup triggers conflict

touch_pad_t **esp_sleep_get_touchpad_wakeup_status**(void)

Get the touch pad which caused wakeup.

If wakeup was caused by another source, this function will return TOUCH_PAD_MAX;

返回 touch pad which caused wakeup

bool **esp_sleep_is_valid_wakeup_gpio**(gpio_num_t gpio_num)

Returns true if a GPIO number is valid for use as wakeup source.

备注: For SoCs with RTC IO capability, this can be any valid RTC IO input pin.

参数 **gpio_num** -- Number of the GPIO to test for wakeup source capability

返回 True if this GPIO number will be accepted as a sleep wakeup source.

esp_err_t **esp_sleep_enable_ext1_wakeup**(uint64_t io_mask, *esp_sleep_ext1_wakeup_mode_t* level_mode)

Enable wakeup using multiple pins.

This function uses external wakeup feature of RTC controller. It will work even if RTC peripherals are shut down during sleep.

This feature can monitor any number of pins which are in RTC IOs. Once selected pins go into the state given by level_mode argument, the chip will be woken up.

备 注: This function does not modify pin configuration. The pins are configured in esp_deep_sleep_start/esp_light_sleep_start, immediately before entering sleep mode.

备注： Internal pullups and pulldowns don't work when RTC peripherals are shut down. In this case, external resistors need to be added. Alternatively, RTC peripherals (and pullups/pulldowns) may be kept enabled using esp_sleep_pd_config function. If we turn off the `RTC_PERIPH` domain or certain chips lack the `RTC_PERIPH` domain, we will use the HOLD feature to maintain the pull-up and pull-down on the pins during sleep. HOLD feature will be acted on the pin internally before the system entering sleep, and this can further reduce power consumption.

备注： Call this func will reset the previous ext1 configuration.

备 注： This function will be deprecated in release/v6.0. Please switch to use `esp_sleep_enable_ext1_wakeup_io` and `esp_sleep_disable_ext1_wakeup_io`

参数

- **`io_mask`** -- Bit mask of GPIO numbers which will cause wakeup. Only GPIOs which have RTC functionality can be used in this bit map. For different SoCs, the related GPIOs are:
  - ESP32: 0, 2, 4, 12-15, 25-27, 32-39
  - ESP32-S2: 0-21
  - ESP32-S3: 0-21
  - ESP32-C6: 0-7
  - ESP32-H2: 7-14
- **`level_mode`** -- Select logic function used to determine wakeup condition: When target chip is ESP32:
  - ESP_EXT1_WAKEUP_ALL_LOW: wake up when all selected GPIOs are low
  - ESP_EXT1_WAKEUP_ANY_HIGH: wake up when any of the selected GPIOs is high When target chip is ESP32-S2, ESP32-S3, ESP32-C6 or ESP32-H2:
  - ESP_EXT1_WAKEUP_ANY_LOW: wake up when any of the selected GPIOs is low
  - ESP_EXT1_WAKEUP_ANY_HIGH: wake up when any of the selected GPIOs is high

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if io_mask is zero,, or mode is invalid

esp_err_t **esp_sleep_enable_ext1_wakeup_io**(uint64_t io_mask, *esp_sleep_ext1_wakeup_mode_t* level_mode)

Enable ext1 wakeup pins with IO masks.

This will append selected IOs to the wakeup IOs, it will not reset previously enabled IOs. To reset specific previously enabled IOs, call esp_sleep_disable_ext1_wakeup_io with the io_mask. To reset all the enabled IOs, call esp_sleep_disable_ext1_wakeup_io(0).

This function uses external wakeup feature of RTC controller. It will work even if RTC peripherals are shut down during sleep.

This feature can monitor any number of pins which are in RTC IOs. Once selected pins go into the state given by level_mode argument, the chip will be woken up.

备 注： This function does not modify pin configuration. The pins are configured in esp_deep_sleep_start/esp_light_sleep_start, immediately before entering sleep mode.

备注： Internal pullups and pulldowns don't work when RTC peripherals are shut down. In this case, external resistors need to be added. Alternatively, RTC peripherals (and pullups/pulldowns) may be kept enabled using esp_sleep_pd_config function. If we turn off the `RTC_PERIPH` domain or certain chips lack

the RTC_PERIPH domain, we will use the HOLD feature to maintain the pull-up and pull-down on the pins during sleep. HOLD feature will be acted on the pin internally before the system entering sleep, and this can further reduce power consumption.

参数

- **io_mask** -- Bit mask of GPIO numbers which will cause wakeup. Only GPIOs which have RTC functionality can be used in this bit map. For different SoCs, the related GPIOs are:
  - ESP32: 0, 2, 4, 12-15, 25-27, 32-39
  - ESP32-S2: 0-21
  - ESP32-S3: 0-21
  - ESP32-C6: 0-7
  - ESP32-H2: 7-14
- **level_mode** -- Select logic function used to determine wakeup condition: When target chip is ESP32:
  - ESP_EXT1_WAKEUP_ALL_LOW: wake up when all selected GPIOs are low
  - ESP_EXT1_WAKEUP_ANY_HIGH: wake up when any of the selected GPIOs is high When target chip is ESP32-S2, ESP32-S3, ESP32-C6 or ESP32-H2:
  - ESP_EXT1_WAKEUP_ANY_LOW: wake up when any of the selected GPIOs is low
  - ESP_EXT1_WAKEUP_ANY_HIGH: wake up when any of the selected GPIOs is high

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if any of the selected GPIOs is not an RTC GPIO, or mode is invalid
- ESP_ERR_NOT_ALLOWED when wakeup level will become different between ext1 IOs if !SOC_PM_SUPPORT_EXT1_WAKEUP_MODE_PER_PIN

esp_err_t **esp_sleep_disable_ext1_wakeup_io**(uint64_t io_mask)

Disable ext1 wakeup pins with IO masks. This will remove selected IOs from the wakeup IOs.

参数 **io_mask** -- Bit mask of GPIO numbers which will cause wakeup. Only GPIOs which have RTC functionality can be used in this bit map. If value is zero, this func will remove all previous ext1 configuration. For different SoCs, the related GPIOs are:

- ESP32: 0, 2, 4, 12-15, 25-27, 32-39
- ESP32-S2: 0-21
- ESP32-S3: 0-21
- ESP32-C6: 0-7
- ESP32-H2: 7-14

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if any of the selected GPIOs is not an RTC GPIO.

esp_err_t **esp_sleep_enable_ext1_wakeup_with_level_mask**(uint64_t io_mask, uint64_t level_mask)

Enable wakeup using multiple pins, allows different trigger mode per pin.

This function uses external wakeup feature of RTC controller. It will work even if RTC peripherals are shut down during sleep.

This feature can monitor any number of pins which are in RTC IOs. Once selected pins go into the state given by level_mode argument, the chip will be woken up.

备 注: This function does not modify pin configuration. The pins are configured in esp_deep_sleep_start/esp_light_sleep_start, immediately before entering sleep mode.

备注: Internal pullups and pulldowns don't work when RTC peripherals are shut down. In this case, external resistors need to be added. Alternatively, RTC peripherals (and pullups/pulldowns) may be kept en-

abled using esp_sleep_pd_config function. If we turn off the RTC_PERIPH domain or certain chips lack the RTC_PERIPH domain, we will use the HOLD feature to maintain the pull-up and pull-down on the pins during sleep. HOLD feature will be acted on the pin internally before the system entering sleep, and this can further reduce power consumption.

参数

- **io_mask** -- Bit mask of GPIO numbers which will cause wakeup. Only GPIOs which have RTC functionality can be used in this bit map. For different SoCs, the related GPIOs are:
  - ESP32-C6: 0-7.
  - ESP32-H2: 7-14.
- **level_mask** -- Select logic function used to determine wakeup condition per pin. Each bit of the level_mask corresponds to the respective GPIO. Each bit's corresponding position is set to 0, the wakeup level will be low, on the contrary, each bit's corresponding position is set to 1, the wakeup level will be high.

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if any of the selected GPIOs is not an RTC GPIO, or mode is invalid

esp_err_t **esp_deep_sleep_enable_gpio_wakeup**(uint64_t gpio_pin_mask, *esp_deepsleep_gpio_wake_up_mode_t* mode)

Enable wakeup using specific gpio pins.

This function enables an IO pin to wake up the chip from deep sleep.

备注: This function does not modify pin configuration. The pins are configured inside esp_deep_sleep_start, immediately before entering sleep mode.

备注: You don't need to worry about pull-up or pull-down resistors before using this function because the ESP_SLEEP_GPIO_ENABLE_INTERNAL_RESISTORS option is enabled by default. It will automatically set pull-up or pull-down resistors internally in esp_deep_sleep_start based on the wakeup mode. However, when using external pull-up or pull-down resistors, please be sure to disable the ESP_SLEEP_GPIO_ENABLE_INTERNAL_RESISTORS option, as the combination of internal and external resistors may cause interference. BTW, when you use low level to wake up the chip, we strongly recommend you to add external resistors (pull-up).

参数

- **gpio_pin_mask** -- Bit mask of GPIO numbers which will cause wakeup. Only GPIOs which have RTC functionality (pads that powered by VDD3P3_RTC) can be used in this bit map.
- **mode** -- Select logic function used to determine wakeup condition:
  - ESP_GPIO_WAKEUP_GPIO_LOW: wake up when the gpio turn to low.
  - ESP_GPIO_WAKEUP_GPIO_HIGH: wake up when the gpio turn to high.

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the mask contains any invalid deep sleep wakeup pin or wakeup mode is invalid

esp_err_t **esp_sleep_enable_gpio_wakeup**(void)

Enable wakeup from light sleep using GPIOs.

Each GPIO supports wakeup function, which can be triggered on either low level or high level. Unlike EXT0 and EXT1 wakeup sources, this method can be used both for all IOs: RTC IOs and digital IOs. It can only be used to wakeup from light sleep though.

To enable wakeup, first call gpio_wakeup_enable, specifying gpio number and wakeup level, for each GPIO which is used for wakeup. Then call this function to enable wakeup feature.

备注： On ESP32, GPIO wakeup source can not be used together with touch or ULP wakeup sources.

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if wakeup triggers conflict

esp_err_t **esp_sleep_enable_uart_wakeup**(int uart_num)

Enable wakeup from light sleep using UART.

Use uart_set_wakeup_threshold function to configure UART wakeup threshold.

Wakeup from light sleep takes some time, so not every character sent to the UART can be received by the application.

备注： ESP32 does not support wakeup from UART2.

参数 **uart_num** -- UART port to wake up from

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if wakeup from given UART is not supported

esp_err_t **esp_sleep_enable_bt_wakeup**(void)

Enable wakeup by bluetooth.

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if wakeup from bluetooth is not supported

esp_err_t **esp_sleep_disable_bt_wakeup**(void)

Disable wakeup by bluetooth.

返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if wakeup from bluetooth is not supported

esp_err_t **esp_sleep_enable_wifi_wakeup**(void)

Enable wakeup by WiFi MAC.

返回

- ESP_OK on success

esp_err_t **esp_sleep_disable_wifi_wakeup**(void)

Disable wakeup by WiFi MAC.

返回

- ESP_OK on success

esp_err_t **esp_sleep_enable_wifi_beacon_wakeup**(void)

Enable beacon wakeup by WiFi MAC, it will wake up the system into modem state.

返回

- ESP_OK on success

esp_err_t **esp_sleep_disable_wifi_beacon_wakeup**(void)

Disable beacon wakeup by WiFi MAC.

返回

- ESP_OK on success

uint64_t **esp_sleep_get_ext1_wakeup_status**(void)

Get the bit mask of GPIOs which caused wakeup (ext1)

If wakeup was caused by another source, this function will return 0.

返回 bit mask, if GPIOn caused wakeup, BIT(n) will be set

uint64_t **esp_sleep_get_gpio_wakeup_status**(void)

Get the bit mask of GPIOs which caused wakeup (gpio)

If wakeup was caused by another source, this function will return 0.

返回 bit mask, if GPIOn caused wakeup, BIT(n) will be set

esp_err_t **esp_sleep_pd_config**(*esp_sleep_pd_domain_t* domain, *esp_sleep_pd_option_t* option)

Set power down mode for an RTC power domain in sleep mode.

If not set set using this API, all power domains default to ESP_PD_OPTION_AUTO.

参数

- **domain** -- power domain to configure
- **option** -- power down option (ESP_PD_OPTION_OFF, ESP_PD_OPTION_ON, or ESP_PD_OPTION_AUTO)

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if either of the arguments is out of range

esp_err_t **esp_deep_sleep_try_to_start**(void)

Enter deep sleep with the configured wakeup options.

The reason for the rejection can be such as a short sleep time.

备注: In general, the function does not return, but if the sleep is rejected, then it returns from it.

返回

- No return - If the sleep is not rejected.
- ESP_ERR_SLEEP_REJECT sleep request is rejected(wakeup source set before the sleep request)

void **esp_deep_sleep_start**(void)

Enter deep sleep with the configured wakeup options.

备注: The function does not do a return (no rejection). Even if wakeup source set before the sleep request it goes to deep sleep anyway.

esp_err_t **esp_light_sleep_start**(void)

Enter light sleep with the configured wakeup options.

返回

- ESP_OK on success (returned after wakeup)
- ESP_ERR_SLEEP_REJECT sleep request is rejected(wakeup source set before the sleep request)
- ESP_ERR_SLEEP_TOO_SHORT_SLEEP_DURATION after deducting the sleep flow overhead, the final sleep duration is too short to cover the minimum sleep duration of the chip, when rtc timer wakeup source enabled

esp_err_t **esp_deep_sleep_try**(uint64_t time_in_us)

Enter deep-sleep mode.

The device will automatically wake up after the deep-sleep time Upon waking up, the device calls deep sleep wake stub, and then proceeds to load application.

Call to this function is equivalent to a call to esp_deep_sleep_enable_timer_wakeup followed by a call to esp_deep_sleep_start.

The reason for the rejection can be such as a short sleep time.

备注: In general, the function does not return, but if the sleep is rejected, then it returns from it.

参数 **time_in_us** -- deep-sleep time, unit: microsecond

返回

- No return - If the sleep is not rejected.
- ESP_ERR_SLEEP_REJECT sleep request is rejected(wakeup source set before the sleep request)

void **esp_deep_sleep**(uint64_t time_in_us)

Enter deep-sleep mode.

The device will automatically wake up after the deep-sleep time Upon waking up, the device calls deep sleep wake stub, and then proceeds to load application.

Call to this function is equivalent to a call to esp_deep_sleep_enable_timer_wakeup followed by a call to esp_deep_sleep_start.

备注: The function does not do a return (no rejection).. Even if wakeup source set before the sleep request it goes to deep sleep anyway.

参数 **time_in_us** -- deep-sleep time, unit: microsecond

esp_err_t **esp_deep_sleep_register_hook**(*esp_deep_sleep_cb_t* new_dslp_cb)

Register a callback to be called from the deep sleep prepare.

警告: deepsleep callbacks should without parameters, and MUST NOT, UNDER ANY CIRCUMSTANCES, CALL A FUNCTION THAT MIGHT BLOCK.

参数 **new_dslp_cb** -- Callback to be called

返回

- ESP_OK: Callback registered to the deepsleep misc_modules_sleep_prepare
- ESP_ERR_NO_MEM: No more hook space for register the callback

void **esp_deep_sleep_deregister_hook**(*esp_deep_sleep_cb_t* old_dslp_cb)

Unregister an deepsleep callback.

参数 **old_dslp_cb** -- Callback to be unregistered

esp_sleep_wakeup_cause_t **esp_sleep_get_wakeup_cause**(void)

Get the wakeup source which caused wakeup from sleep.

返回 cause of wake up from last sleep (deep sleep or light sleep)

void **esp_wake_deep_sleep**(void)

Default stub to run on wake from deep sleep.

Allows for executing code immediately on wake from sleep, before the software bootloader or ESP-IDF app has started up.

This function is weak-linked, so you can implement your own version to run code immediately when the chip wakes from sleep.

See docs/deep-sleep-stub.rst for details.

void **esp_set_deep_sleep_wake_stub**(*esp_deep_sleep_wake_stub_fn_t* new_stub)

Install a new stub at runtime to run on wake from deep sleep.

If implementing esp_wake_deep_sleep() then it is not necessary to call this function.

However, it is possible to call this function to substitute a different deep sleep stub. Any function used as a deep sleep stub must be marked RTC_IRAM_ATTR, and must obey the same rules given for esp_wake_deep_sleep().

void **esp_set_deep_sleep_wake_stub_default_entry**(void)

Set wake stub entry to default `esp_wake_stub_entry`

esp_deep_sleep_wake_stub_fn_t **esp_get_deep_sleep_wake_stub**(void)

Get current wake from deep sleep stub.

返回 Return current wake from deep sleep stub, or NULL if no stub is installed.

void **esp_default_wake_deep_sleep**(void)

The default esp-idf-provided esp_wake_deep_sleep() stub.

See docs/deep-sleep-stub.rst for details.

void **esp_deep_sleep_disable_rom_logging**(void)

Disable logging from the ROM code after deep sleep.

Using LSB of RTC_STORE4.

void **esp_sleep_config_gpio_isolate**(void)

Configure to isolate all GPIO pins in sleep state.

void **esp_sleep_enable_gpio_switch**(bool enable)

Enable or disable GPIO pins status switching between slept status and waked status.

参数 **enable** -- decide whether to switch status or not

Macros

ESP_PD_DOMAIN_RTC8M

ESP_SLEEP_POWER_DOWN_CPU

Type Definitions

typedef void (***esp_deep_sleep_cb_t**)(void)

typedef *esp_sleep_source_t* **esp_sleep_wakeup_cause_t**

typedef void (***esp_deep_sleep_wake_stub_fn_t**)(void)

Function type for stub to run on wake from sleep.

Enumerations

enum **esp_sleep_ext1_wakeup_mode_t**

Logic function used for EXT1 wakeup mode.

Values:

enumerator **ESP_EXT1_WAKEUP_ANY_LOW**

Wake the chip when any of the selected GPIOs go low.

enumerator **ESP_EXT1_WAKEUP_ANY_HIGH**

Wake the chip when any of the selected GPIOs go high.

enumerator **ESP_EXT1_WAKEUP_ALL_LOW**

enum **esp_deepsleep_gpio_wake_up_mode_t**

Values:

enumerator **ESP_GPIO_WAKEUP_GPIO_LOW**

enumerator **ESP_GPIO_WAKEUP_GPIO_HIGH**

enum **esp_sleep_pd_domain_t**

Power domains which can be powered down in sleep mode.

Values:

enumerator **ESP_PD_DOMAIN_RTC_PERIPH**

RTC IO, sensors and ULP co-processor.

enumerator **ESP_PD_DOMAIN_XTAL**

XTAL oscillator.

enumerator **ESP_PD_DOMAIN_XTAL32K**

External 32 kHz XTAL oscillator.

enumerator **ESP_PD_DOMAIN_RC32K**

Internal 32 kHz RC oscillator.

enumerator **ESP_PD_DOMAIN_RC_FAST**

Internal Fast oscillator.

enumerator **ESP_PD_DOMAIN_VDDSDIO**

VDD_SDIO.

enumerator **ESP_PD_DOMAIN_TOP**

SoC TOP.

enumerator **ESP_PD_DOMAIN_CNNT**

Hight-speed connect peripherals power domain.

enumerator **ESP_PD_DOMAIN_MAX**

Number of domains.

enum **esp_sleep_pd_option_t**

Power down options.

Values:

enumerator **ESP_PD_OPTION_OFF**

Power down the power domain in sleep mode.

enumerator **ESP_PD_OPTION_ON**

Keep power domain enabled during sleep mode.

enumerator **ESP_PD_OPTION_AUTO**

Keep power domain enabled in sleep mode, if it is needed by one of the wakeup options. Otherwise power it down.

enum **esp_sleep_source_t**

Sleep wakeup cause.

Values:

enumerator **ESP_SLEEP_WAKEUP_UNDEFINED**

In case of deep sleep, reset was not caused by exit from deep sleep.

enumerator **ESP_SLEEP_WAKEUP_ALL**

Not a wakeup cause, used to disable all wakeup sources with esp_sleep_disable_wakeup_source.

enumerator **ESP_SLEEP_WAKEUP_EXT0**

Wakeup caused by external signal using RTC_IO.

enumerator **ESP_SLEEP_WAKEUP_EXT1**

Wakeup caused by external signal using RTC_CNTL.

enumerator **ESP_SLEEP_WAKEUP_TIMER**

Wakeup caused by timer.

enumerator **ESP_SLEEP_WAKEUP_TOUCHPAD**

Wakeup caused by touchpad.

enumerator **ESP_SLEEP_WAKEUP_ULP**

Wakeup caused by ULP program.

enumerator **ESP_SLEEP_WAKEUP_GPIO**

Wakeup caused by GPIO (light sleep only on ESP32, S2 and S3)

enumerator **ESP_SLEEP_WAKEUP_UART**

Wakeup caused by UART (light sleep only)

enumerator **ESP_SLEEP_WAKEUP_WIFI**

Wakeup caused by WIFI (light sleep only)

enumerator **ESP_SLEEP_WAKEUP_COCPU**

Wakeup caused by COCPU int.

enumerator **ESP_SLEEP_WAKEUP_COCPU_TRAP_TRIG**

Wakeup caused by COCPU crash.

enumerator **ESP_SLEEP_WAKEUP_BT**

Wakeup caused by BT (light sleep only)

enum **esp_sleep_mode_t**

Sleep mode.

Values:

enumerator **ESP_SLEEP_MODE_LIGHT_SLEEP**

light sleep mode

enumerator **ESP_SLEEP_MODE_DEEP_SLEEP**

deep sleep mode

enum **[anonymous]**

Values:

enumerator **ESP_ERR_SLEEP_REJECT**

enumerator **ESP_ERR_SLEEP_TOO_SHORT_SLEEP_DURATION**

2.9.29 SoC 功能

此文档介绍了 ESP32-P4 SoC 硬件功能的宏定义。ESP-IDF 中的条件编译指令通常使用这些宏来确定哪些依赖于硬件的功能受到支持，从而控制需编译的代码内容。

备注： 目前，这些宏定义不属于公共 API，未来可能发生重大更改。如需了解详情，请前往*ESP-IDF 版本简介*。

API 参考

Header File

- components/soc/esp32p4/include/soc/soc_caps.h
- This header file can be included with:

```
#include "soc/soc_caps.h"
```

Macros

SOC_ADC_SUPPORTED

SOC_ANA_CMPR_SUPPORTED

SOC_DEDICATED_GPIO_SUPPORTED

SOC_UART_SUPPORTED

SOC_GDMA_SUPPORTED

SOC_AHB_GDMA_SUPPORTED

SOC_AXI_GDMA_SUPPORTED

SOC_DW_GDMA_SUPPORTED

SOC_DMA2D_SUPPORTED

SOC_GPTIMER_SUPPORTED

SOC_PCNT_SUPPORTED

SOC_LCDCAM_SUPPORTED

SOC_LCDCAM_CAM_SUPPORTED

SOC_LCDCAM_I80_LCD_SUPPORTED

SOC_MIPI_CSI_SUPPORTED

SOC_MIPI_DSI_SUPPORTED

SOC_MCPWM_SUPPORTED

SOC_TWAI_SUPPORTED

SOC_ETM_SUPPORTED

SOC_PARLIO_SUPPORTED

SOC_ASYNC_MEMCPY_SUPPORTED

SOC_EMAC_SUPPORTED

SOC_USB_OTG_SUPPORTED

SOC_WIRELESS_HOST_SUPPORTED

SOC_USB_SERIAL_JTAG_SUPPORTED

SOC_TEMP_SENSOR_SUPPORTED

SOC_SUPPORTS_SECURE_DL_MODE

SOC_ULP_SUPPORTED

SOC_LP_CORE_SUPPORTED

SOC_EFUSE_KEY_PURPOSE_FIELD

SOC_EFUSE_SUPPORTED

SOC_RTC_FAST_MEM_SUPPORTED

SOC_RTC_MEM_SUPPORTED

SOC_RMT_SUPPORTED

SOC_I2S_SUPPORTED

SOC_SDM_SUPPORTED

SOC_GPSPI_SUPPORTED

SOC_LEDC_SUPPORTED

SOC_ISP_SUPPORTED

SOC_I2C_SUPPORTED

SOC_SYSTIMER_SUPPORTED

SOC_AES_SUPPORTED

SOC_MPI_SUPPORTED

SOC_SHA_SUPPORTED

SOC_HMAC_SUPPORTED

SOC_DIG_SIGN_SUPPORTED

SOC_ECC_SUPPORTED

SOC_ECC_EXTENDED_MODES_SUPPORTED

SOC_ECDSA_SUPPORTED

SOC_KEY_MANAGER_SUPPORTED

SOC_FLASH_ENC_SUPPORTED

SOC_SECURE_BOOT_SUPPORTED

SOC_BOD_SUPPORTED

SOC_APM_SUPPORTED

SOC_PMU_SUPPORTED

SOC_DCDC_SUPPORTED

SOC_PAU_SUPPORTED

SOC_LP_TIMER_SUPPORTED

SOC_ULP_LP_UART_SUPPORTED

SOC_LP_GPIO_MATRIX_SUPPORTED

SOC_LP_PERIPHERALS_SUPPORTED

SOC_LP_I2C_SUPPORTED

SOC_LP_SPI_SUPPORTED

SOC_SPIRAM_SUPPORTED

SOC_PSRAM_DMA_CAPABLE

SOC_SDMMC_HOST_SUPPORTED

SOC_CLK_TREE_SUPPORTED

SOC_ASSIST_DEBUG_SUPPORTED

SOC_WDT_SUPPORTED

SOC_SPI_FLASH_SUPPORTED

SOC_TOUCH_SENSOR_SUPPORTED

SOC_RNG_SUPPORTED

SOC_GP_LDO_SUPPORTED

SOC_PPA_SUPPORTED

SOC_LIGHT_SLEEP_SUPPORTED

SOC_DEEP_SLEEP_SUPPORTED

SOC_PM_SUPPORTED

SOC_XTAL_SUPPORT_40M

SOC_AES_SUPPORT_DMA

SOC_AES_SUPPORT_GCM

SOC_AES_GDMA

SOC_AES_SUPPORT_AES_128

SOC_AES_SUPPORT_AES_256

SOC_ADC_RTC_CTRL_SUPPORTED
< SAR ADC Module

SOC_ADC_DIG_CTRL_SUPPORTED

SOC_ADC_DIG_SUPPORTED_UNIT(UNIT)

SOC_ADC_DMA_SUPPORTED

SOC_ADC_PERIPH_NUM

SOC_ADC_CHANNEL_NUM(PERIPH_NUM)

SOC_ADC_MAX_CHANNEL_NUM

SOC_ADC_ATTEN_NUM
Digital

SOC_ADC_DIGI_CONTROLLER_NUM

SOC_ADC_PATT_LEN_MAX

Four pattern tables, each contains 4 items. Each item takes 1 byte

SOC_ADC_DIGI_MAX_BITWIDTH

SOC_ADC_DIGI_MIN_BITWIDTH

SOC_ADC_DIGI_IIR_FILTER_NUM

SOC_ADC_DIGI_MONITOR_NUM

SOC_ADC_DIGI_RESULT_BYTES

SOC_ADC_DIGI_DATA_BYTES_PER_CONV

F_sample = F_digi_con / 2 / interval. F_digi_con = 5M for now. 30 <= interval <= 4095

SOC_ADC_SAMPLE_FREQ_THRES_HIGH

SOC_ADC_SAMPLE_FREQ_THRES_LOW

RTC

SOC_ADC_RTC_MIN_BITWIDTH

SOC_ADC_RTC_MAX_BITWIDTH

Calibration

SOC_ADC_CALIBRATION_V1_SUPPORTED

support HW offset calibration version 1 ADC power control is shared by PWDET, TempSensor

SOC_ADC_SHARED_POWER

SOC_APB_BACKUP_DMA

SOC_BROWNOUT_RESET_SUPPORTED

SOC_SHARED_IDCACHE_SUPPORTED

SOC_CACHE_WRITEBACK_SUPPORTED

SOC_CACHE_FREEZE_SUPPORTED

SOC_CACHE_INTERNAL_MEM_VIA_L1CACHE

SOC_CPU_CORES_NUM

SOC_CPU_INTR_NUM

SOC_CPU_HAS_FLEXIBLE_INTC

SOC_INT_CLIC_SUPPORTED

SOC_INT_HW_NESTED_SUPPORTED

SOC_BRANCH_PREDICTOR_SUPPORTED

SOC_CPU_COPROC_NUM

SOC_CPU_HAS_FPU

SOC_CPU_HAS_FPU_EXT_ILL_BUG

SOC_CPU_HAS_HWLOOP

SOC_CPU_HAS_PIE

SOC_HP_CPU_HAS_MULTIPLE_CORES

SOC_CPU_BREAKPOINTS_NUM

SOC_CPU_WATCHPOINTS_NUM

SOC_CPU_WATCHPOINT_MAX_REGION_SIZE

SOC_CPU_HAS_PMA

SOC_CPU_IDRAM_SPLIT_USING_PMP

SOC_CPU_PMP_REGION_GRANULARITY

SOC_CPU_HAS_LOCKUP_RESET

SOC_DS_SIGNATURE_MAX_BIT_LEN

The maximum length of a Digital Signature in bits.

SOC_DS_KEY_PARAM_MD_IV_LENGTH

Initialization vector (IV) length for the RSA key parameter message digest (MD) in bytes.

SOC_DS_KEY_CHECK_MAX_WAIT_US

Maximum wait time for DS parameter decryption key. If overdue, then key error. See TRM DS chapter for more details

SOC_DMA_CAN_ACCESS_FLASH

DMA can access Flash memory

SOC_AHB_GDMA_VERSION

SOC_GDMA_SUPPORT_CRC

SOC_GDMA_NUM_GROUPS_MAX

SOC_GDMA_PAIRS_PER_GROUP_MAX

SOC_AXI_GDMA_SUPPORT_PSRAM

SOC_GDMA_SUPPORT_ETM

SOC_AXI_DMA_EXT_MEM_ENC_ALIGNMENT

SOC_DMA2D_GROUPS

SOC_DMA2D_TX_CHANNELS_PER_GROUP

SOC_DMA2D_RX_CHANNELS_PER_GROUP

SOC_ETM_GROUPS

SOC_ETM_CHANNELS_PER_GROUP

SOC_GPIO_PORT

SOC_GPIO_PIN_COUNT

SOC_GPIO_SUPPORT_PIN_GLITCH_FILTER

SOC_GPIO_FLEX_GLITCH_FILTER_NUM

SOC_GPIO_SUPPORT_PIN_HYS_FILTER

SOC_GPIO_SUPPORT_ETM

SOC_GPIO_SUPPORT_RTC_INDEPENDENT

SOC_GPIO_SUPPORT_DEEPSLEEP_WAKEUP

SOC_LP_IO_HAS_INDEPENDENT_WAKEUP_SOURCE

SOC_GPIO_VALID_GPIO_MASK

SOC_GPIO_VALID_OUTPUT_GPIO_MASK

SOC_GPIO_IN_RANGE_MAX

SOC_GPIO_OUT_RANGE_MAX

SOC_GPIO_DEEP_SLEEP_WAKE_VALID_GPIO_MASK

SOC_GPIO_DEEP_SLEEP_WAKE_SUPPORTED_PIN_CNT

SOC_GPIO_VALID_DIGITAL_IO_PAD_MASK

SOC_GPIO_CLOCKOUT_BY_GPIO_MATRIX

SOC_GPIO_CLOCKOUT_CHANNEL_NUM

SOC_CLOCKOUT_SUPPORT_CHANNEL_DIVIDER

SOC_GPIO_SUPPORT_FORCE_HOLD

SOC_RTCIO_PIN_COUNT

SOC_RTCIO_INPUT_OUTPUT_SUPPORTED

SOC_RTCIO_HOLD_SUPPORTED

SOC_RTCIO_WAKE_SUPPORTED

SOC_DEDIC_GPIO_OUT_CHANNELS_NUM

8 outward channels on each CPU core

SOC_DEDIC_GPIO_IN_CHANNELS_NUM

8 inward channels on each CPU core

SOC_DEDIC_PERIPH_ALWAYS_ENABLE

The dedicated GPIO (a.k.a. fast GPIO) is featured by some customized CPU instructions, which is always enabled

SOC_ANA_CMPR_NUM

SOC_ANA_CMPR_CAN_DISTINGUISH_EDGE

SOC_ANA_CMPR_SUPPORT_ETM

SOC_I2C_NUM

SOC_HP_I2C_NUM

SOC_I2C_FIFO_LEN
I2C hardware FIFO depth

SOC_I2C_CMD_REG_NUM
Number of I2C command registers

SOC_I2C_SUPPORT_SLAVE

SOC_I2C_SUPPORT_HW_FSM_RST

SOC_I2C_SUPPORT_HW_CLR_BUS

SOC_I2C_SUPPORT_XTAL

SOC_I2C_SUPPORT_RTC

SOC_I2C_SUPPORT_10BIT_ADDR

SOC_I2C_SLAVE_SUPPORT_BROADCAST

SOC_I2C_SLAVE_SUPPORT_I2CRAM_ACCESS

SOC_I2C_SLAVE_SUPPORT_SLAVE_UNMATCH

SOC_LP_I2C_NUM

SOC_LP_I2C_FIFO_LEN
LP_I2C hardware FIFO depth

SOC_I2S_NUM

SOC_I2S_HW_VERSION_2

SOC_I2S_SUPPORTS_XTAL

SOC_I2S_SUPPORTS_APLL

SOC_I2S_SUPPORTS_PCM

SOC_I2S_SUPPORTS_PDM

SOC_I2S_SUPPORTS_PDM_TX

SOC_I2S_SUPPORTS_PDM_RX

SOC_I2S_SUPPORTS_PDM_RX_HP_FILTER

SOC_I2S_SUPPORTS_TX_SYNC_CNT

SOC_I2S_SUPPORTS_TDM

SOC_I2S_PDM_MAX_TX_LINES

SOC_I2S_PDM_MAX_RX_LINES

SOC_I2S_TDM_FULL_DATA_WIDTH

No limitation to data bit width when using multiple slots

SOC_ISP_BF_SUPPORTED

SOC_ISP_CCM_SUPPORTED

SOC_ISP_DVP_SUPPORTED

SOC_ISP_NUMS

SOC_ISP_DVP_CTLR_NUMS

SOC_ISP_AF_CTLR_NUMS

SOC_ISP_AF_WINDOW_NUMS

SOC_ISP_SHARE_CSI_BRG

SOC_ISP_BF_TEMPLATE_X_NUMS

SOC_ISP_BF_TEMPLATE_Y_NUMS

SOC_ISP_CCM_DIMENSION

SOC_ISP_DVP_DATA_WIDTH_MAX

SOC_LEDC_SUPPORT_PLL_DIV_CLOCK

SOC_LEDC_SUPPORT_XTAL_CLOCK

SOC_LEDC_CHANNEL_NUM

SOC_LEDC_TIMER_BIT_WIDTH

SOC_LEDC_GAMMA_CURVE_FADE_SUPPORTED

SOC_LEDC_GAMMA_CURVE_FADE_RANGE_MAX

SOC_LEDC_SUPPORT_FADE_STOP

SOC_LEDC_FADE_PARAMS_BIT_WIDTH

SOC_MMU_PAGE_SIZE_CONFIGURABLE

SOC_MMU_PERIPH_NUM

SOC_MMU_LINEAR_ADDRESS_REGION_NUM

SOC_MMU_DI_VADDR_SHARED
D/I vaddr are shared

SOC_MMU_PER_EXT_MEM_TARGET
MMU is per physical external memory target (flash, psram)

SOC_MPU_CONFIGURABLE_REGIONS_SUPPORTED

SOC_MPU_MIN_REGION_SIZE

SOC_MPU_REGIONS_MAX_NUM

SOC_MPU_REGION_RO_SUPPORTED

SOC_MPU_REGION_WO_SUPPORTED

SOC_PCNT_GROUPS

SOC_PCNT_UNITS_PER_GROUP

SOC_PCNT_CHANNELS_PER_UNIT

SOC_PCNT_THRES_POINT_PER_UNIT

SOC_PCNT_SUPPORT_RUNTIME_THRES_UPDATE

SOC_PCNT_SUPPORT_CLEAR_SIGNAL
Support clear signal input

SOC_RMT_GROUPS
One RMT group

SOC_RMT_TX_CANDIDATES_PER_GROUP
Number of channels that capable of Transmit in each group

SOC_RMT_RX_CANDIDATES_PER_GROUP

Number of channels that capable of Receive in each group

SOC_RMT_CHANNELS_PER_GROUP

Total 8 channels

SOC_RMT_MEM_WORDS_PER_CHANNEL

Each channel owns 48 words memory (1 word = 4 Bytes)

SOC_RMT_SUPPORT_RX_PINGPONG

Support Ping-Pong mode on RX path

SOC_RMT_SUPPORT_RX_DEMODULATION

Support signal demodulation on RX path (i.e. remove carrier)

SOC_RMT_SUPPORT_TX_ASYNC_STOP

Support stop transmission asynchronously

SOC_RMT_SUPPORT_TX_LOOP_COUNT

Support transmit specified number of cycles in loop mode

SOC_RMT_SUPPORT_TX_LOOP_AUTO_STOP

Hardware support of auto-stop in loop mode

SOC_RMT_SUPPORT_TX_SYNCHRO

Support coordinate a group of TX channels to start simultaneously

SOC_RMT_SUPPORT_TX_CARRIER_DATA_ONLY

TX carrier can be modulated to data phase only

SOC_RMT_SUPPORT_XTAL

Support set XTAL clock as the RMT clock source

SOC_RMT_SUPPORT_DMA

RMT peripheral can connect to DMA channel

SOC_LCD_I80_SUPPORTED

support intel 8080 driver

SOC_LCDCAM_I80_NUM_BUSES

LCD_CAM peripheral provides one LCD Intel 8080 bus

SOC_LCDCAM_I80_BUS_WIDTH

Intel 8080 bus max data width

SOC_LCDCAM_RGB_NUM_PANELS

Support one RGB LCD panel

SOC_MCPWM_GROUPS

2 MCPWM groups on the chip (i.e., the number of independent MCPWM peripherals)

SOC_MCPWM_TIMERS_PER_GROUP

The number of timers that each group has.

SOC_MCPWM_OPERATORS_PER_GROUP

The number of operators that each group has.

SOC_MCPWM_COMPARATORS_PER_OPERATOR

The number of comparators that each operator has.

SOC_MCPWM_EVENT_COMPARATORS_PER_OPERATOR

The number of event comparators that each operator has.

SOC_MCPWM_GENERATORS_PER_OPERATOR

The number of generators that each operator has.

SOC_MCPWM_TRIGGERS_PER_OPERATOR

The number of triggers that each operator has.

SOC_MCPWM_GPIO_FAULTS_PER_GROUP

The number of fault signal detectors that each group has.

SOC_MCPWM_CAPTURE_TIMERS_PER_GROUP

The number of capture timers that each group has.

SOC_MCPWM_CAPTURE_CHANNELS_PER_TIMER

The number of capture channels that each capture timer has.

SOC_MCPWM_GPIO_SYNCHROS_PER_GROUP

The number of GPIO synchros that each group has.

SOC_MCPWM_SWSYNC_CAN_PROPAGATE

Software sync event can be routed to its output.

SOC_MCPWM_SUPPORT_ETM

Support ETM (Event Task Matrix)

SOC_MCPWM_SUPPORT_EVENT_COMPARATOR

Support event comparator (based on ETM)

SOC_MCPWM_CAPTURE_CLK_FROM_GROUP

Capture timer shares clock with other PWM timers.

SOC_PARLIO_GROUPS

Number of parallel IO peripherals

SOC_PARLIO_TX_UNITS_PER_GROUP

number of TX units in each group

SOC_PARLIO_RX_UNITS_PER_GROUP

number of RX units in each group

SOC_PARLIO_TX_UNIT_MAX_DATA_WIDTH

Number of data lines of the TX unit

SOC_PARLIO_RX_UNIT_MAX_DATA_WIDTH

Number of data lines of the RX unit

SOC_PARLIO_TX_CLK_SUPPORT_GATING

Support gating TX clock

SOC_PARLIO_RX_CLK_SUPPORT_GATING

Support gating RX clock

SOC_PARLIO_RX_CLK_SUPPORT_OUTPUT

Support output RX clock to a GPIO

SOC_PARLIO_TRANS_BIT_ALIGN

Support bit alignment in transaction

SOC_PARLIO_TX_SIZE_BY_DMA

Transaction length is controlled by DMA instead of indicated by register

SOC_MPI_MEM_BLOCKS_NUM

SOC_MPI_OPERATIONS_NUM

SOC_RSA_MAX_BIT_LEN

SOC_SDMMC_USE_IOMUX

Card detect, write protect, interrupt use GPIO Matrix on all chips. Slot 0 clock/cmd/data pins use IOMUX Slot 1 clock/cmd/data pins use GPIO Matrix

SOC_SDMMC_USE_GPIO_MATRIX

SOC_SDMMC_NUM_SLOTS

SOC_SDMMC_DELAY_PHASE_NUM

SOC_SDMMC_IO_POWER_EXTERNAL

SDMMC IO power controlled by external power supply.

SOC_SDMMC_PSRAM_DMA_CAPABLE

SDMMC peripheral can do DMA transfer to/from PSRAM.

SOC_SHA_DMA_MAX_BUFFER_SIZE

SOC_SHA_SUPPORT_DMA

SOC_SHA_SUPPORT_RESUME

SOC_SHA_GDMA

SOC_SHA_SUPPORT_SHA1

SOC_SHA_SUPPORT_SHA224

SOC_SHA_SUPPORT_SHA256

SOC_SHA_SUPPORT_SHA384

SOC_SHA_SUPPORT_SHA512

SOC_SHA_SUPPORT_SHA512_224

SOC_SHA_SUPPORT_SHA512_256

SOC_SHA_SUPPORT_SHA512_T

SOC_ECDSA_SUPPORT_EXPORT_PUBKEY

SOC_ECDSA_SUPPORT_DETERMINISTIC_MODE

SOC_ECDSA_USES_MPI

SOC_SDM_GROUPS

SOC_SDM_CHANNELS_PER_GROUP

SOC_SDM_CLK_SUPPORT_PLL_F80M

SOC_SDM_CLK_SUPPORT_XTAL

SOC_SPI_PERIPH_NUM

SOC_SPI_PERIPH_CS_NUM(i)

SOC_SPI_MAX_CS_NUM

SOC_SPI_MAXIMUM_BUFFER_SIZE

SOC_SPI_SUPPORT_SLAVE_HD_VER2

SOC_SPI_SLAVE_SUPPORT_SEG_TRANS

SOC_SPI_SUPPORT_DDRCLK

SOC_SPI_SUPPORT_CD_SIG

SOC_SPI_SUPPORT_OCT

SOC_SPI_SUPPORT_CLK_XTAL

SOC_SPI_SUPPORT_CLK_RC_FAST

SOC_SPI_SUPPORT_CLK_SPLL

SOC_SPI_PERIPH_SUPPORT_MULTILINE_MODE(host_id)

SOC_MEMSPI_IS_INDEPENDENT

SOC_SPI_MAX_PRE_DIVIDER

SOC_LP_SPI_PERIPH_NUM

SOC_LP_SPI_MAXIMUM_BUFFER_SIZE

SOC_SPI_MEM_SUPPORT_AUTO_WAIT_IDLE

SOC_SPI_MEM_SUPPORT_AUTO_SUSPEND

SOC_SPI_MEM_SUPPORT_AUTO_RESUME

SOC_SPI_MEM_SUPPORT_IDLE_INTR

SOC_SPI_MEM_SUPPORT_SW_SUSPEND

SOC_SPI_MEM_SUPPORT_CHECK_SUS

SOC_SPI_MEM_SUPPORT_TIMING_TUNING

SOC_MEMSPI_TIMING_TUNING_BY_DQS

SOC_SPI_MEM_SUPPORT_CACHE_32BIT_ADDR_MAP

SOC_SPI_PERIPH_SUPPORT_CONTROL_DUMMY_OUT

SOC_MEMSPI_SRC_FREQ_80M_SUPPORTED

SOC_MEMSPI_SRC_FREQ_40M_SUPPORTED

SOC_MEMSPI_SRC_FREQ_20M_SUPPORTED

SOC_SPIRAM_XIP_SUPPORTED

SOC_SYSTIMER_COUNTER_NUM

SOC_SYSTIMER_ALARM_NUM

SOC_SYSTIMER_BIT_WIDTH_LO

SOC_SYSTIMER_BIT_WIDTH_HI

SOC_SYSTIMER_FIXED_DIVIDER

SOC_SYSTIMER_SUPPORT_RC_FAST

SOC_SYSTIMER_INT_LEVEL

SOC_SYSTIMER_ALARM_MISS_COMPENSATE

SOC_SYSTIMER_SUPPORT_ETM

SOC_LP_TIMER_BIT_WIDTH_LO

SOC_LP_TIMER_BIT_WIDTH_HI

SOC_TIMER_GROUPS

SOC_TIMER_GROUP_TIMERS_PER_GROUP

SOC_TIMER_GROUP_COUNTER_BIT_WIDTH

SOC_TIMER_GROUP_SUPPORT_XTAL

SOC_TIMER_GROUP_SUPPORT_RC_FAST

SOC_TIMER_GROUP_TOTAL_TIMERS

SOC_TIMER_SUPPORT_ETM

SOC_TIMER_SUPPORT_SLEEP_RETENTION

SOC_MWDT_SUPPORT_XTAL

SOC_MWDT_SUPPORT_SLEEP_RETENTION

SOC_TOUCH_SENSOR_VERSION

Hardware version of touch sensor

SOC_TOUCH_SENSOR_NUM

Touch available channel number. Actually there are 15 Touch channels, but channel 14 is not pinned out, limit to 14 channels

SOC_TOUCH_SUPPORT_SLEEP_WAKEUP

Touch sensor supports sleep awake

SOC_TOUCH_SUPPORT_WATERPROOF

Touch sensor supports waterproof

SOC_TOUCH_SUPPORT_PROX_SENSING

Touch sensor supports proximity sensing

SOC_TOUCH_PROXIMITY_CHANNEL_NUM

Support touch proximity channel number.

SOC_TOUCH_PROXIMITY_MEAS_DONE_SUPPORTED

Support touch proximity channel measure done interrupt type.

SOC_TOUCH_SUPPORT_FREQ_HOP

Touch sensor supports frequency hopping

SOC_TOUCH_SAMPLE_CFG_NUM

The sample configurations number in total, each sampler can be used to sample on one frequency

SOC_TWAI_CONTROLLER_NUM

SOC_TWAI_CLK_SUPPORT_XTAL

SOC_TWAI_BRP_MIN

SOC_TWAI_BRP_MAX

SOC_TWAI_SUPPORTS_RX_STATUS

SOC_EFUSE_DIS_PAD_JTAG

SOC_EFUSE_DIS_USB_JTAG

SOC_EFUSE_DIS_DIRECT_BOOT

SOC_EFUSE_SOFT_DIS_JTAG

SOC_EFUSE_DIS_DOWNLOAD_MSPI

SOC_EFUSE_ECDSA_KEY

SOC_KEY_MANAGER_ECDSA_KEY_DEPLOY
Key manager responsible to deploy ECDSA key

SOC_KEY_MANAGER_FE_KEY_DEPLOY
Key manager responsible to deploy Flash Encryption key

SOC_SECURE_BOOT_V2_RSA

SOC_SECURE_BOOT_V2_ECC

SOC_EFUSE_SECURE_BOOT_KEY_DIGESTS

SOC_EFUSE_REVOKE_BOOT_KEY_DIGESTS

SOC_SUPPORT_SECURE_BOOT_REVOKE_KEY

SOC_FLASH_ENCRYPTED_XTS_AES_BLOCK_MAX

SOC_FLASH_ENCRYPTION_XTS_AES

SOC_FLASH_ENCRYPTION_XTS_AES_OPTIONS

SOC_FLASH_ENCRYPTION_XTS_AES_128

SOC_FLASH_ENCRYPTION_XTS_AES_256

SOC_UART_NUM

SOC_UART_HP_NUM

SOC_UART_LP_NUM

SOC_UART_FIFO_LEN
The UART hardware FIFO length

SOC_LP_UART_FIFO_LEN
The LP UART hardware FIFO length

SOC_UART_BITRATE_MAX
Max bit rate supported by UART

SOC_UART_SUPPORT_PLL_F80M_CLK
Support PLL_F80M as the clock source

SOC_UART_SUPPORT_RTC_CLK
Support RTC clock as the clock source

SOC_UART_SUPPORT_XTAL_CLK
Support XTAL clock as the clock source

SOC_UART_SUPPORT_WAKEUP_INT
Support UART wakeup interrupt

SOC_UART_HAS_LP_UART
Support LP UART

SOC_UART_SUPPORT_SLEEP_RETENTION
Support back up registers before sleep

SOC_UART_SUPPORT_FSM_TX_WAIT_SEND

SOC_COEX_HW_PTI

SOC_PHY_DIG_REGS_MEM_SIZE

SOC_WIFI_LIGHT_SLEEP_CLK_WIDTH

SOC_PM_SUPPORT_EXT1_WAKEUP

SOC_PM_SUPPORT_EXT1_WAKEUP_MODE_PER_PIN
Supports one bit per pin to configure the EXT1 trigger level

SOC_PM_EXT1_WAKEUP_BY_PMU

SOC_PM_SUPPORT_WIFI_WAKEUP

SOC_PM_SUPPORT_TOUCH_SENSOR_WAKEUP
Supports waking up from touch pad trigger

SOC_PM_SUPPORT_XTAL32K_PD

SOC_PM_SUPPORT_RC32K_PD

SOC_PM_SUPPORT_RC_FAST_PD

SOC_PM_SUPPORT_VDDSDIO_PD

SOC_PM_SUPPORT_TOP_PD

SOC_PM_SUPPORT_CNNT_PD

SOC_PM_SUPPORT_RTC_PERIPH_PD

SOC_PM_SUPPORT_DEEPSLEEP_CHECK_STUB_ONLY
Supports CRC only the stub code in RTC memory

SOC_PM_CPU_RETENTION_BY_SW

SOC_PM_PAU_LINK_NUM

SOC_PAU_IN_TOP_DOMAIN

SOC_CPU_IN_TOP_DOMAIN

SOC_PSRAM_VDD_POWER_MPLL

SOC_CLK_RC_FAST_SUPPORT_CALIBRATION

SOC_MODEM_CLOCK_IS_INDEPENDENT

SOC_CLK_APLL_SUPPORTED
Support Audio PLL

SOC_CLK_MPLL_SUPPORTED
Support MSPI PLL

SOC_CLK_XTAL32K_SUPPORTED
Support to connect an external low frequency crystal

SOC_CLK_RC32K_SUPPORTED
Support an internal 32kHz RC oscillator

SOC_CLK_LP_FAST_SUPPORT_LP_PLL
Support LP_PLL clock as the LP_FAST clock source

SOC_PERIPH_CLK_CTRL_SHARED
Peripheral clock control (e.g. set clock source) is shared between various peripherals

SOC_TEMPERATURE_SENSOR_LP_PLL_SUPPORT

SOC_TEMPERATURE_SENSOR_INTR_SUPPORT

SOC_TSENS_IS_INDEPENDENT_FROM_ADC
Temperature sensor is a separate module, not share regs with ADC

SOC_TEMPERATURE_SENSOR_SUPPORT_ETM

SOC_MEM_TCM_SUPPORTED

SOC_MEM_NON_CONTIGUOUS_SRAM

SOC_ASYNCHRONOUS_BUS_ERROR_MODE

SOC_EMAC_IEEE_1588_SUPPORT
EMAC Supports IEEE1588 time stamping

SOC_EMAC_USE_MULTI_IO_MUX
Multiple GPIO pad options exist to connect EMAC signal via IO_MUX

SOC_EMAC_MII_USE_GPIO_MATRIX
EMAC MII signals are connected to GPIO pads via GPIO Matrix

SOC_JPEG_CODEC_SUPPORTED

SOC_JPEG_DECODE_SUPPORTED

SOC_JPEG_ENCODE_SUPPORTED

SOC_LCDCAM_CAM_SUPPORT_RGB_YUV_CONV

SOC_LCDCAM_CAM_PERIPH_NUM

SOC_LCDCAM_CAM_DATA_WIDTH_MAX

SOC_LP_CORE_SUPPORT_ETM
LP Core supports ETM

2.9.30 系统时间

概述

ESP32-P4 使用两种硬件时钟源建立和保持系统时间。根据应用目的及对系统时间的精度要求，既可以仅使用其中一种时钟源，也可以同时使用两种时钟源。这两种硬件时钟源为：

- **RTC 定时器**：RTC 定时器在任何睡眠模式下及在任何复位后均可保持系统时间（上电复位除外，因为上电复位会重置 RTC 定时器）。时钟频率偏差取决于*RTC 定时器时钟源*，该偏差只会在睡眠模式下影响时间精度。睡眠模式下，时间分辨率为 6.667 µs。
- **高分辨率定时器**：高分辨率定时器在睡眠模式下及在复位后不可用，但其时间精度更高。该定时器使用 APB_CLK 时钟源（通常为 80 MHz），时钟频率偏差小于 ±10 ppm，时间分辨率为 1 µs。

可供选择的硬件时钟源组合如下所示：

- RTC 和高分辨率定时器（默认）
- RTC
- 高分辨率定时器
- 无

默认时钟源的时间精度最高，建议使用该配置。此外，用户也可以通过配置选项*CONFIG_NEWLIB_TIME_SYSCALL* 来选择其他时钟源。

RTC 定时器时钟源

RTC 定时器有以下时钟源：

- 内置 136 kHz RC 振荡器（默认）：Deep-sleep 模式下电流消耗最低，不依赖任何外部元件。但由于温度波动会影响该时钟源的频率稳定性，在 Deep-sleep 和 Light-sleep 模式下都有可能发生时间偏移。
- 外置 32 kHz 晶振：需要将一个 32 kHz 晶振连接到 XTAL_32K_P 和 XTAL_32K_N 管脚。频率稳定性更高，但在 Deep-sleep 模式下电流消耗略高（比默认模式高 1 μA）。
- 管脚 XTAL_32K_P 外置 32 kHz 振荡器：允许使用由外部电路产生的 32 kHz 时钟。外部时钟信号必须连接到管脚 XTAL_32K_P。正弦波信号的振幅应小于 1.2 V，方波信号的振幅应小于 1 V。正常模式下，电压范围应为 0.1 < Vcm < 0.5 xVamp，其中 Vamp 代表信号振幅。使用此时钟源时，管脚 XTAL_32K_P 无法用作 GPIO 管脚。
- 内置 32 kHz RC 振荡器

时钟源的选择取决于系统时间精度要求和睡眠模式下的功耗要求。要修改 RTC 时钟源，请在项目配置中设置*CONFIG_RTC_CLK_SRC*。

想要了解外置晶振或外置振荡器的更多布线要求，请参考 硬件设计指南。

获取当前时间

要获取当前时间，请使用 POSIX 函数 `gettimeofday()`。此外，也可以使用以下标准 C 库函数来获取时间并对其进行操作：

```
gettimeofday
time
asctime
clock
ctime
difftime
gmtime
localtime
mktime
strftime
adjtime*
```

如需立即更新当前时间，并暂停平滑时间校正，请使用 POSIX 函数 `settimeofday()`。

若要求时间的分辨率为 1 s，请使用以下代码片段：

```
time_t now;
char strftime_buf[64];
struct tm timeinfo;

time(&now);
// 将时区设置为中国标准时间
setenv("TZ", "CST-8", 1);
tzset();

localtime_r(&now, &timeinfo);
strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
ESP_LOGI(TAG, "The current date/time in Shanghai is: %s", strftime_buf);
```

若要求时间的分辨率为 1 μs，请使用以下代码片段：

```
struct timeval tv_now;
gettimeofday(&tv_now, NULL);
int64_t time_us = (int64_t)tv_now.tv_sec * 1000000L + (int64_t)tv_now.tv_usec;
```

SNTP 时间同步

要设置当前时间，可以使用 POSIX 函数 `settimeofday()` 和 `adjtime()`。lwIP 中的 SNTP 库会在收到 NTP 服务器的响应报文后，调用这两个函数以更新当前的系统时间。当然，用户可以在 lwIP SNTP 库之外独立地使用这两个函数。

包括 SNTP 函数在内的一些 lwIP API 并非线程安全，因此建议在与 SNTP 模块交互时使用*esp_netif component*。

要初始化特定的 SNTP 服务器并启动 SNTP 服务，只需创建有特定服务器名称的默认 SNTP 服务器配置，然后调用*esp_netif_sntp_init()* 注册该服务器并启动 SNTP 服务。

```
esp_sntp_config_t config = ESP_NETIF_SNTP_DEFAULT_CONFIG("pool.ntp.org");
esp_netif_sntp_init(&config);
```

一旦收到 SNTP 服务器的响应，此代码会自动执行时间同步。有时等待时间同步很有意义，调用*esp_netif_sntp_sync_wait()* 可实现此目的：

```
if (esp_netif_sntp_sync_wait(pdMS_TO_TICKS(10000)) != ESP_OK) {
    printf("Failed to update system time within 10s timeout");
}
```

要配置多个 NTP 服务器（或使用更高级的设置，例如 DHCP 提供的 NTP 服务器），请参考*esp_netif* 文档*SNTP API* 中的详细说明。

lwIP SNTP 库可在下列任一同步模式下工作：

- *SNTP_SYNC_MODE_IMMED*（默认）：使用 `settimeofday()`，收到 SNTP 服务器响应后立即更新系统时间。
- *SNTP_SYNC_MODE_SMOOTH*：使用函数 `adjtime()` 逐渐减少时间误差以平滑更新时间。如果 SNTP 响应时间和系统时间之差超过 35 分钟，请立即使用 `settimeofday()` 更新系统时间。

如要选择*SNTP_SYNC_MODE_SMOOTH* 模式，请将 SNTP 配置结构体中的 `esp_sntp_config::smooth` 设置为 `true`，否则将默认使用*SNTP_SYNC_MODE_IMMED* 模式。

设置时间同步时的回调函数，请使用配置结构体中的*esp_sntp_config::sync_cb* 字段。

添加此初始化代码后，应用程序将定期同步时间。时间同步周期由*CONFIG_LWIP_SNTP_UPDATE_DELAY* 设置（默认为一小时）。如需修改，请在项目配置中设置*CONFIG_LWIP_SNTP_UPDATE_DELAY*。

如需查看示例代码，请前往 protocols/sntp 目录。该目录下的示例展示了如何基于 lwIP SNTP 库实现时间同步。

也可以直接使用 lwIP API，但请务必注意线程安全。线程安全的 API 如下：

- *sntp_set_time_sync_notification_cb()* 用于设置通知时间同步过程的回调函数。
- *sntp_get_sync_status()* 和*sntp_set_sync_status()* 用于获取/设置时间同步状态。
- *sntp_set_sync_mode()* 用于设置同步模式。
- *esp_sntp_setoperatingmode()* 用于设置首选操作模式。*ESP_SNTP_OPMODE_POLL* 和*esp_sntp_init()* 可初始化 SNTP 模块。
- *esp_sntp_setservername()* 用于配置特定 SNTP 服务器。

时区

要设置本地时区，请使用以下 POSIX 函数：

1. 调用 `setenv()`，将 `TZ` 环境变量根据设备位置设置为正确的值。时间字符串的格式与 GNU libc 文档 中描述的相同（但实现方式不同）。

2. 调用 `tzset()`，为新的时区更新 C 库的运行数据。

完成上述步骤后，请调用标准 C 库函数 `localtime()`。该函数将返回排除时区偏差和夏令时干扰后的准确本地时间。

2036 年和 2038 年溢出问题

SNTP/NTP 2036 年溢出问题 SNTP/NTP 时间戳为 64 位无符号定点数，其中前 32 位表示整数部分，后 32 位表示小数部分。该 64 位无符号定点数代表从 1900 年 1 月 1 日 00:00 起经过的秒数，因此 SNTP/NTP 时间将在 2036 年溢出。

为了解决这一问题，可以使用整数部分的 MSB（惯例为位 0）来表示 1968 年到 2104 年之间的时间范围（查看 RFC2030 了解更多信息），这一惯例将使得 SNTP/NTP 时间戳的生命周期延长。该惯例会在 lwIP 库的 SNTP 模块中实现，因此 ESP-IDF 中 SNTP 相关功能在 2104 年之前能够经受住时间的考验。

Unix 时间 2038 年溢出问题 Unix 时间（类型 `time_t`）此前为有符号的 32 位整数，因此将于 2038 年溢出（即 Y2K38 问题）。为了解决 Y2K38 问题，ESP-IDF 从 v5.0 版本起开始使用有符号的 64 位整数来表示 `time_t`，从而将 `time_t` 溢出推迟 2920 亿年。

API 参考

Header File

- components/lwip/include/apps/esp_sntp.h
- This header file can be included with:

```
#include "esp_sntp.h"
```

- This header file is a part of the API provided by the `lwip` component. To declare that your component depends on `lwip`, add the following to your CMakeLists.txt:

```
REQUIRES lwip
```

or

```
PRIV_REQUIRES lwip
```

Functions

void **sntp_sync_time**(struct timeval *tv)

This function updates the system time.

This is a weak-linked function. It is possible to replace all SNTP update functionality by placing a sntp_sync_time() function in the app firmware source. If the default implementation is used, calling sntp_set_sync_mode() allows the time synchronization mode to be changed to instant or smooth. If a callback function is registered via sntp_set_time_sync_notification_cb(), it will be called following time synchronization.

参数 **tv** -- Time received from SNTP server.

void **sntp_set_sync_mode**(*sntp_sync_mode_t* sync_mode)

Set the sync mode.

Modes allowed: SNTP_SYNC_MODE_IMMED and SNTP_SYNC_MODE_SMOOTH.

参数 **sync_mode** -- Sync mode.

sntp_sync_mode_t **sntp_get_sync_mode**(void)

Get set sync mode.

返回 SNTP_SYNC_MODE_IMMED: Update time immediately. SNTP_SYNC_MODE_SMOOTH: Smooth time updating.

sntp_sync_status_t **sntp_get_sync_status** (void)

Get status of time sync.

After the update is completed, the status will be returned as SNTP_SYNC_STATUS_COMPLETED. After that, the status will be reset to SNTP_SYNC_STATUS_RESET. If the update operation is not completed yet, the status will be SNTP_SYNC_STATUS_RESET. If a smooth mode was chosen and the synchronization is still continuing (adjtime works), then it will be SNTP_SYNC_STATUS_IN_PROGRESS.

返回 SNTP_SYNC_STATUS_RESET: Reset status. SNTP_SYNC_STATUS_COMPLETED: Time is synchronized. SNTP_SYNC_STATUS_IN_PROGRESS: Smooth time sync in progress.

void **sntp_set_sync_status** (*sntp_sync_status_t* sync_status)

Set status of time sync.

参数 **sync_status** -- status of time sync (see sntp_sync_status_t)

void **sntp_set_time_sync_notification_cb** (*sntp_sync_time_cb_t* callback)

Set a callback function for time synchronization notification.

参数 **callback** -- a callback function

void **sntp_set_sync_interval** (uint32_t interval_ms)

Set the sync interval of SNTP operation.

Note: SNTPv4 RFC 4330 enforces a minimum sync interval of 15 seconds. This sync interval will be used in the next attempt update time throught SNTP. To apply the new sync interval call the sntp_restart() function, otherwise, it will be applied after the last interval expired.

参数 **interval_ms** -- The sync interval in ms. It cannot be lower than 15 seconds, otherwise 15 seconds will be set.

uint32_t **sntp_get_sync_interval** (void)

Get the sync interval of SNTP operation.

返回 the sync interval

bool **sntp_restart** (void)

Restart SNTP.

返回 True - Restart False - SNTP was not initialized yet

void **esp_sntp_setoperatingmode** (*esp_sntp_operatingmode_t* operating_mode)

Sets SNTP operating mode. The mode has to be set before init.

参数 **operating_mode** -- Desired operating mode

void **esp_sntp_init** (void)

Init and start SNTP service.

void **esp_sntp_stop** (void)

Stops SNTP service.

void **esp_sntp_setserver** (u8_t idx, const ip_addr_t *addr)

Sets SNTP server address.

参数

- **idx** -- Index of the server
- **addr** -- IP address of the server

void **esp_sntp_setservername** (u8_t idx, const char *server)

Sets SNTP hostname.

参数

- **idx** -- Index of the server
- **server** -- Name of the server

const char ***esp_sntp_getservername**(u8_t idx)

Gets SNTP server name.

参数 **idx** -- Index of the server
返回 Name of the server

const ip_addr_t ***esp_sntp_getserver**(u8_t idx)

Get SNTP server IP.

参数 **idx** -- Index of the server
返回 IP address of the server

bool **esp_sntp_enabled**(void)

Checks if sntp is enabled.

返回 true if sntp module is enabled

uint8_t **esp_sntp_getreachability**(uint8_t idx)

Gets the server reachability shift register as described in RFC 5905.

参数 **idx** -- Index of the SNTP server
返回 reachability shift register

esp_sntp_operatingmode_t **esp_sntp_getoperatingmode**(void)

Get the configured operating mode.

返回 operating mode enum

static inline void **sntp_setoperatingmode**(u8_t operating_mode)

if not build within lwip, provide translating inlines, that will warn about thread safety

static inline void **sntp_servermode_dhcp**(int set_servers_from_dhcp)

static inline void **sntp_setservername**(u8_t idx, const char *server)

static inline void **sntp_init**(void)

static inline const char ***sntp_getservername**(u8_t idx)

static inline const ip_addr_t ***sntp_getserver**(u8_t idx)

static inline uint8_t **sntp_getreachability**(uint8_t idx)

static inline *esp_sntp_operatingmode_t* **sntp_getoperatingmode**(void)

Macros

esp_sntp_sync_time

Aliases for esp_sntp prefixed API (inherently thread safe)

esp_sntp_set_sync_mode

esp_sntp_get_sync_mode

esp_sntp_get_sync_status

esp_sntp_set_sync_status

esp_sntp_set_time_sync_notification_cb

esp_sntp_set_sync_interval

esp_sntp_get_sync_interval

esp_sntp_restart

SNTP_OPMODE_POLL

Type Definitions

typedef void (***sntp_sync_time_cb_t**)(struct timeval *tv)

SNTP callback function for notifying about time sync event.

Param tv Time received from SNTP server.

Enumerations

enum **sntp_sync_mode_t**

SNTP time update mode.

Values:

enumerator **SNTP_SYNC_MODE_IMMED**

Update system time immediately when receiving a response from the SNTP server.

enumerator **SNTP_SYNC_MODE_SMOOTH**

Smooth time updating. Time error is gradually reduced using adjtime function. If the difference between SNTP response time and system time is large (more than 35 minutes) then update immediately.

enum **sntp_sync_status_t**

SNTP sync status.

Values:

enumerator **SNTP_SYNC_STATUS_RESET**

enumerator **SNTP_SYNC_STATUS_COMPLETED**

enumerator **SNTP_SYNC_STATUS_IN_PROGRESS**

enum **esp_sntp_operatingmode_t**

SNTP operating modes per lwip SNTP module.

Values:

enumerator **ESP_SNTP_OPMODE_POLL**

enumerator **ESP_SNTP_OPMODE_LISTENONLY**

2.9.31 异步内存复制

概述

ESP32-P4 有一个 DMA 引擎，能够以异步方式帮助 CPU 完成内部内存复制操作。

异步 memcpy API 中封装了所有 DMA 配置和操作，*esp_async_memcpy()* 的签名与标准 C 库的 memcpy 函数基本相同。

DMA 允许多个内存复制请求在首个请求完成之前排队，即允许计算和内存复制的重叠。此外，通过注册事件回调函数，还可以知道内存复制请求完成的准确时间。

如果异步 memcpy 是基于 AXI GDMA 构建的，那么也可以用适合的对齐方式从 PSRAM 复制数据或向其中复制数据。

配置并安装驱动

安装异步 memcpy 驱动的方法取决于底层 DMA 引擎：

- *esp_async_memcpy_install_gdma_ahb()* 用于安装基于 AHB GDMA 引擎的异步 memcpy 驱动。
- *esp_async_memcpy_install_gdma_axi()* 用于安装基于 AXI GDMA 引擎的异步 memcpy 驱动。
- *esp_async_memcpy_install()* 是一个通用 API，用于安装带有默认 DMA 引擎的异步 memcpy 驱动。如果 SoC 具有 CP DMA 引擎，则默认 DMA 引擎为 CP DMA，否则，默认 DMA 引擎为 AHB GDMA。

在*async_memcpy_config_t* 中设置驱动配置：

- backlog：此项用于配置首个请求完成前可以排队的最大内存复制事务数量。如果将此字段设置为零，会应用默认值 4。
- dma_burst_size：设置单次 DMA 传输中突发数据量的大小。
- flags：此项可以启用一些特殊的驱动功能。

```
async_memcpy_config_t config = ASYNC_MEMCPY_DEFAULT_CONFIG();
// 更新底层 DMA 引擎支持的最大数据流
config.backlog = 8;
async_memcpy_handle_t driver = NULL;
ESP_ERROR_CHECK(esp_async_memcpy_install(&config, &driver)); // 使用默认 DMA␣
↪引擎安装驱动
```

发送内存复制请求

使用*esp_async_memcpy()* API 将内存复制请求发送到 DMA 引擎。在驱动程序成功安装后才能调用该 API。此 API 是线程安全的，因此可以从不同的任务中调用。

与 libc 版本的 memcpy 不同，你可以选择给*esp_async_memcpy()* 设置一个回调函数，以便在内存复制完成时收到通知。注意，回调是在 ISR 上下文中执行的，请不要在回调中调用任何阻塞函数。

回调函数的原型是*async_memcpy_isr_cb_t*。回调函数只有在借助 RTOS API（如 xSemaphoreGiveFromISR()）唤醒了高优先级任务后才能返回 true。

```
// 回调实现，在 ISR 上下文中运行
static bool my_async_memcpy_cb(async_memcpy_handle_t mcp_hdl, async_memcpy_event_t␣
↪*event, void *cb_args)
{
    SemaphoreHandle_t sem = (SemaphoreHandle_t)cb_args;
    BaseType_t high_task_wakeup = pdFALSE;
    xSemaphoreGiveFromISR(semphr, &high_task_wakeup); //␣
↪如果解锁了一些高优先级任务，则将 high_task_wakeup 设置为 pdTRUE
    return high_task_wakeup == pdTRUE;
```

(下页继续)

(续上页)

```
}

// 创建一个信号量，在异步 memcpy 完成时进行报告
SemaphoreHandle_t semphr = xSemaphoreCreateBinary();

// 从用户的上下文中调用
ESP_ERROR_CHECK(esp_async_memcpy(driver_handle, to, from, copy_len, my_async_
↪memcpy_cb, my_semaphore));
// 其他事项
xSemaphoreTake(my_semaphore, portMAX_DELAY); // 等待 buffer 复制完成
```

卸载驱动

使用*esp_async_memcpy_uninstall()* 卸载异步 memcpy 驱动。无需在每次 memcpy 操作后手动卸载。如果你的应用程序不再需要此驱动，此 API 可以帮助回收内存和其他硬件资源。

ETM 事件

在异步 memcpy 操作完成时会生成一个事件，此事件能够与*ETM* 模块进行交互。可以调用*esp_async_memcpy_new_etm_event()* 获取 ETM 事件句柄。

如需了解如何将此事件连接到 ETM 通道，请参考文档*ETM*。

API 参考

Header File

- components/esp_hw_support/include/esp_async_memcpy.h
- This header file can be included with:

```
#include "esp_async_memcpy.h"
```

Functions

esp_err_t **esp_async_memcpy_install_gdma_ahb**(const *async_memcpy_config_t* *config, *async_memcpy_handle_t* *mcp)

Install async memcpy driver, with AHB-GDMA as the backend.

参数

- **config** -- **[in]** Configuration of async memcpy
- **mcp** -- **[out]** Returned driver handle

返回

- ESP_OK: Install async memcpy driver successfully
- ESP_ERR_INVALID_ARG: Install async memcpy driver failed because of invalid argument
- ESP_ERR_NO_MEM: Install async memcpy driver failed because out of memory
- ESP_FAIL: Install async memcpy driver failed because of other error

esp_err_t **esp_async_memcpy_install_gdma_axi**(const *async_memcpy_config_t* *config, *async_memcpy_handle_t* *mcp)

Install async memcpy driver, with AXI-GDMA as the backend.

参数

- **config** -- **[in]** Configuration of async memcpy
- **mcp** -- **[out]** Returned driver handle

返回

- ESP_OK: Install async memcpy driver successfully

- ESP_ERR_INVALID_ARG: Install async memcpy driver failed because of invalid argument
- ESP_ERR_NO_MEM: Install async memcpy driver failed because out of memory
- ESP_FAIL: Install async memcpy driver failed because of other error

esp_err_t **esp_async_memcpy_install**(const *async_memcpy_config_t* *config, *async_memcpy_handle_t* *mcp)

Install async memcpy driver with the default DMA backend.

备注： On chip with CPDMA support, CPDMA is the default choice. On chip with AHB-GDMA support, AHB-GDMA is the default choice.

参数

- **config** -- **[in]** Configuration of async memcpy
- **mcp** -- **[out]** Returned driver handle

返回

- ESP_OK: Install async memcpy driver successfully
- ESP_ERR_INVALID_ARG: Install async memcpy driver failed because of invalid argument
- ESP_ERR_NO_MEM: Install async memcpy driver failed because out of memory
- ESP_FAIL: Install async memcpy driver failed because of other error

esp_err_t **esp_async_memcpy_uninstall**(*async_memcpy_handle_t* mcp)

Uninstall async memcpy driver.

参数 **mcp** -- **[in]** Handle of async memcpy driver that returned from `esp_async_memcpy_install`

返回

- ESP_OK: Uninstall async memcpy driver successfully
- ESP_ERR_INVALID_ARG: Uninstall async memcpy driver failed because of invalid argument
- ESP_FAIL: Uninstall async memcpy driver failed because of other error

esp_err_t **esp_async_memcpy**(*async_memcpy_handle_t* mcp, void *dst, void *src, size_t n, *async_memcpy_isr_cb_t* cb_isr, void *cb_args)

Send an asynchronous memory copy request.

备注： The callback function is invoked in interrupt context, never do blocking jobs in the callback.

参数

- **mcp** -- **[in]** Handle of async memcpy driver that returned from `esp_async_memcpy_install`
- **dst** -- **[in]** Destination address (copy to)
- **src** -- **[in]** Source address (copy from)
- **n** -- **[in]** Number of bytes to copy
- **cb_isr** -- **[in]** Callback function, which got invoked in interrupt context. Set to NULL can bypass the callback.
- **cb_args** -- **[in]** User defined argument to be passed to the callback function

返回

- ESP_OK: Send memory copy request successfully
- ESP_ERR_INVALID_ARG: Send memory copy request failed because of invalid argument
- ESP_FAIL: Send memory copy request failed because of other error

esp_err_t **esp_async_memcpy_new_etm_event**(*async_memcpy_handle_t* mcp, *async_memcpy_etm_event_t* event_type, *esp_etm_event_handle_t* *out_event)

Get the ETM event handle for async memcpy done signal.

备注: The created ETM event object can be deleted later by calling `esp_etm_del_event`

参数

- **mcp** -- **[in]** Handle of async memcpy driver that returned from `esp_async_memcpy_install`
- **event_type** -- **[in]** ETM event type
- **out_event** -- **[out]** Returned ETM event handle

返回

- ESP_OK: Get ETM event successfully
- ESP_ERR_INVALID_ARG: Get ETM event failed because of invalid argument
- ESP_ERR_NOT_SUPPORTED: Get ETM event failed because the DMA hardware doesn't support ETM submodule
- ESP_FAIL: Get ETM event failed because of other error

Structures

struct **async_memcpy_event_t**

Async memory copy event data.

Public Members

void ***data**

Event data

struct **async_memcpy_config_t**

Type of async memcpy configuration.

Public Members

uint32_t **backlog**

Maximum number of transactions that can be prepared in the background

size_t **sram_trans_align**

DMA transfer alignment (both in size and address) for SRAM memory

size_t **psram_trans_align**

DMA transfer alignment (both in size and address) for PSRAM memory

size_t **dma_burst_size**

DMA transfer burst size, in bytes

uint32_t **flags**

Extra flags to control async memcpy feature

Macros

ASYNC_MEMCPY_DEFAULT_CONFIG()

Default configuration for async memcpy.

Type Definitions

typedef struct async_memcpy_context_t ***async_memcpy_handle_t**

Async memory copy driver handle.

typedef bool (***async_memcpy_isr_cb_t**)(*async_memcpy_handle_t* mcp_hdl, *async_memcpy_event_t* *event, void *cb_args)

Type of async memcpy interrupt callback function.

备注： User can call OS primitives (semaphore, mutex, etc) in the callback function. Keep in mind, if any OS primitive wakes high priority task up, the callback should return true.

Param mcp_hdl Handle of async memcpy
Param event Event object, which contains related data, reserved for future
Param cb_args User defined arguments, passed from esp_async_memcpy function
Return Whether a high priority task is woken up by the callback function

Enumerations

enum **async_memcpy_etm_event_t**

Async memory copy specific events that supported by the ETM module.

Values:

enumerator **ASYNC_MEMCPY_ETM_EVENT_COPY_DONE**

memory copy finished

2.9.32 ULP LP-Core 协处理器编程

ULP LP-Core（低功耗内核）协处理器是 ESP32-P4 中 ULP 的一个变型。它具有超低功耗，同时还能在主 CPU 处于低功耗模式时保持运行。因此，LP-Core 协处理器能够在主 CPU 处于睡眠模式时处理 GPIO 或传感器读取等任务，从而显著降低整个系统的整体功耗。

ULP LP-Core 协处理器具有以下功能：

- 利用基于 RISC-V ISA 的 32 位处理器，包括标准扩展整数 (I)、乘法/除法 (M)、原子 (A) 和压缩 (C)。
- 中断控制器。
- 包含一个调试模块，支持通过 JTAG 进行外部调试。
- 当整个系统处于 active 模式时，可以访问所有的高功耗 (HP) SRAM 和外设。
- 当 HP 系统处于睡眠模式时，可以访问低功耗 (LP) SRAM 和外设。

编译 ULP LP-Core 代码

ULP LP-Core 代码会与 ESP-IDF 项目共同编译，生成一个单独的二进制文件，并自动嵌入到主项目的二进制文件中。编译操作如下：

1. 将用 C 语言或汇编语言编写的 ULP LP-Core 代码（带有 `.S` 扩展名）放在组件目录下的专用目录中，例如 `ulp/`。
2. 在 CMakeLists.txt 文件中注册组件后，调用 `ulp_embed_binary` 函数。例如：

```
idf_component_register()

set(ulp_app_name ulp_${COMPONENT_NAME})
set(ulp_sources "ulp/ulp_c_source_file.c" "ulp/ulp_assembly_source_file.S")
set(ulp_exp_dep_srcs "ulp_c_source_file.c")

ulp_embed_binary(${ulp_app_name} "${ulp_sources}" "${ulp_exp_dep_srcs}")
```

ulp_embed_binary 的第一个参数为 ULP 二进制文件的文件名，该文件名也用于其他生成的文件，如 ELF 文件、映射文件、头文件和链接器导出文件。第二个参数为 ULP 源文件。第三个参数为组件源文件列表，用于包含要生成的头文件。要正确构建依赖关系、确保在编译这些文件前创建要生成的头文件，都需要此文件列表。有关 ULP 应用程序生成头文件的概念，请参阅本文档后续章节。

1. 在 menuconfig 中启用*CONFIG_ULP_COPROC_ENABLED* 和*CONFIG_ULP_COPROC_TYPE* 选项，并将*CONFIG_ULP_COPROC_TYPE* 设置为 CONFIG_ULP_COPROC_TYPE_LP_CORE。*CONFIG_ULP_COPROC_RESERVE_MEM* 选项为 ULP 保留 RTC 内存，因此必须设置为一个足够大的值，以存储 ULP LP-Core 代码和数据。如果应用程序组件包含多个 ULP 程序，那么 RTC 内存的大小必须足够容纳其中最大的程序。
2. 按照常规步骤构建应用程序（例如 idf.py app）。

在构建过程中，采取以下步骤来构建 ULP 程序：

1. **通过 C 编译器和汇编器运行每个源文件**。此步骤会在组件构建目录中生成目标文件 .obj.c 或 .obj.S，具体取决于处理的源文件。
2. **通过 C 预处理器运行链接器脚本模板**。模板位于 components/ulp/ld 目录中。
3. **将对象文件链接到一个 ELF 输出文件中，**即 ulp_app_name.elf。在此阶段生成的映射文件 ulp_app_name.map 可用于调试。
4. **将 ELF 文件的内容转储到一个二进制文件中，**即 ulp_app_name.bin。此二进制文件接下来可以嵌入到应用程序中。
5. 使用 riscv32-esp-elf-nm 在 ELF 文件中 **生成全局符号列表，**即 ulp_app_name.sym。
6. **创建一个 LD 导出脚本和一个头文件，**即 ulp_app_name.ld 和 ulp_app_name.h，并在其中包含 ulp_app_name.sym 中的符号。此步骤可以通过 esp32ulp_mapgen.py 实现。
7. **将生成的二进制文件添加到要嵌入到应用程序中的二进制文件列表**。

访问 ULP LP-Core 程序变量

在主程序中可以使用在 ULP LP-Core 程序中定义的全局符号。

例如，ULP LP-Core 程序定义了一个变量 measurement_count，用来表示程序从深度睡眠中唤醒芯片前所需的 GPIO 测量次数。

```
volatile int measurement_count;

int some_function()
{
    //读取测量次数以便后续使用。
    int temp = measurement_count;

    ...do something.
}
```

主程序可以访问 ULP LP-Core 程序全局变量，这是因为构建系统生成了 ${ULP_APP_NAME}.h 和 ${ULP_APP_NAME}.ld 文件，文件中定义了 ULP LP-Core 程序中现有的的全局符号。在 ULP LP-Core 程序中定义的每个全局符号都包含在这两个文件中，并具有前缀 ulp_。

头文件中包含符号的声明：

```
extern uint32_t ulp_measurement_count;
```

注意，所有的符号（变量、数组、函数）都被声明为 uint32_t 类型。对于函数和数组，获取符号的地址并将其转换为合适的类型。

生成的链接器脚本文件定义了 LP_MEM 中符号的位置：

```
PROVIDE ( ulp_measurement_count = 0x50000060 );
```

要从主程序访问 ULP LP-Core 程序变量，应使用 include 语句将生成的头文件包含在主程序中，这样就可以像访问常规变量一样访问 ULP LP-Core 程序变量。

```
#include "ulp_app_name.h"

void init_ulp_vars() {
    ulp_measurement_count = 64;
}
```

备注： LP-Core 程序全局变量存储在二进制文件的 .bss 或者 .data 部分。这些部分在加载和执行 LP-Core 二进制文件时被初始化。在首次运行 LP-Core 之前，从 HP-Core 主程序访问这些变量可能会导致未定义行为。

启动 ULP LP-Core 程序

要运行 ULP LP-Core 程序，主应用程序需要先使用*ulp_lp_core_load_binary()* 函数将 ULP 程序加载到 RTC 内存中，然后使用*ulp_lp_core_run()* 函数进行启动。

每个 ULP LP-Core 程序以二进制 blob 的形式嵌入到 ESP-IDF 应用程序中。应用程序可以按照如下方式引用和加载该 blob（假设 ULP_APP_NAME 被定义为 ulp_app_name）：

```
extern const uint8_t bin_start[] asm("_binary_ulp_app_name_bin_start");
extern const uint8_t bin_end[]   asm("_binary_ulp_app_name_bin_end");

void start_ulp_program() {
    ESP_ERROR_CHECK( ulp_lp_core_load_binary( bin_start,
        (bin_end - bin_start)) );
}
```

将程序加载到 LP 内存后，就可以调用*ulp_lp_core_run()* 配置和启动应用程序：

```
ulp_lp_core_cfg_t cfg = {
    .wakeup_source = ULP_LP_CORE_WAKEUP_SOURCE_LP_TIMER, // LP 内核会定期被 LP␣
↪定时器唤醒
    .lp_timer_sleep_duration_us = 10000,
};

ESP_ERROR_CHECK( ulp_lp_core_run(&cfg) );
```

ULP LP-Core 程序流程

ULP LP-Core 协处理器如何启动取决于*ulp_lp_core_cfg_t* 中选择的唤醒源。最常见的用例是 ULP 定期唤醒，在进行一些测量后唤醒主 CPU，或者再次进入睡眠状态。

ULP 有以下唤醒源：

- *ULP_LP_CORE_WAKEUP_SOURCE_HP_CPU* - LP 内核可以被 HP CPU 唤醒。
- *ULP_LP_CORE_WAKEUP_SOURCE_LP_TIMER* - LP 内核可以被 LP 定时器唤醒。
- *ULP_LP_CORE_WAKEUP_SOURCE_ETM* - LP 内核可以被 ETM 事件唤醒。（暂不支持）
- *ULP_LP_CORE_WAKEUP_SOURCE_LP_IO* - 当 LP IO 电平变化时，LP 内核会被唤醒。（暂不支持）
- *ULP_LP_CORE_WAKEUP_SOURCE_LP_UART* - LP 内核在接收到一定数量的 UART RX 脉冲后会被唤醒。（暂不支持）

ULP 被唤醒时会经历以下步骤：

1. 除非已指定*ulp_lp_core_cfg_t::skip_lp_rom_boot*，否则运行 ROM 启动代码并跳转至 LP RAM 中的入口地址。ROM 启动代码将初始化 LP UART 并打印启动信息。
2. 初始化系统功能，如中断
3. 调用用户代码 `main()`
4. 从 `main()` 返回
5. 如果指定了 `lp_timer_sleep_duration_us`，则配置下一个唤醒闹钟
6. 调用*ulp_lp_core_halt()*

ULP LP-Core 支持的外设

为了增强 ULP LP-Core 协处理器的功能，它可以访问在低功耗电源域运行的外设。ULP LP-Core 协处理器可以在主 CPU 处于睡眠模式时与这些外设进行交互，并在达到唤醒条件时唤醒主 CPU。以下为支持的外设：

- LP IO
- LP I2C
- LP UART
- LP SPI

ULP LP-Core ROM

ULP LP-Core ROM 是位于 LP-ROM 中的一小段预编译代码，用户无法修改。与主 CPU 运行的引导加载程序 ROM 代码类似，ULP LP-Core ROM 也在 ULP LP-Core 协处理器启动时执行。该 ROM 代码会初始化 ULP LP-Core 协处理器，随后跳转到用户程序。如果已初始化 LP UART，该 ROM 代码还会打印启动信息。

如果已将*ulp_lp_core_cfg_t::skip_lp_rom_boot* 设置为真，则不会执行 ULP LP-Core ROM 代码。如需尽快唤醒 ULP，同时避免初始化和信息打印产生额外开销，则可使用这一功能。

除上述启动代码，ULP LP-Core ROM 代码还提供以下功能和接口：

- ROM.ld 接口
- newlib.ld 接口

在任何情况下，这些函数都存在于 LP-ROM 中，因此在程序中使用这些函数可以减少 ULP 应用程序的 RAM 占用。

ULP LP-Core 中断

配置 LP-Core 协处理器，可以处理各种类型的中断，例如 LP IO 低/高电平中断或是 LP 定时器中断。只需重写 IDF 提供的任何一个弱处理函数，就可以注册一个中断处理程序。所有处理程序可见 ulp_lp_core_interrupts.h 。有关特定目标可使用的中断的详细信息，请参阅 **ESP32-P4 技术参考手册** [PDF]。

例如，要重写 LP IO 中断的处理程序，可以在 ULP LP-Core 代码中定义以下函数：

```
void LP_CORE_ISR_ATTR ulp_lp_core_lp_io_intr_handler(void)
{
    // 处理中断，清除中断源
}
```

LP_CORE_ISR_ATTR 宏用于定义中断处理函数，可确保调用中断处理程序时妥善保存并恢复寄存器。

除了为需要处理的中断源配置相关的中断寄存器外，还要调用*ulp_lp_core_intr_enable()* 函数，在 LP-Core 中断控制器中使能全局中断。

调试 ULP LP-Core 应用程序

在编程 LP-Core 时，有时很难弄清楚程序未按预期运行的原因。请参考以下策略，调试 LP-Core 程序：

- 使用 LP-UART 打印：LP-Core 可以访问 LP-UART 外设，在主 CPU 处于睡眠状态时独立打印信息。有关使用此驱动程序的示例，请参阅 system/ulp/lp_core/lp_uart/lp_uart_print。
- 通过*CONFIG_ULP_HP_UART_CONSOLE_PRINT*，将`lp_core_printf()` 路由到 HP-Core 控制台 UART，可以轻松地将 LP-Core 信息打印到已经连接的 HP-Core 控制台 UART。此方法的缺点是需要主 CPU 处于唤醒状态，并且由于 LP 核与 HP 核未同步，输出可能会交错。
- 通过共享变量共享程序状态：如*访问 ULP LP-Core 程序变量* 所述，主 CPU 和 ULP 内核都可以轻松访问 RTC 内存中的全局变量。若想了解 ULP 内核的运行状态，可以将状态信息从 ULP 写入变量中，并通过主 CPU 读取信息。这种方法的缺点在于它需要主 CPU 一直处于唤醒状态，而这通常很难实现。另外，若主 CPU 一直处于唤醒状态，可能会掩盖某些问题，因为部分问题只会在特定电源域断电时发生。
- 紧急处理程序：当检测到异常时，LP-Core 的紧急处理程序会把 LP-Core 寄存器的状态通过 LP-UART 发送出去。将*CONFIG_ULP_PANIC_OUTPUT_ENABLE* 选项设置为 `y`，可以启用紧急处理程序。禁用此选项将减少 LP-Core 应用程序的 LP-RAM 使用量。若想从紧急转储中解析栈回溯，可以使用 esp-idf-monitor，例如：

```
python -m esp_idf_monitor --toolchain-prefix riscv32-esp-elf- --target␣
↪ESP32-P4 --decode-panic backtrace PATH_TO_ULP_ELF_FILE
```

应用示例

- 在示例 system/ulp/lp_core/gpio 中，ULP LP-Core 协处理器在主 CPU 深度睡眠时轮询 GPIO。
- 在示例 system/ulp/lp_core/lp_i2c 中，ULP LP-Core 协处理器在主 CPU 深度睡眠时读取外部 I2C 环境光传感器 (BH1750)，并在达到阈值时唤醒主 CPU。
- 在示例 system/ulp/lp_core/lp_uart/lp_uart_echo 中，低功耗内核上运行的 LP UART 驱动程序读取并回显写入串行控制台的数据。
- system/ulp/lp_core/lp_uart/lp_uart_print 展示了如何在低功耗内核上使用串口打印功能。
- system/ulp/lp_core/interrupt 展示了如何在 LP 内核上注册中断处理程序，接收由主 CPU 触发的中断。
- system/ulp/lp_core/gpio_intr_pulse_counter 展示了如何在主 CPU 处于 Deep-sleep 模式时，使用 GPIO 中断为脉冲计数。

API 参考

主 CPU API 参考

Header File

- components/ulp/lp_core/include/ulp_lp_core.h
- This header file can be included with:

```
#include "ulp_lp_core.h"
```

- This header file is a part of the API provided by the `ulp` component. To declare that your component depends on `ulp`, add the following to your CMakeLists.txt:

```
REQUIRES ulp
```

or

```
PRIV_REQUIRES ulp
```

Functions

esp_err_t **ulp_lp_core_run**(*ulp_lp_core_cfg_t* *cfg)

Configure the ULP and run the program loaded into RTC memory.

返回 ESP_OK on success

esp_err_t **ulp_lp_core_load_binary**(const uint8_t *program_binary, size_t program_size_bytes)

Load the program binary into RTC memory.

参数

- **program_binary** -- pointer to program binary
- **program_size_bytes** -- size of the program binary

返回

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if program_size_bytes is more than KiB

void **ulp_lp_core_stop**(void)

Puts the ulp to sleep and disables all wakeup sources. To restart the ULP call ulp_lp_core_run() with the desired wake up trigger.

void **ulp_lp_core_sw_intr_trigger**(void)

Trigger a SW interrupt to the LP CPU from the PMU.

备注: This is the same SW trigger that is used to wake up the LP CPU

Structures

struct **ulp_lp_core_cfg_t**

ULP LP core init parameters.

Public Members

uint32_t **wakeup_source**

Wakeup source flags

uint32_t **lp_timer_sleep_duration_us**

Sleep duration when ULP_LP_CORE_WAKEUP_SOURCE_LP_TIMER is specified. Measurement unit: us

bool **skip_lp_rom_boot**

Skips the LP rom code and boots directly into the app code placed in LP RAM, this gives faster boot time for time sensitive use-cases at the cost of skipping setup e.g. of UART

Macros

ULP_LP_CORE_WAKEUP_SOURCE_HP_CPU

ULP_LP_CORE_WAKEUP_SOURCE_LP_UART

ULP_LP_CORE_WAKEUP_SOURCE_LP_IO

ULP_LP_CORE_WAKEUP_SOURCE_ETM

ULP_LP_CORE_WAKEUP_SOURCE_LP_TIMER

Header File

- components/ulp/lp_core/include/lp_core_i2c.h
- This header file can be included with:

```
#include "lp_core_i2c.h"
```

- This header file is a part of the API provided by the `ulp` component. To declare that your component depends on `ulp`, add the following to your CMakeLists.txt:

```
REQUIRES ulp
```

or

```
PRIV_REQUIRES ulp
```

Functions

esp_err_t **lp_core_i2c_master_init** (*i2c_port_t* lp_i2c_num, const *lp_core_i2c_cfg_t* *cfg)

Initialize and configure the LP I2C for use by the LP core Currently LP I2C can only be used in master mode.

备注: The internal pull-up resistors for SDA and SCL pins, if enabled, will provide a weak pull-up value of about 30-50 kOhm. Users are adviced to enable external pull-ups for better performance at higher SCL frequencies.

参数

- **lp_i2c_num** -- LP I2C port number
- **cfg** -- Configuration parameters

返回 esp_err_t ESP_OK when successful

Structures

struct **lp_core_i2c_pin_cfg_t**

LP Core I2C pin config parameters.

Public Members

gpio_num_t **sda_io_num**

GPIO pin for SDA signal. Only GPIO#6 can be used as the SDA pin.

gpio_num_t **scl_io_num**

GPIO pin for SCL signal. Only GPIO#7 can be used as the SCL pin.

bool **sda_pullup_en**

SDA line enable internal pullup. Can be configured if external pullup is not used.

bool **scl_pullup_en**

SCL line enable internal pullup. Can be configured if external pullup is not used.

struct **lp_core_i2c_timing_cfg_t**

LP Core I2C timing config parameters.

Public Members

uint32_t **clk_speed_hz**
LP I2C clock speed for master mode

struct **lp_core_i2c_cfg_t**
LP Core I2C config parameters.

Public Members

lp_core_i2c_pin_cfg_t **i2c_pin_cfg**
LP I2C pin configuration

lp_core_i2c_timing_cfg_t **i2c_timing_cfg**
LP I2C timing configuration

soc_periph_lp_i2c_clk_src_t **i2c_src_clk**
LP I2C source clock type

Macros

LP_I2C_DEFAULT_GPIO_CONFIG()

LP_I2C_FAST_MODE_CONFIG()

LP_I2C_STANDARD_MODE_CONFIG()

LP_I2C_DEFAULT_SRC_CLK()

LP_CORE_I2C_DEFAULT_CONFIG()

Header File

- components/ulp/lp_core/include/lp_core_uart.h
- This header file can be included with:

```
#include "lp_core_uart.h"
```

- This header file is a part of the API provided by the `ulp` component. To declare that your component depends on `ulp`, add the following to your CMakeLists.txt:

```
REQUIRES ulp
```

or

```
PRIV_REQUIRES ulp
```

Functions

esp_err_t **lp_core_uart_init**(const *lp_core_uart_cfg_t* *cfg)
Initialize and configure the LP UART to be used from the LP core.

备注: The LP UART initialization must be called from the main core (HP CPU)

参数 **cfg** -- Configuration parameters
返回 esp_err_t ESP_OK when successful

Structures

struct **lp_core_uart_pin_cfg_t**

LP UART IO pins configuration.

Public Members

gpio_num_t **tx_io_num**

GPIO pin for UART Tx signal. Only GPIO#5 can be used as the UART Tx pin

gpio_num_t **rx_io_num**

GPIO pin for UART Rx signal. Only GPIO#4 can be used as the UART Rx pin

gpio_num_t **rts_io_num**

GPIO pin for UART RTS signal. Only GPIO#2 can be used as the UART RTS pin

gpio_num_t **cts_io_num**

GPIO pin for UART CTS signal. Only GPIO#3 can be used as the UART CTS pin

struct **lp_core_uart_proto_cfg_t**

LP UART protocol configuration.

Public Members

int **baud_rate**

LP UART baud rate

uart_word_length_t **data_bits**

LP UART byte size

uart_parity_t **parity**

LP UART parity mode

uart_stop_bits_t **stop_bits**

LP UART stop bits

uart_hw_flowcontrol_t **flow_ctrl**

LP UART HW flow control mode (cts/rts)

uint8_t **rx_flow_ctrl_thresh**

LP UART HW RTS threshold

struct **lp_core_uart_cfg_t**

LP UART configuration parameters.

Public Members

lp_core_uart_pin_cfg_t **uart_pin_cfg**

LP UART pin configuration

lp_core_uart_proto_cfg_t **uart_proto_cfg**

LP UART protocol configuration

lp_uart_sclk_t **lp_uart_source_clk**

LP UART source clock selection

Macros

LP_UART_DEFAULT_TX_GPIO_NUM

Default LP_IO Mux pins for LP UART

LP_UART_DEFAULT_RX_GPIO_NUM

LP_UART_DEFAULT_RTS_GPIO_NUM

LP_UART_DEFAULT_CTS_GPIO_NUM

LP_UART_DEFAULT_GPIO_CONFIG()

LP_UART_DEFAULT_PROTO_CONFIG()

LP_UART_DEFAULT_CLOCK_CONFIG()

LP_CORE_UART_DEFAULT_CONFIG()

Header File

- components/ulp/lp_core/include/lp_core_spi.h
- This header file can be included with:

```
#include "lp_core_spi.h"
```

- This header file is a part of the API provided by the `ulp` component. To declare that your component depends on `ulp`, add the following to your CMakeLists.txt:

```
REQUIRES ulp
```

or

```
PRIV_REQUIRES ulp
```

Functions

esp_err_t **lp_core_lp_spi_bus_initialize**(*lp_spi_host_t* host_id, const *lp_spi_bus_config_t* *bus_config)

Initialize the LP SPI bus for use by the LP core.

参数

- **host_id** -- LP SPI host number
- **bus_config** -- LP SPI bus configuration parameters

返回 esp_err_t ESP_OK when successful ESP_ERR_INVALID_ARG if the configuration is invalid

esp_err_t **lp_core_lp_spi_bus_add_device**(*lp_spi_host_t* host_id, const *lp_spi_device_config_t* *dev_config)

Initialize the LP SPI controller in master mode and add an SPI device to the LP SPI bus.

参数

- **host_id** -- LP SPI host number
- **dev_config** -- LP SPI device configuration parameters

返回 esp_err_t ESP_OK when successful ESP_ERR_INVALID_ARG if the configuration is invalid ESP_FAIL if the device could not be added

esp_err_t **lp_core_lp_spi_slave_initialize**(*lp_spi_host_t* host_id, const *lp_spi_slave_config_t* *slave_config)

Initialize the LP SPI controller in slave mode.

参数

- **host_id** -- LP SPI host number
- **slave_config** -- LP SPI slave configuration parameters

返回 esp_err_t ESP_OK when successful ESP_FAIL if the SPI controller could not be initialized in slave mode

Structures

struct **lp_spi_bus_config_t**

LP SPI bus configuration parameters.

Public Members

int **mosi_io_num**

GPIO pin for Master out, Slave In signal, a.k.a, SPI_D.

int **miso_io_num**

GPIO pin for Master in, Slave Out signal, a.k.a, SPI_Q.

int **sclk_io_num**

GPIO pin for LP SPI Clock signal.

struct **lp_spi_device_config_t**

LP SPI device configuration parameters.

Public Members

int **cs_io_num**

GPIO pin for the device Chip Select (CS) signal.

int **clock_speed_hz**

SPI clock speed in Hz.

int **spi_mode**

SPI mode, representing a pair of Clock Polarity (CPOL) and Clock Phase (CPHA) configuration:

- SPI Mode 0: (0, 0)
- SPI Mode 1: (0, 1)
- SPI Mode 2: (1, 0)

- SPI Mode 3: (1, 1)

int **duty_cycle**

Duty cycle of positive SPI clock, in 1/256th increments (128 = 50% duty cycle). Setting this to 0 (=not setting it) is equivalent to setting this to 128.

int **flags**

Bitwise OR of LP_SPI_DEVICE_* flags

int **cs_ena_pretrans**

Amount of SPI bit-cycles the CS should be active for, before the transmission (0-16). This only works on half-duplex transactions.

int **cs_ena_posttrans**

Amount of SPI bit-cycles the CS should stay active for, after the transmission (0-16). This only works on half-duplex transactions.

struct **lp_spi_slave_config_t**

LP SPI slave configuration parameters.

Public Members

int **cs_io_num**

GPIO pin for the device Chip Select (CS) signal.

int **spi_mode**

SPI mode, representing a pair of Clock Polarity (CPOL) and Clock Phase (CPHA) configuration:

- SPI Mode 0: (0, 0)
- SPI Mode 1: (0, 1)
- SPI Mode 2: (1, 0)
- SPI Mode 3: (1, 1)

int **flags**

Bitwise OR of LP_SPI_DEVICE_* flags

Macros

LP_SPI_DEVICE_TXBIT_LSBFIRST

LP SPI device configuration flags.

Transmit command/address/data LSB first instead of the default MSB first

LP_SPI_DEVICE_RXBIT_LSBFIRST

Receive data LSB first instead of the default MSB first

LP_SPI_DEVICE_BIT_LSBFIRST

Transmit and receive LSB first

LP_SPI_DEVICE_3WIRE

Use MOSI (=spid) for both sending and receiving data

LP_SPI_DEVICE_CS_ACTIVE_HIGH

Make CS line active-high during a transanction instead of the default active-low state. Only available in SPI master mode.

LP_SPI_DEVICE_HALF_DUPLEX

Transmit data before receiving it, instead of simultaneously. Only available in SPI master mode.

Type Definitions

typedef uint32_t **lp_spi_host_t**

LP SPI peripheral Since we have just one LP SPI peripheral, we can define it as a uint32_t type for now, instead of an enum.

Header File

- components/ulp/lp_core/include/lp_core_etm.h
- This header file can be included with:

```
#include "lp_core_etm.h"
```

- This header file is a part of the API provided by the `ulp` component. To declare that your component depends on `ulp`, add the following to your CMakeLists.txt:

```
REQUIRES ulp
```

or

```
PRIV_REQUIRES ulp
```

Functions

esp_err_t **lp_core_new_etm_event**(const *lp_core_etm_event_config_t* *config, *esp_etm_event_handle_t* *out_event)

Create a ETM event for LP-Core.

备注: The created ETM event object can be deleted later by calling `esp_etm_del_event`

参数

- **config** -- **[in]** LP-Core ETM event configuration
- **out_event** -- **[out]** Returned ETM event handle

返回

- ESP_OK: Get ETM event successfully
- ESP_ERR_INVALID_ARG: Get ETM event failed because of invalid argument
- ESP_FAIL: Get ETM event failed because of other error

esp_err_t **lp_core_new_etm_task**(const *lp_core_etm_task_config_t* *config, *esp_etm_task_handle_t* *out_task)

Create a ETM task for LP-Core.

备注: The created ETM task object can be deleted later by calling `esp_etm_del_task`

参数

- **config** -- **[in]** LP-Core ETM task configuration
- **out_task** -- **[out]** Returned ETM task handle

返回

- ESP_OK: Get ETM task successfully
- ESP_ERR_INVALID_ARG: Get ETM task failed because of invalid argument
- ESP_FAIL: Get ETM task failed because of other error

Structures

struct **lp_core_etm_event_config_t**

LP-Core ETM event configuration.

Public Members

lp_core_etm_event_type_t **event_type**

LP-Core ETM event type

struct **lp_core_etm_task_config_t**

LP-Core ETM task configuration.

Public Members

lp_core_etm_task_type_t **task_type**

LP-Core ETM task type

Header File

- components/hal/include/hal/lp_core_types.h
- This header file can be included with:

```
#include "hal/lp_core_types.h"
```

Enumerations

enum **lp_core_etm_event_type_t**

LP-Core event types enum.

Values:

enumerator **LP_CORE_EVENT_ERR_INTR**

Exception triggered on LP-Core

enumerator **LP_CORE_EVENT_START_INTR**

LP-Core clock has been turned on

enumerator **LP_CORE_EVENT_MAX**

Maximum number of LP-Core events

enum **lp_core_etm_task_type_t**

LP-Core task types enum.

Values:

enumerator **LP_CORE_TASK_WAKEUP_CPU**

LP-Core wake-up task

enumerator **LP_CORE_TASK_MAX**

Maximum number of LP-Core tasks

LP 内核 API 参考

Header File

- components/ulp/lp_core/lp_core/include/ulp_lp_core_utils.h

Functions

void **ulp_lp_core_update_wakeup_cause**(void)

Traverse all possible wake-up sources and update the wake-up cause so that ulp_lp_core_get_wakeup_cause can obtain the bitmap of the wake-up reasons.

uint32_t **ulp_lp_core_get_wakeup_cause**(void)

Get the wakeup source which caused LP_CPU to wakeup from sleep.

返回 Wakeup cause in bit map, for the meaning of each bit, refer to the definition of wakeup source in lp_core_ll.h

void **ulp_lp_core_wakeup_main_processor**(void)

Wakeup main CPU from sleep or deep sleep.

This raises a software interrupt signal, if the main CPU has configured the ULP as a wakeup source calling this function will make the main CPU to exit from sleep or deep sleep.

void **ulp_lp_core_delay_us**(uint32_t us)

Makes the co-processor busy wait for a certain number of microseconds.

参数 **us** -- Number of microseconds to busy-wait for

void **ulp_lp_core_delay_cycles**(uint32_t cycles)

Makes the co-processor busy wait for a certain number of cycles.

参数 **cycles** -- Number of cycles to busy-wait for

void **ulp_lp_core_halt**(void)

Finishes the ULP program and powers down the ULP until next wakeup.

备注: This function does not return. After called it will fully reset the ULP.

备注: The program will automatically call this function when returning from main().

备注: To stop the ULP from waking up, call ulp_lp_core_lp_timer_disable() before halting.

---

void **ulp_lp_core_stop_lp_core**(void)

The LP core puts itself to sleep and disables all wakeup sources.

void **ulp_lp_core_sw_intr_enable**(bool enable)

Enable the SW triggered interrupt from the PMU.

备注: This is the same SW trigger interrupt that is used to wake up the LP CPU

参数 enable -- true to enable, false to disable

void **ulp_lp_core_sw_intr_clear**(void)

Clear the interrupt status for the SW triggered interrupt from the PMU.

void **ulp_lp_core_wait_for_intr**(void)

Puts the CPU into a wait state until an interrupt is triggered.

备注: The CPU will draw less power when in this state compared to actively running

Header File

- components/ulp/lp_core/lp_core/include/ulp_lp_core_gpio.h

Functions

static inline void **ulp_lp_core_gpio_init**(*lp_io_num_t* lp_io_num)

Initialize a rtcio pin.

备注: If IO is used in LP application, `rtc_gpio_init` must be called at least once for the using IO before loading LP core firmware in HP Code.

参数 lp_io_num -- The rtc io pin to initialize

static inline void **ulp_lp_core_gpio_output_enable**(*lp_io_num_t* lp_io_num)

Enable output.

参数 lp_io_num -- The rtc io pin to enable output for

static inline void **ulp_lp_core_gpio_output_disable**(*lp_io_num_t* lp_io_num)

Disable output.

参数 lp_io_num -- The rtc io pin to disable output for

static inline void **ulp_lp_core_gpio_input_enable**(*lp_io_num_t* lp_io_num)

Enable input.

参数 lp_io_num -- The rtc io pin to enable input for

static inline void **ulp_lp_core_gpio_input_disable**(*lp_io_num_t* lp_io_num)

Disable input.

参数 lp_io_num -- The rtc io pin to disable input for

static inline void **ulp_lp_core_gpio_set_level**(*lp_io_num_t* lp_io_num, uint8_t level)

Set rtcio output level.

参数

- **lp_io_num** -- The rtc io pin to set the output level for
- **level** -- 0: output low; 1: output high.

static inline uint32_t **ulp_lp_core_gpio_get_level**(*lp_io_num_t* lp_io_num)

Get rtcio output level.

参数 **lp_io_num** -- The rtc io pin to get the output level for

static inline void **ulp_lp_core_gpio_set_output_mode**(*lp_io_num_t* lp_io_num, rtcio_ll_out_mode_t mode)

Set rtcio output mode.

参数

- **lp_io_num** -- The rtc io pin to set the output mode for
- **mode** -- RTCIO_LL_OUTPUT_NORMAL: normal, RTCIO_LL_OUTPUT_OD: open drain

static inline void **ulp_lp_core_gpio_pullup_enable**(*lp_io_num_t* lp_io_num)

Enable internal pull-up resistor.

参数 **lp_io_num** -- The rtc io pin to enable pull-up for

static inline void **ulp_lp_core_gpio_pullup_disable**(*lp_io_num_t* lp_io_num)

Disable internal pull-up resistor.

参数 **lp_io_num** -- The rtc io pin to disable pull-up for

static inline void **ulp_lp_core_gpio_pulldown_enable**(*lp_io_num_t* lp_io_num)

Enable internal pull-down resistor.

参数 **lp_io_num** -- The rtc io pin to enable pull-down for

static inline void **ulp_lp_core_gpio_pulldown_disable**(*lp_io_num_t* lp_io_num)

Disable internal pull-down resistor.

参数 **lp_io_num** -- The rtc io pin to disable pull-down for

static inline void **ulp_lp_core_gpio_intr_enable**(*lp_io_num_t* lp_io_num, *lp_io_intr_type_t* intr_type)

Enable interrupt for lp io pin.

参数

- **lp_io_num** -- The lp io pin to enable interrupt for
- **intr_type** -- The interrupt type to enable

static inline void **ulp_lp_core_gpio_clear_intr_status**(void)

Clear interrupt status for all lp io.

Macros

RTCIO_OUTPUT_NORMAL

RTCIO_OUTPUT_OD

Enumerations

enum **lp_io_num_t**

Values:

enumerator **LP_IO_NUM_0**

GPIO0, input and output

enumerator **LP_IO_NUM_1**

GPIO1, input and output

enumerator **LP_IO_NUM_2**
GPIO2, input and output

enumerator **LP_IO_NUM_3**
GPIO3, input and output

enumerator **LP_IO_NUM_4**
GPIO4, input and output

enumerator **LP_IO_NUM_5**
GPIO5, input and output

enumerator **LP_IO_NUM_6**
GPIO6, input and output

enumerator **LP_IO_NUM_7**
GPIO7, input and output

enumerator **LP_IO_NUM_8**
GPIO8, input and output

enumerator **LP_IO_NUM_9**
GPIO9, input and output

enumerator **LP_IO_NUM_10**
GPIO10, input and output

enumerator **LP_IO_NUM_11**
GPIO11, input and output

enumerator **LP_IO_NUM_12**
GPIO12, input and output

enumerator **LP_IO_NUM_13**
GPIO13, input and output

enumerator **LP_IO_NUM_14**
GPIO14, input and output

enumerator **LP_IO_NUM_15**
GPIO15, input and output

enum **lp_io_intr_type_t**
Values:

enumerator **LP_IO_INTR_DISABLE**
Disable GPIO interrupt

enumerator **LP_IO_INTR_POSEDGE**

GPIO interrupt type : rising edge

enumerator **LP_IO_INTR_NEGEDGE**

GPIO interrupt type : falling edge

enumerator **LP_IO_INTR_ANYEDGE**

GPIO interrupt type : both rising and falling edge

enumerator **LP_IO_INTR_LOW_LEVEL**

GPIO interrupt type : input low level trigger

enumerator **LP_IO_INTR_HIGH_LEVEL**

GPIO interrupt type : input high level trigger

Header File

- components/ulp/lp_core/lp_core/include/ulp_lp_core_i2c.h

Functions

esp_err_t **lp_core_i2c_master_read_from_device**(*i2c_port_t* lp_i2c_num, uint16_t device_addr, uint8_t *data_rd, size_t size, int32_t ticks_to_wait)

Read from I2C device.

备注: The LP I2C must have been initialized from the HP core using the lp_core_i2c_master_init() API before invoking this API.

备注: the LP I2C does not support 10-bit I2C device addresses.

备注: the LP I2C port number is ignored at the moment.

参数

- **lp_i2c_num** -- LP I2C port number
- **device_addr** -- I2C device address (7-bit)
- **data_rd** -- Buffer to hold data to be read
- **size** -- Size of data to be read in bytes
- **ticks_to_wait** -- Operation timeout in CPU cycles. Set to -1 to wait forever.

返回 esp_err_t ESP_OK when successful

esp_err_t **lp_core_i2c_master_write_to_device**(*i2c_port_t* lp_i2c_num, uint16_t device_addr, const uint8_t *data_wr, size_t size, int32_t ticks_to_wait)

Write to I2C device.

备注: The LP I2C must have been initialized from the HP core using the lp_core_i2c_master_init() API before invoking this API.

备注： the LP I2C does not support 10-bit I2C device addresses.

备注： the LP I2C port number is ignored at the moment.

参数

- **lp_i2c_num** -- LP I2C port number
- **device_addr** -- I2C device address (7-bit)
- **data_wr** -- Buffer which holds the data to be written
- **size** -- Size of data to be written in bytes
- **ticks_to_wait** -- Operation timeout in CPU cycles. Set to -1 to wait forever.

返回 esp_err_t ESP_OK when successful

esp_err_t **lp_core_i2c_master_write_read_device**(*i2c_port_t* lp_i2c_num, uint16_t device_addr, const uint8_t *data_wr, size_t write_size, uint8_t *data_rd, size_t read_size, int32_t ticks_to_wait)

Write to and then read from an I2C device in a single transaction.

备注： The LP I2C must have been initialized from the HP core using the lp_core_i2c_master_init() API before invoking this API.

备注： the LP I2C does not support 10-bit I2C device addresses.

备注： the LP I2C port number is ignored at the moment.

参数

- **lp_i2c_num** -- LP I2C port number
- **device_addr** -- I2C device address (7-bit)
- **data_wr** -- Buffer which holds the data to be written
- **write_size** -- Size of data to be written in bytes
- **data_rd** -- Buffer to hold data to be read
- **read_size** -- Size of data to be read in bytes
- **ticks_to_wait** -- Operation timeout in CPU cycles. Set to -1 to wait forever.

返回 esp_err_t ESP_OK when successful

void **lp_core_i2c_master_set_ack_check_en**(*i2c_port_t* lp_i2c_num, bool ack_check_en)

Enable or disable ACK checking by the LP_I2C controller during write operations.

When ACK checking is enabled, the hardware will check the ACK/NACK level received during write operations against the expected ACK/NACK level. If the received ACK/NACK level does not match the expected ACK/NACK level then the hardware will generate the I2C_NACK_INT and a STOP condition will be generated to stop the data transfer.

备注： ACK checking is enabled by default

备注： the LP I2C port number is ignored at the moment.

参数

- **lp_i2c_num** -- LP I2C port number
- **ack_check_en** -- true: enable ACK check false: disable ACK check

Header File

- components/ulp/lp_core/lp_core/include/ulp_lp_core_uart.h

Functions

int **lp_core_uart_tx_chars**(*uart_port_t* lp_uart_num, const void *src, size_t size)

Send data to the LP UART port if there is space available in the Tx FIFO.

This function will not wait for enough space in the Tx FIFO to be available. It will just fill the available Tx FIFO slots and return when the FIFO is full. If there are no empty slots in the Tx FIFO, this function will not write any data.

参数

- **lp_uart_num** -- LP UART port number
- **src** -- data buffer address
- **size** -- data length to send

返回 - (-1) Error

- OTHERS (>=0) The number of bytes pushed to the Tx FIFO

esp_err_t **lp_core_uart_write_bytes**(*uart_port_t* lp_uart_num, const void *src, size_t size, int32_t timeout)

Write data to the LP UART port.

This function will write data to the Tx FIFO. If a timeout value is configured, this function will timeout once the number of CPU cycles expire.

参数

- **lp_uart_num** -- LP UART port number
- **src** -- data buffer address
- **size** -- data length to send
- **timeout** -- Operation timeout in CPU cycles. Set to -1 to wait forever.

返回 esp_err_t ESP_OK when successful

int **lp_core_uart_read_bytes**(*uart_port_t* lp_uart_num, void *buf, size_t size, int32_t timeout)

Read data from the LP UART port.

This function will read data from the Rx FIFO. If a timeout value is configured, then this function will timeout once the number of CPU cycles expire.

参数

- **lp_uart_num** -- LP UART port number
- **buf** -- data buffer address
- **size** -- data length to send
- **timeout** -- Operation timeout in CPU cycles. Set to -1 to wait forever.

返回 - (-1) Error

- OTHERS (>=0) The number of bytes read from the Rx FIFO

Header File

- components/ulp/lp_core/lp_core/include/ulp_lp_core_print.h

Functions

void **lp_core_printf**(const char *format, ...)

Print from the LP core.

备注： This function uses the LP UART peripheral to enable prints.The LP UART must be initialized with lp_core_uart_init() before using this function.

备注： This function is not a standard printf function and may not support all format specifiers or special characters.

参数

- **format** -- string to be printed
- **...** -- variable argument list

void **lp_core_print_char**(char c)

Print a single character from the LP core.

参数 **c** -- character to be printed

Header File

- components/ulp/lp_core/lp_core/include/ulp_lp_core_interrupts.h

Functions

void **ulp_lp_core_intr_enable**(void)

Enables interrupts globally for the low power core.

Available interrupt handlers for the low power core are as follows:

ulp_lp_core_lp_io_intr_handler(void);
ulp_lp_core_lp_uart_intr_handler(void);
ulp_lp_core_lp_pmu_intr_handler(void);
ulp_lp_core_trng_intr_handler(void);
ulp_lp_core_lp_touch_intr_handler(void);
ulp_lp_core_efuse_intr_handler(void);
ulp_lp_core_lp_ana_peri_intr_handler(void);
ulp_lp_core_lp_wdt_intr_handler(void);
ulp_lp_core_sw_intr_handler(void);
ulp_lp_core_lp_i2c_intr_handler(void);
ulp_lp_core_lp_timer_intr_handler(void);
ulp_lp_core_lp_spi_intr_handler(void);
ulp_lp_core_lp_adc_intr_handler(void);
ulp_lp_core_tsens_intr_handler(void);
ulp_lp_core_lp_sysreg_intr_handler(void);
ulp_lp_core_mailbox_intr_handler(void);
ulp_lp_core_lp_rtc_intr_handler(void);

Not all handlers are available on all chips. Please refer to the Technical Reference Manual for your chip for more information.

void **ulp_lp_core_intr_disable**(void)

Disables interrupts globally for the low power core.

Macros

LP_CORE_ISR_ATTR

Header File

- components/ulp/lp_core/lp_core/include/ulp_lp_core_spi.h

Functions

esp_err_t **lp_core_lp_spi_master_transfer**(*lp_spi_transaction_t* *trans_desc, int32_t ticks_to_wait)

Initiate an LP SPI transaction in master mode to transmit device to an SPI device and optionally receive data from the device.

参数

- **trans_desc** -- LP SPI transaction configuration descriptor
- **ticks_to_wait** -- Operation timeout in CPU cycles. Set to -1 to wait forever.

返回 esp_err_t ESP_OK when successful ESP_ERR_INVALID_ARG if the configuration is invalid ESP_ERR_TIMEOUT when the operation times out

esp_err_t **lp_core_lp_spi_slave_transfer**(*lp_spi_transaction_t* *trans_desc, int32_t ticks_to_wait)

Initiate an LP SPI transaction in slave mode to receive data from an SPI master and optionally transmit data back to the master.

参数

- **trans_desc** -- LP SPI transaction configuration descriptor
- **ticks_to_wait** -- Operation timeout in CPU cycles. Set to -1 to wait forever.

返回 esp_err_t ESP_OK when successful ESP_ERR_INVALID_ARG if the configuration is invalid ESP_ERR_TIMEOUT when the operation times out

Structures

struct **lp_spi_transaction_t**

This structure describes one SPI transaction. The descriptor should not be modified until the transaction finishes.

Public Members

uint32_t **tx_length**

Total data length to transmit in bytes

uint32_t **rx_length**

Total data length to receive in bytes

const void ***tx_buffer**

Pointer to the transmit buffer. Must be set for master mode transactions. Can be NULL for slave mode transactions.

void ***rx_buffer**

Pointer to the receive buffer. Must be set for slave mode transactions. Can be NULL for master mode transactions.

lp_spi_bus_t **bus**

The LP SPI bus to transmit the data on

int **command**

Command data, of which the length is set in the `command_bits` field of this structure.

uint32_t **address**

Address data, of which the length is set in the `address_bits` field of this structure.

uint8_t **command_bits**

Default amount of bits in command phase

uint8_t **address_bits**

Default amount of bits in address phase

uint8_t **dummy_bits**

Amount of dummy bits to insert between address and data phase.

Type Definitions

typedef uint32_t **lp_spi_bus_t**

The LP SPI bus identifier to initiate a transaction on.

2.9.33 看门狗

概述

ESP-IDF 支持以下类型的看门狗定时器：

- 硬件看门狗定时器
- 中断看门狗定时器 (IWDT)
- 任务看门狗定时器 (TWDT)

中断看门狗负责确保 ISR（中断服务程序）不被长时间阻塞，TWDT 负责检测任务长时间运行而不让步的情况。

通过项目配置菜单 可启用各种看门狗定时器。其中，TWDT 也可以在程序运行时启用。

硬件看门狗定时器

芯片有两组看门狗定时器：

- 主系统看门狗定时器 (MWDT_WDT) - 用于中断看门狗定时器 (IWDT) 和任务看门狗定时器 (TWDT)。
- RTC 看门狗定时器 (RTC_WDT) - 用于跟踪从上电到执行用户主函数的启动时间（默认情况下，RTC 看门狗在执行用户主函数之前会被立即禁用）。

请参阅看门狗 小节，了解如何在引导加载程序中使用看门狗。

用户可以调整应用程序行为，使 RTC 看门狗在应用程序启动后保持启用状态。应用程序需要显式重置（即喂狗）或禁用看门狗，以避免芯片重置。具体而言，用户可设置*CONFIG_BOOTLOADER_WDT_DISABLE_IN_USER_CODE* 选项，根据需要修改应用程序并重新编译。此过程中应使用以下 API：

- `wdt_hal_disable()`：参考禁用 *RTC_WDT*
- `wdt_hal_feed()`：参考重置 *RTC_WDT* 计数器

如果未能及时重置或禁用 RTC_WDT，芯片将自动重置。请参阅*RTC 看门狗超时* 了解更多信息。

中断看门狗定时器 (IWDT)

IWDT 的目的是，确保中断服务例程 (ISR) 运行不会受到长时间阻塞（即 IWDT 超时）。阻塞 ISR 及时运行会增加 ISR 延迟，也会阻止任务切换（因为任务切换是从 ISR 执行的）。阻止 ISR 运行的事项包括：

- 禁用中断
- 临界区（也会禁用中断）

- 其他相同或更高优先级的 ISR，在完成前会阻止相同或较低优先级的 ISR

IWDT 利用定时器组 1 中的 MWDT_WDT 看门狗定时器作为其底层硬件定时器，并在每个 CPU 上使用 FreeRTOS 时钟滴答中断，即 tick 中断。如果某个 CPU 上的 tick 中断没有在 IWDT 超时前运行，就表明该 CPU 上的 ISR 运行受阻（参见上文原因列表）。

当 IWDT 超时后，默认操作是调用紧急处理程序 (Panic Handler)，并显示出错原因（`Interrupt wdt timeout on CPU0` 或 `Interrupt wdt timeout on CPU1`，视情况而定）。根据紧急处理程序的配置行为（参见*CONFIG_ESP_SYSTEM_PANIC*），用户可通过回溯、OpenOCD、gdbstub 等来调试 IWDT 超时问题，也可以重置芯片（这在生产环境中可能是首选）。

如果出于某种原因，IWDT 超时后紧急处理程序无法运行，IWDT 还可以通过其二阶段超时来硬重置芯片（即系统重置）。

配置

- IWDT 默认通过*CONFIG_ESP_INT_WDT* 选项启用。
- 通过*CONFIG_ESP_INT_WDT_TIMEOUT_MS* 选项设置 IWDT 超时。
  - 注意，如果启用了 PSRAM 支持，那么默认的超时时间会更长，因为在某些情况下，临界区或中断例程访问大量 PSRAM 需要更长时间。
  - 超时时间至少应是 FreeRTOS tick 周期的两倍时长（参见*CONFIG_FREERTOS_HZ*）。

调优 如果 IWDT 超时是中断或临界区运行超时导致的，可以考虑重写代码：

- 临界区应尽可能短。任何非关键的代码或计算都应放在临界区外。
- 中断处理程序也应尽可能减少计算量。考虑让 ISR 使用队列向任务推送数据，从而将计算推迟到任务中进行。

临界区或中断处理程序都不应阻塞其他事件。如果不能或不希望通过更改代码减少处理时间，可以通过设置*CONFIG_ESP_INT_WDT_TIMEOUT_MS* 延长超时时间。

任务看门狗定时器 (TWDT)

任务看门狗定时器 (TWDT) 用于监视特定任务，确保任务在配置的超时时间内执行。TWDT 主要监视每个 CPU 的空闲任务，但其他任务也可以订阅 TWDT 监视。通过监视每个 CPU 的空闲任务，TWDT 可以检测到任务长时间运行没有让出的情况。这可能表明代码编写不当，在外设上自旋循环，或者任务陷入了无限循环。

TWDT 是基于定时器组 0 中的 MWDT_WDT 看门狗定时器构建的。超时发生时会触发中断。

可以在用户代码中定义函数 `esp_task_wdt_isr_user_handler` 来接收超时事件，并扩展默认行为。

使用

调用以下函数，用 TWDT 监视任务：

- *`esp_task_wdt_init()`* 初始化 TWDT 并订阅空闲任务。
- *`esp_task_wdt_add()`* 为其他任务订阅 TWDT。
- 订阅后，应从任务中调用*`esp_task_wdt_reset()`* 来喂 TWDT。
- *`esp_task_wdt_delete()`* 可以取消之前订阅的任务。
- *`esp_task_wdt_deinit()`* 取消订阅空闲任务并反初始化 TWDT。

在需要更细粒度级别监视的情况下（即确保调用特定的函数、存根、代码路径），TWDT 允许订阅 `users`。

- *`esp_task_wdt_add_user()`* 订阅 TWDT 的任意用户。此函数返回添加用户的用户句柄。
- 必须使用用户句柄调用*`esp_task_wdt_reset_user()`*，防止 TWDT 超时。
- *`esp_task_wdt_delete_user()`* 取消订阅 TWDT 的任意用户。

配置 TWDT 的默认超时时间可以通过*CONFIG_ESP_TASK_WDT_TIMEOUT_S* 配置项进行设置，并应至少设置为任何单个任务预计需要独占 CPU 的时长，例如某应用程序将进行长时间的密集计算且不让位给其他任务时的预计时长。也可以调用`esp_task_wdt_init()`，在运行时更改此时间。

备注： 擦除较大的 flash 区域可能会非常耗时，并可能导致任务连续运行，触发 TWDT 超时。以下两种方法可以避免这种情况：

- 在 menuconfig 中增加*CONFIG_ESP_TASK_WDT_TIMEOUT_S*，延长看门狗超时时间。
- 在擦除 flash 区域前，调用`esp_task_wdt_init()` 增加看门狗超时时间。

如需了解更多信息，请参考*SPI flash API*。

以下配置选项控制 TWDT 配置，默认情况下全部启用：

- *CONFIG_ESP_TASK_WDT_EN* - 启用 TWDT 功能。如果禁用此选项，TWDT 即使运行时已初始化也无法使用。
- *CONFIG_ESP_TASK_WDT_INIT* - TWDT 在启动期间自动初始化。禁用此选项时，仍可以调用`esp_task_wdt_init()` 在运行时初始化 TWDT。
- *CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0* - 空闲任务在启动时订阅了 TWDT。如果此选项被禁用，仍可以调用`esp_task_wdt_init()` 再次订阅。
- *CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1* - CPU1 空闲任务在启动时订阅了 TWDT。

备注： 如果 TWDT 超时，会默认在继续运行应用程序前打印警告和回溯。如希望超时触发系统严重错误和系统重置，可以通过*CONFIG_ESP_TASK_WDT_PANIC* 进行配置。

JTAG & 看门狗

在使用 OpenOCD 进行调试时，CPU 会在每次达到断点时停止运行。然而，如果遇到断点后看门狗定时器继续运行，就会最终触发复位，为调试代码带来巨大的困难。因此，OpenOCD 会在每个断点处禁用中断和任务的看门狗的硬件定时器。此外，在离开断点时，OpenOCD 也不会重新启用定时器，也就是说，中断看门狗和任务看门狗实际上被禁用。当 ESP32-P4 通过 JTAG 连接到 OpenOCD 时，看门狗不会打印任何警告或出现严重错误。

API 参考

任务看门狗 在 ESP-IDF 中使用任务看门狗的完整示例：system/task_watchdog

Header File

- components/esp_system/include/esp_task_wdt.h
- This header file can be included with:

```
#include "esp_task_wdt.h"
```

Functions

esp_err_t **esp_task_wdt_init** (const *esp_task_wdt_config_t* *config)

Initialize the Task Watchdog Timer (TWDT)

This function configures and initializes the TWDT. This function will subscribe the idle tasks if configured to do so. For other tasks, users can subscribe them using esp_task_wdt_add() or esp_task_wdt_add_user(). This function won't start the timer if no task have been registered yet.

备注： esp_task_wdt_init() must only be called after the scheduler is started. Moreover, it must not be called by multiple tasks simultaneously.

参数 **config** -- **[in]** Configuration structure

返回

- ESP_OK: Initialization was successful
- ESP_ERR_INVALID_STATE: Already initialized
- Other: Failed to initialize TWDT

esp_err_t **esp_task_wdt_reconfigure**(const *esp_task_wdt_config_t* *config)

Reconfigure the Task Watchdog Timer (TWDT)

The function reconfigures the running TWDT. It must already be initialized when this function is called.

备注： esp_task_wdt_reconfigure() must not be called by multiple tasks simultaneously.

参数 **config** -- **[in]** Configuration structure

返回

- ESP_OK: Reconfiguring was successful
- ESP_ERR_INVALID_STATE: TWDT not initialized yet
- Other: Failed to initialize TWDT

esp_err_t **esp_task_wdt_deinit**(void)

Deinitialize the Task Watchdog Timer (TWDT)

This function will deinitialize the TWDT, and unsubscribe any idle tasks. Calling this function whilst other tasks are still subscribed to the TWDT, or when the TWDT is already deinitialized, will result in an error code being returned.

备注： esp_task_wdt_deinit() must not be called by multiple tasks simultaneously.

返回

- ESP_OK: TWDT successfully deinitialized
- Other: Failed to deinitialize TWDT

esp_err_t **esp_task_wdt_add**(*TaskHandle_t* task_handle)

Subscribe a task to the Task Watchdog Timer (TWDT)

This function subscribes a task to the TWDT. Each subscribed task must periodically call esp_task_wdt_reset() to prevent the TWDT from elapsing its timeout period. Failure to do so will result in a TWDT timeout.

参数 **task_handle** -- Handle of the task. Input NULL to subscribe the current running task to the TWDT

返回

- ESP_OK: Successfully subscribed the task to the TWDT
- Other: Failed to subscribe task

esp_err_t **esp_task_wdt_add_user**(const char *user_name, *esp_task_wdt_user_handle_t* *user_handle_ret)

Subscribe a user to the Task Watchdog Timer (TWDT)

This function subscribes a user to the TWDT. A user of the TWDT is usually a function that needs to run periodically. Each subscribed user must periodically call esp_task_wdt_reset_user() to prevent the TWDT from elapsing its timeout period. Failure to do so will result in a TWDT timeout.

参数

- **user_name** -- **[in]** String to identify the user
- **user_handle_ret** -- **[out]** Handle of the user

返回

- ESP_OK: Successfully subscribed the user to the TWDT
- Other: Failed to subscribe user

esp_err_t **esp_task_wdt_reset**(void)

Reset the Task Watchdog Timer (TWDT) on behalf of the currently running task.

This function will reset the TWDT on behalf of the currently running task. Each subscribed task must periodically call this function to prevent the TWDT from timing out. If one or more subscribed tasks fail to reset the TWDT on their own behalf, a TWDT timeout will occur.

返回

- ESP_OK: Successfully reset the TWDT on behalf of the currently running task
- Other: Failed to reset

esp_err_t **esp_task_wdt_reset_user**(*esp_task_wdt_user_handle_t* user_handle)

Reset the Task Watchdog Timer (TWDT) on behalf of a user.

This function will reset the TWDT on behalf of a user. Each subscribed user must periodically call this function to prevent the TWDT from timing out. If one or more subscribed users fail to reset the TWDT on their own behalf, a TWDT timeout will occur.

参数 **user_handle** -- **[in]** User handle

- ESP_OK: Successfully reset the TWDT on behalf of the user
- Other: Failed to reset

esp_err_t **esp_task_wdt_delete**(*TaskHandle_t* task_handle)

Unsubscribes a task from the Task Watchdog Timer (TWDT)

This function will unsubscribe a task from the TWDT. After being unsubscribed, the task should no longer call esp_task_wdt_reset().

参数 **task_handle** -- **[in]** Handle of the task. Input NULL to unsubscribe the current running task.

返回

- ESP_OK: Successfully unsubscribed the task from the TWDT
- Other: Failed to unsubscribe task

esp_err_t **esp_task_wdt_delete_user**(*esp_task_wdt_user_handle_t* user_handle)

Unsubscribes a user from the Task Watchdog Timer (TWDT)

This function will unsubscribe a user from the TWDT. After being unsubscribed, the user should no longer call esp_task_wdt_reset_user().

参数 **user_handle** -- **[in]** User handle

返回

- ESP_OK: Successfully unsubscribed the user from the TWDT
- Other: Failed to unsubscribe user

esp_err_t **esp_task_wdt_status**(*TaskHandle_t* task_handle)

Query whether a task is subscribed to the Task Watchdog Timer (TWDT)

This function will query whether a task is currently subscribed to the TWDT, or whether the TWDT is initialized.

参数 **task_handle** -- **[in]** Handle of the task. Input NULL to query the current running task.

返回 :

- ESP_OK: The task is currently subscribed to the TWDT
- ESP_ERR_NOT_FOUND: The task is not subscribed
- ESP_ERR_INVALID_STATE: TWDT was never initialized

void **esp_task_wdt_isr_user_handler**(void)

User ISR callback placeholder.

This function is called by task_wdt_isr function (ISR for when TWDT times out). It can be defined in user code to handle TWDT events.

备注: It has the same limitations as the interrupt function. Do not use ESP_LOGx functions inside.

---

esp_err_t **esp_task_wdt_print_triggered_tasks**(*task_wdt_msg_handler* msg_handler, void *opaque, int *cpus_fail)

Prints or retrieves information about tasks/users that triggered the Task Watchdog Timeout.

This function provides various operations to handle tasks/users that did not reset the Task Watchdog in time. It can print detailed information about these tasks/users, such as their names, associated CPUs, and whether they have been reset. Additionally, it can retrieve the total length of the printed information or the CPU affinity of the failing tasks.

备注:

- If `msg_handler` is not provided, the information will be printed to console using ESP_EARLY_LOGE.
- If `msg_handler` is provided, the function will send the printed information to the provided message handler function.
- If `cpus_fail` is provided, the function will store the CPU affinity of the failing tasks in the provided integer.
- During the execution of this function, logging is allowed in critical sections, as TWDT timeouts are considered fatal errors.

参数

- **msg_handler** -- **[in]** Optional message handler function that will be called for each printed line.
- **opaque** -- **[in]** Optional pointer to opaque data that will be passed to the message handler function.
- **cpus_fail** -- **[out]** Optional pointer to an integer where the CPU affinity of the failing tasks will be stored.

返回

- ESP_OK: The function executed successfully.
- ESP_FAIL: No triggered tasks were found, and thus no information was printed or retrieved.

Structures

struct **esp_task_wdt_config_t**

Task Watchdog Timer (TWDT) configuration structure.

Public Members

uint32_t **timeout_ms**

TWDT timeout duration in milliseconds

uint32_t **idle_core_mask**

Bitmask of the core whose idle task should be subscribed on initialization where 1 « i means that core i's idle task will be monitored by the TWDT

bool **trigger_panic**

Trigger panic when timeout occurs

Type Definitions

typedef struct esp_task_wdt_user_handle_s ***esp_task_wdt_user_handle_t**

Task Watchdog Timer (TWDT) user handle.

typedef void (***task_wdt_msg_handler**)(void *opaque, const char *msg)

此部分 API 代码示例存放在 ESP-IDF 示例项目的 system 目录下。

Chapter 3

H/W 硬件参考

Chapter 4

API 指南

4.1 应用层跟踪库

4.1.1 概述

ESP-IDF 中提供了应用层跟踪功能，用于分析应用程序的行为。这一功能在相应的库中实现，可以通过 menuconfig 开启。此功能允许用户在程序运行开销很小的前提下，通过 JTAG、UART 或 USB 接口在主机和 ESP32-P4 之间传输任意数据。用户也可同时使用 JTAG 和 UART 接口。UART 接口主要用于连接 SEGGER SystemView 工具（参见 SystemView）。

开发人员可以使用这一功能库将应用程序的运行状态发送给主机，在运行时接收来自主机的命令或者其他类型的信息。该库的主要使用场景有：

1. 收集来自特定应用程序的数据。具体请参阅特定应用程序的跟踪。
2. 记录到主机的轻量级日志。具体请参阅记录日志到主机。
3. 系统行为分析。具体请参阅基于 *SEGGER SystemView* 的系统行为分析。
4. 获取源代码覆盖率。具体请参阅*Gcov*（源代码覆盖）。

使用 JTAG 接口的跟踪组件工作示意图如下所示：

4.1.2 运行模式

该库支持两种运行模式：

后验模式： 后验模式为默认模式，该模式不需要和主机进行交互。在这种模式下，跟踪模块不会检查主机是否已经从 *HW UP BUFFER* 缓冲区读走所有数据，而是直接使用新数据覆盖旧数据。如果用户仅对最新的跟踪数据感兴趣，例如想要分析程序在崩溃之前的行为，则推荐使用该模式。主机可以稍后根据用户的请求来读取数据，例如在使用 JTAG 接口的情况下，通过特殊的 OpenOCD 命令进行读取。

流模式： 当主机连接到 ESP32-P4 时，跟踪模块会进入此模式。在这种模式下，跟踪模块在新数据写入 *HW UP BUFFER* 之前会检查其中是否有足够的空间，并在必要的时候等待主机读取数据并释放足够的内存。最大等待时间是由用户传递给相应 API 函数的超时时间参数决定的。因此当应用程序尝试使用有限的最大等待时间值来将数据写入跟踪缓冲区时，这些数据可能会被丢弃。尤其需要注意的是，如果在对时效要求严格的代码中（如中断处理函数、操作系统调度等）指定了无限的超时时间，将会导致系统故障。为了避免丢失此类关键数据，开发人员可以在 menuconfig 中开启*CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX* 选项，以启用额外的数据缓冲区。此宏还指定了在上述条件下可以缓冲的数据大小，它有助于缓解由于 USB 总线拥塞等原因导致的向主机传输数据间歇性减缓的状况。但是，当跟踪数据流的平均比特率超出硬件接口的能力时，该选项无法发挥作用。

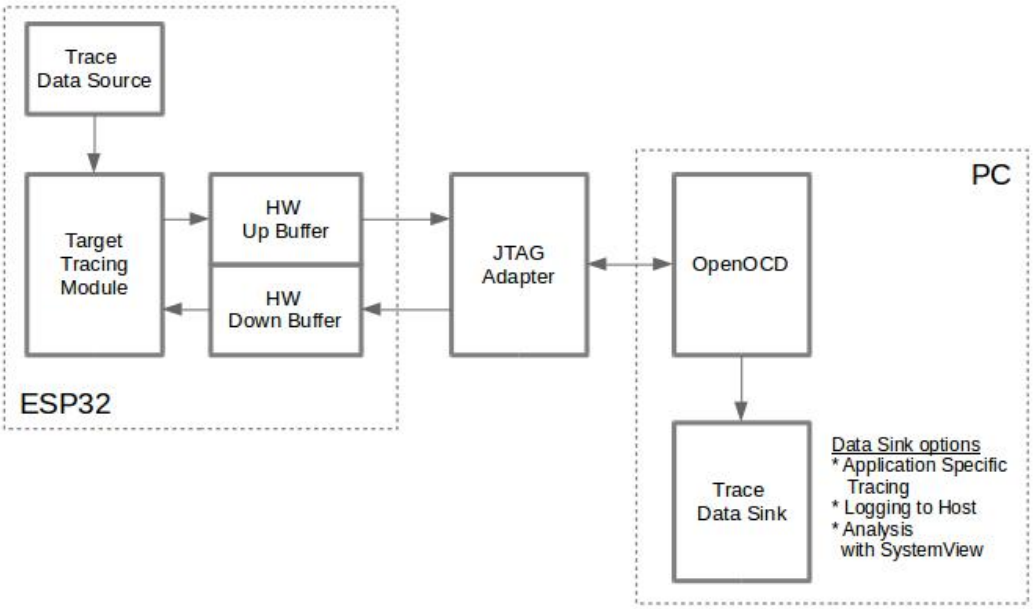


图 1: 使用 JTAG 接口的跟踪组件

4.1.3 配置选项与依赖项

使用此功能需要在主机端和目标端进行以下配置：

1. **主机端：** 应用程序跟踪通过 JTAG 来完成，因此需要在主机上安装并运行 OpenOCD。详细信息请参阅*JTAG 调试*。
2. **目标端：** 在 menuconfig 中开启应用程序跟踪功能。前往 `Component config > Application Level Tracing` 菜单，选择跟踪数据的传输目标（具体用于传输的硬件接口：JTAG 和/或 UART），选择任一非 None 的目标都会自动开启 `CONFIG_APPTRACE_ENABLE` 这个选项。对于 UART 接口，用户必须定义波特率、TX 和 RX 管脚及其他相关参数。

备注： 为了实现更高的数据速率并降低丢包率，建议优化 JTAG 的时钟频率，使其达到能够稳定运行的最大值。详细信息请参阅*优化 JTAG 的速度*。

以下为前述未提及的另外两个 menuconfig 选项：

1. *Threshold for flushing last trace data to host on panic* (*CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH*)。使用 JTAG 接口时，此选项是必选项。在该模式下，跟踪数据以 16 KB 数据块的形式暴露给主机。在后验模式中，一个块被填充后会被暴露给主机，同时之前的块不再可用。也就是说，跟踪数据以 16 KB 的粒度进行覆盖。发生 Panic 时，当前输入块的最新数据将会被暴露给主机，主机可以读取数据以进行后续分析。如果系统发生 Panic 时，仍有少量数据还没来得及暴露给主机，那么之前收集的 16 KB 数据将丢失，主机只能获取少部分的最新跟踪数据，从而可能无法诊断问题。此 menuconfig 选项有助于避免此类情况，它可以控制发生 Panic 时刷新数据的阈值。例如，用户可以设置需要不少于 512 字节的最新跟踪数据，如果在发生 Panic 时待处理的数据少于 512 字节，则数据不会被刷新，也不会覆盖之前的 16 KB 数据。该选项仅在后验模式和使用 JTAG 工作时可发挥作用。
2. *Timeout for flushing last trace data to host on panic* (*CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO*)。该选项仅在流模式下才可发挥作用，它可用于控制跟踪模块在发生 Panic 时等待主机读取最新数据的最长时间。
3. *UART RX/TX ring buffer size* (*CONFIG_APPTRACE_UART_TX_BUFF_SIZE*)。缓冲区的大小取决于通过 UART 传输的数据量。

4. *UART TX message size* (: ref:*CONFIG_APPTRACE_UART_TX_MSG_size*)。要传输的单条消息的最大尺寸。

4.1.4 如何使用此库

该库提供了用于在主机和 ESP32-P4 之间传输任意数据的 API。在 menuconfig 中启用该库后，目标应用程序的跟踪模块会在系统启动时自动初始化。因此，用户需要做的就是调用相应的 API 来发送、接收或者刷新数据。

特定应用程序的跟踪

通常，用户需要决定在每个方向上待传输数据的类型以及如何解析（处理）这些数据。要想在目标和主机之间传输数据，则需执行以下几个步骤：

1. 在目标端，用户需要实现将跟踪数据写入主机的算法。下面的代码片段展示了如何执行此操作。

```
#include "esp_app_trace.h"
...
char buf[] = "Hello World!";
esp_err_t res = esp_apptrace_write(ESP_APPTRACE_DEST_TRAX, buf, strlen(buf),␣
↪ESP_APPTRACE_TMO_INFINITE);
if (res != ESP_OK) {
    ESP_LOGE(TAG, "Failed to write data to host!");
    return res;
}
```

`esp_apptrace_write()` 函数使用 memcpy 把用户数据复制到内部缓存中。在某些情况下，使用 `esp_apptrace_buffer_get()` 和 `esp_apptrace_buffer_put()` 函数会更加理想，它们允许开发人员自行分配缓冲区并填充。下面的代码片段展示了如何执行此操作。

```
#include "esp_app_trace.h"
...
int number = 10;
char *ptr = (char *)esp_apptrace_buffer_get(ESP_APPTRACE_DEST_TRAX, 32, 100/
↪*tmo in us*/);
if (ptr == NULL) {
    ESP_LOGE(TAG, "Failed to get buffer!");
    return ESP_FAIL;
}
sprintf(ptr, "Here is the number %d", number);
esp_err_t res = esp_apptrace_buffer_put(ESP_APPTRACE_DEST_TRAX, ptr, 100/*tmo␣
↪in us*/);
if (res != ESP_OK) {
    /* in case of error host tracing tool (e.g. OpenOCD) will report␣
↪incomplete user buffer */
    ESP_LOGE(TAG, "Failed to put buffer!");
    return res;
}
```

另外，根据实际项目的需要，用户可能希望从主机接收数据。下面的代码片段展示了如何执行此操作。

```
#include "esp_app_trace.h"
...
char buf[32];
char down_buf[32];
size_t sz = sizeof(buf);

/* config down buffer */
esp_apptrace_down_buffer_config(down_buf, sizeof(down_buf));
/* check for incoming data and read them if any */
```

(下页继续)

(续上页)

```
esp_err_t res = esp_apptrace_read(ESP_APPTRACE_DEST_TRAX, buf, &sz, 0/*do not␣
↪wait*/);
if (res != ESP_OK) {
    ESP_LOGE(TAG, "Failed to read data from host!");
    return res;
}
if (sz > 0) {
    /* we have data, process them */
    ...
}
```

esp_apptrace_read() 函数使用 memcpy 把主机端的数据复制到用户缓存区。在某些情况下，使用 esp_apptrace_down_buffer_get() 和 esp_apptrace_down_buffer_put() 函数可能更为理想。它们允许开发人员占用一块读缓冲区并就地进行有关处理操作。下面的代码片段展示了如何执行此操作。

```
#include "esp_app_trace.h"
...
char down_buf[32];
uint32_t *number;
size_t sz = 32;

/* config down buffer */
esp_apptrace_down_buffer_config(down_buf, sizeof(down_buf));
char *ptr = (char *)esp_apptrace_down_buffer_get(ESP_APPTRACE_DEST_TRAX, &sz,␣
↪100/*tmo in us*/);
if (ptr == NULL) {
    ESP_LOGE(TAG, "Failed to get buffer!");
    return ESP_FAIL;
}
if (sz > 4) {
    number = (uint32_t *)ptr;
    printf("Here is the number %d", *number);
} else {
    printf("No data");
}
esp_err_t res = esp_apptrace_down_buffer_put(ESP_APPTRACE_DEST_TRAX, ptr, 100/
↪*tmo in us*/);
if (res != ESP_OK) {
    /* in case of error host tracing tool (e.g. OpenOCD) will report␣
↪incomplete user buffer */
    ESP_LOGE(TAG, "Failed to put buffer!");
    return res;
}
```

2. 下一步是编译应用程序的镜像，并将其下载到目标板上。这一步可以参考文档构建并烧写。
3. 运行 OpenOCD（参见*JTAG 调试*）。
4. 连接到 OpenOCD 的 telnet 服务器。用户可在终端执行命令 telnet <oocd_host> 4444。如果用户是在运行 OpenOCD 的同一台机器上打开 telnet 会话，可以使用 localhost 替换上面命令中的 <oocd_host>。
5. 使用特殊的 OpenOCD 命令开始收集待跟踪的命令。此命令将传输跟踪数据并将其重定向到指定的文件或套接字（当前仅支持文件作为跟踪数据目标）。相关命令的说明，请参阅启动调试器。
6. 最后，处理接收到的数据。由于数据格式由用户自己定义，本文档中省略数据处理的具体流程。数据处理的范例可以参考位于 $IDF_PATH/tools/esp_app_trace 下的 Python 脚本 apptrace_proc.py（用于功能测试）和 logtrace_proc.py（请参阅记录日志到主机 章节中的详细信息）。

OpenOCD 应用程序跟踪命令 *HW UP BUFFER* 在用户数据块之间共享，并且会代替 API 调用者（在任务或者中断上下文中）填充分配到的内存。在多线程环境中，正在填充缓冲区的任务/中断可能会被另一个高优先级的任务/中断抢占，因此主机可能会读取到还未准备好的用户数据。对此，跟踪模块在所有用户数据块之前添加一个数据头，其中包含有分配的用户缓冲区的大小（2 字节）和实际写入的数据长度

(2 字节)，也就是说数据头总共长 4 字节。负责读取跟踪数据的 OpenOCD 命令在读取到不完整的用户数据块时会报错，但是无论如何，它都会将整个用户数据块（包括还未填充的区域）的内容放到输出文件中。

下文介绍了如何使用 OpenOCD 应用程序跟踪命令。

备注： 目前，OpenOCD 还不支持将任意用户数据发送到目标的命令。

命令用法：

```
esp apptrace [start <options>] | [stop] | [status] | [dump <cores_num>
<outfile>]
```

子命令：

start 开始跟踪（连续流模式）。
stop 停止跟踪。
status 获取跟踪状态。
dump 转储所有后验模式的数据。

Start 子命令的语法：

```
start <outfile> [poll_period [trace_size [stop_tmo [wait4halt
[skip_size]]]]
```

outfile 用于保存来自两个 CPU 的数据文件的路径，该参数需要具有以下格式：`file://path/to/file`。
poll_period 轮询跟踪数据的周期（单位：毫秒），如果大于 0 则以非阻塞模式运行。默认为 1 毫秒。
trace_size 最多要收集的数据量（单位：字节），接收到指定数量的数据后将会停止跟踪。默认为 -1（禁用跟踪大小停止触发器）。
stop_tmo 空闲超时（单位：秒），如果指定的时间段内都没有数据就会停止跟踪。默认为 -1（禁用跟踪超时停止触发器）。还可以将其设置为比目标跟踪命令之间的最长暂停值更长的值（可选）。
wait4halt 如果设置为 0 则立即开始跟踪，否则命令会先等待目标停止（复位、打断点等），然后对其进行自动恢复并开始跟踪。默认值为 0。
skip_size 开始时要跳过的字节数，默认为 0。

备注： 如果 `poll_period` 为 0，则在跟踪停止之前，OpenOCD 的 telnet 命令将不可用。必须通过复位电路板或者在 OpenOCD 的窗口中（非 telnet 会话窗口）使用快捷键 Ctrl+C。另一种选择是设置 `trace_size` 并等待，当收集到指定数据量时，跟踪会自动停止。

命令使用示例：

1. 将 2048 个字节的跟踪数据收集到 `trace.log` 文件中，该文件将保存在 `openocd-esp32` 目录中。

```
esp apptrace start file://trace.log 1 2048 5 0 0

跟踪数据会被检索并以非阻塞的模式保存到文件中，如果收集满 2048 字节的数据或者在␣
↪5 秒内都没有新的数据，那么该过程就会停止。

.. note::

    在将数据提供给 OpenOCD 之前，会对其进行缓冲。如果看到 “Data timeout!” ␣
↪的消息，则表示目标可能在超时之前没有向 OpenOCD␣
↪发送足够的数据以清空缓冲区。要解决这个问题，可以增加超时时间或者使用函数␣
↪``esp_apptrace_flush()`` 以特定间隔刷新数据。
```

2. 在非阻塞模式下无限地检索跟踪数据。

```
esp apptrace start file://trace.log 1 -1 -1 0 0

对收集数据的大小没有限制，也不设置超时时间。要停止此过程，可以在 OpenOCD 的␣
↪telnet 会话窗口中发送 ``esp apptrace stop`` 命令，或者在 OpenOCD␣
↪窗口中使用快捷键 Ctrl+C。
```

(下页继续)

(续上页)

3. 检索跟踪数据并无限期保存。

```
esp apptrace start file://trace.log 0 -1 -1 0 0

在跟踪停止之前，OpenOCD 的 telnet 会话窗口将不可用。要停止跟踪，请在 OpenOCD␣
↪的窗口中使用快捷键 Ctrl+C。
```

4. 等待目标停止，然后恢复目标的操作并开始检索数据。当收集满 2048 字节的数据后就停止：

```
esp apptrace start file://trace.log 0 2048 -1 1 0

想要复位后立即开始跟踪，请使用 OpenOCD 的 ``reset halt`` 命令。
```

记录日志到主机

记录日志到主机是 ESP-IDF 中一个非常实用的功能：通过应用层跟踪库将日志保存到主机端。某种程度上，这也算是一种半主机 (semihosting) 机制，相较于调用 `ESP_LOGx` 将待打印的字符串发送到 UART 的日志记录方式，此功能将大部分工作转移到了主机端，从而减少了本地工作量。

ESP-IDF 的日志库会默认使用类 vprintf 的函数将格式化的字符串输出到专用的 UART，一般来说涉及以下几个步骤：

1. 解析格式字符串以获取每个参数的类型。
2. 根据其类型，将每个参数都转换为字符串。
3. 格式字符串与转换后的参数一起发送到 UART。

虽然可以对类 vprintf 函数进行一定程度的优化，但由于在任何情况下都必须执行上述步骤，并且每个步骤都会消耗一定的时间（尤其是步骤 3），所以经常会发生以下这种情况：向程序中添加额外的打印信息以诊断问题，却改变了应用程序的行为，使得问题无法复现。在最严重的情况下，程序无法正常工作，最终导致报错甚至挂起。

想要解决此类问题，可以使用更高的波特率或者其他更快的接口，并将字符串格式化的工作转移到主机端。

通过应用层跟踪库的 `esp_apptrace_vprintf` 函数，可以将日志信息发送到主机，该函数不执行格式字符串和参数的完全解析，而仅仅计算传递参数的数量，并将它们与格式字符串地址一起发送给主机。主机端会通过一个特殊的 Python 脚本来处理并打印接收到的日志数据。

局限 目前通过 JTAG 实现记录日志还存在以下几点局限：

1. 不支持使用 `ESP_EARLY_LOGx` 宏进行跟踪。
2. 不支持大小超过 4 字节的 printf 参数（例如 `double` 和 `uint64_t`）。
3. 仅支持.rodata 段中的格式字符串和参数。
4. 最多支持 256 个 printf 参数。

如何使用 为了使用跟踪模块来记录日志，用户需要执行以下步骤：

1. 在目标端，需要安装特殊的类 vprintf 函数*esp_apptrace_vprintf()*，该函数负责将日志数据发送给主机，使用方法为 `esp_log_set_vprintf(esp_apptrace_vprintf);`。如需将日志数据再次重定向给 UART，请使用 `esp_log_set_vprintf(vprintf);`。
2. 按照特定应用程序的跟踪 章节中的第 2-5 步进行操作。
3. 打印接收到的日志记录，请在终端运行以下命令：`$IDF_PATH/tools/esp_app_trace/logtrace_proc.py /path/to/trace/file /path/to/program/elf/file`。

Log Trace Processor 命令选项 命令用法：

```
logtrace_proc.py [-h] [--no-errors] <trace_file> <elf_file>
```

位置参数（必要）：

trace_file 日志跟踪文件的路径。
elf_file 程序 ELF 文件的路径。

可选参数：

-h, --help 显示此帮助信息并退出。
--no-errors, -n 不打印错误信息。

基于 SEGGER SystemView 的系统行为分析

ESP-IDF 中另一个基于应用层跟踪库的实用功能是系统级跟踪，它会生成与 SEGGER SystemView 工具相兼容的跟踪信息。SEGGER SystemView 是一款实时记录和可视化工具，用来分析应用程序运行时的行为，可通过 UART 接口实时查看事件。

如何使用 若需使用这个功能，需要在 menuconfig 中开启*CONFIG_APPTRACE_SV_ENABLE* 选项，具体路径为 `Component config > Application Level Tracing > FreeRTOS SystemView Tracing`。同一菜单栏下还开启了其它几个选项：

1. *SytemView destination*。选择需要使用的接口：JTAG 或 UART。使用 UART 接口时，可以将 SystemView 应用程序直接连接到 ESP32-P4 并实时接收数据。
2. *ESP32-P4 timer to use as SystemView timestamp source*（*CONFIG_APPTRACE_SV_TS_SOURCE*）。选择 SystemView 事件使用的时间戳来源。在单核模式下，使用 ESP32-P4 内部的循环计数器生成时间戳，其最大的工作频率是 240 MHz（时间戳粒度大约为 4 ns）。在双核模式下，使用工作在 40 MHz 的外部定时器，因此时间戳粒度为 25 ns。
3. 可以单独启用或禁用的 SystemView 事件集合 (`CONFIG_APPTRACE_SV_EVT_XXX`)：
   - Trace Buffer Overflow Event
   - ISR Enter Event
   - ISR Exit Event
   - ISR Exit to Scheduler Event
   - Task Start Execution Event
   - Task Stop Execution Event
   - Task Start Ready State Event
   - Task Stop Ready State Event
   - Task Create Event
   - Task Terminate Event
   - System Idle Event
   - Timer Enter Event
   - Timer Exit Event

ESP-IDF 中已经包含了所有用于生成兼容 SystemView 跟踪信息的代码，用户只需配置必要的项目选项（如上所示），然后构建、烧写映像到目标板，接着参照前面的介绍，使用 OpenOCD 收集数据。

4. 想要通过 UART 接口进行实时跟踪，请在菜单配置选项 `Component config > Application Level Tracing > FreeRTOS SystemView Tracing` 中选择 Pro 或 App CPU。

OpenOCD SystemView 跟踪命令选项 命令用法：

```
esp sysview [start <options>] | [stop] | [status]
```

子命令：

start 开启跟踪（连续流模式）。
stop 停止跟踪。
status 获取跟踪状态。

Start 子命令语法：

```
start <outfile1> [outfile2] [poll_period [trace_size [stop_tmo]]]
```

outfile1 保存 PRO CPU 数据的文件路径。此参数需要具有如下格式：`file://path/to/file`。
outfile2 保存 APP CPU 数据的文件路径。此参数需要具有如下格式：`file://path/to/file`。

poll_period 跟踪数据的轮询周期（单位：毫秒）。如果该值大于 0，则命令以非阻塞的模式运行。默认为 1 毫秒。
trace_size 最多要收集的数据量（单位：字节）。当收到指定数量的数据后，将停止跟踪。默认值是 -1（禁用跟踪大小停止触发器）。
stop_tmo 空闲超时（单位：秒）。如果指定的时间内没有数据，将停止跟踪。默认值是 -1（禁用跟踪超时停止触发器）。

备注： 如果 `poll_period` 为 0，则在跟踪停止之前，OpenOCD 的 telnet 命令行将不可用。你需要复位板卡，或者在 OpenOCD 的窗口（非 telnet 会话窗口）输入 Ctrl+C 命令，手动停止跟踪。另一个办法是设置 `trace_size`，等到收集满指定数量的数据后自动停止跟踪。

命令使用示例：

1. 将 SystemView 跟踪数据收集到文件 `pro-cpu.SVDat` 和 `pro-cpu.SVDat` 中。这些文件会被保存在 `openocd-esp32` 目录中。

   ```
   esp sysview start file://pro-cpu.SVDat file://app-cpu.SVDat
   ```

   跟踪数据被检索并以非阻塞的方式保存。要停止此过程，需要在 OpenOCD 的 telnet 会话窗口输入 `esp sysview stop` 命令，也可以在 OpenOCD 窗口中按下快捷键 Ctrl+C。
2. 检索跟踪数据并无限保存。

   ```
   esp32 sysview start file://pro-cpu.SVDat file://app-cpu.SVDat 0 -1 -1
   ```

   OpenOCD 的 telnet 命令行在跟踪停止前会无法使用，要停止跟踪，请在 OpenOCD 窗口使用 Ctrl+C 快捷键。

数据可视化 收集到跟踪数据后，用户可以使用特殊的工具对结果进行可视化并分析程序行为。

遗憾的是，SystemView 不支持从多个核心进行跟踪。所以当使用 JTAG 追踪双核模式下的 ESP32-P4 时会生成两个文件：一个用于 PRO CPU，另一个用于 APP CPU。用户可以将每个文件加载到工具中单独分析。使用 UART 进行追踪时，用户可以在 menuconfig Pro 或 App 中点击 `Component config` > `Application Level Tracing` > `FreeRTOS SystemView Tracing` 并选择要追踪的 CPU。

在工具中单独分析每个核的跟踪数据是比较棘手的，但是 Eclipse 提供了 *Impulse* 插件，该插件可以加载多个跟踪文件，并且可以在同一视图中检查来自两个内核的事件。此外，与免费版的 SystemView 相比，此插件没有 1,000,000 个事件的限制。

关于如何安装、配置 Impulse 并使用它来可视化来自单个核心的跟踪数据，请参阅 官方教程 。

备注： ESP-IDF 使用自己的 SystemView FreeRTOS 事件 ID 映射，因此用户需要将 `$SYSVIEW_INSTALL_DIR/Description/SYSVIEW_FreeRTOS.txt` 替换成 `$IDF_PATH/tools/esp_app_trace/SYSVIEW_FreeRTOS.txt`。在使用上述链接配置 SystemView 序列化程序时，也应该使用该特定文件的内容。

配置 Impulse 实现双核跟踪 在安装好 Impulse 插件并确保 Impulse 能够在单独的选项卡中成功加载每个核心的跟踪文件后，用户可以添加特殊的 Multi Adapter 端口并将这两个文件加载到一个视图中。为此，用户需要在 Eclipse 中执行以下操作：

1. 打开 `Signal Ports` 视图，前往 `Windows` > `Show View` > `Other` 菜单，在 Impulse 文件夹中找到 `Signal Ports` 视图并双击。
2. 在 `Signal Ports` 视图中，右键 `Ports` 并选择 `Add`，然后选择 `New Multi Adapter Port`。
3. 在打开的对话框中按下 `add` 按钮，选择 `New Pipe/File`。
4. 在打开的对话框中选择 `SystemView Serializer` 并设置 PRO CPU 跟踪文件的路径，按下 `OK` 保存设置。
5. 对 APP CPU 的跟踪文件重复步骤 3 和 4。
6. 双击创建的端口，会打开此端口的视图。
7. 单击 `Start/Stop Streaming` 按钮，数据将会被加载。

8. 使用 Zoom Out，Zoom In 和 Zoom Fit 按钮来查看数据。
9. 有关设置测量光标和其他的功能，请参阅 Impulse 官方文档 。

备注： 如果你在可视化方面遇到了问题（未显示数据或者缩放操作异常），可以尝试删除当前的信号层次结构，再双击必要的文件或端口。Eclipse 会请求创建新的信号层次结构。

Gcov（源代码覆盖）

Gcov 和 Gcovr 简介 源代码覆盖率显示程序运行时间内执行的每一条程序执行路径的数量和频率。Gcov 是一款 GCC 工具，与编译器协同使用时，可生成日志文件，显示源文件每行的执行次数。Gcovr 是管理 Gcov 和生成代码覆盖率总结的工具。

一般来说，使用 Gcov 在主机上编译和运行程序会经过以下步骤：

1. 使用 GCC 以及 `--coverage` 选项编译源代码。编译器会在编译过程中生成一个 `.gcno` 注释文件，该文件包含重建执行路径块图以及将每个块映射到源代码行号等信息。每个用 `--coverage` 选项编译的源文件都会生成自己的同名 `.gcno` 文件（如 `main.c` 在编译时会生成 `main.gcno`）。
2. 执行程序。在执行过程中，程序会生成 `.gcda` 数据文件。这些数据文件包含了执行路径的次数统计。程序将为每个用 `--coverage` 选项编译的源文件生成一个 `.gcda` 文件（如 `main.c` 将生成 `main.gcda`）。
3. Gcov 或 Gcovr 可用于生成基于 `.gcno`、`.gcda` 和源文件的代码覆盖。Gcov 将以 `.gcov` 文件的形式为每个源文件生成基于文本的覆盖报告，而 Gcovr 将以 HTML 格式生成覆盖报告。

ESP-IDF 中的 Gcov 和 Gcovr 应用 在 ESP-IDF 中使用 Gcov 的过程比较复杂，因为程序不在主机上运行，而在目标机上运行。代码覆盖率数据（即 `.gcda` 文件）最初存储在目标机上，OpenOCD 在运行时通过 JTAG 将代码覆盖数据从目标机转储到主机上。在 ESP-IDF 中使用 Gcov 可以分为以下几个步骤：

1. *为 Gcov 设置项目*
2. *转储代码覆盖数据*
3. *生成代码覆盖报告*

为 Gcov 设置项目

编译器选项 为了获取项目中的代码覆盖率数据，必须用 `--coverage` 选项编译项目中的一个或多个源文件。在 ESP-IDF 中，这可以在组件级或单个源文件级实现：

- 在组件的 `CMakeLists.txt` 文件中添加 `target_compile_options(${COMPONENT_LIB} PRIVATE --coverage)` 可确保使用 `--coverage` 选项编译组件中的所有源文件。
- 在组件的 `CMakeLists.txt` 文件中添加 `set_source_files_properties(source1.c source2.c PROPERTIES COMPILE_FLAGS --coverage)` 可确保使用 `--coverage` 选项编译同一组件中选定的一些源文件（如 `source1.c` 和 `source2.c`）。

当一个源文件用 `--coverage` 选项编译时（例如 `gcov_example.c`），编译器会在项目的构建目录下生成 `gcov_example.gcno` 文件。

项目配置 在构建有源代码覆盖的项目之前，请运行 `idf.py menuconfig` 以启用以下项目配置选项。

- 通过*CONFIG_APPTRACE_DESTINATION1* 选项选择 `Trace Memory` 来启用应用程序跟踪模块。
- 通过*CONFIG_APPTRACE_GCOV_ENABLE* 选项启用 Gcov 主机。

转储代码覆盖数据 一旦项目使用 `--coverage` 选项编译并烧录到目标机上，在应用程序运行时，代码覆盖数据将存储在目标机内部（即在跟踪存储器中）。将代码覆盖率数据从目标机转移到主机上的过程称为转储。

覆盖率数据的转储通过 OpenOCD 进行（关于如何设置和运行 OpenOCD，请参考*JTAG 调试*）。由于该过程需要通过向 OpenOCD 发出命令来触发转储，因此必须打开 telnet 会话，以向 OpenOCD 发出这些命令

(运行 `telnet localhost 4444`)。GDB 也可以代替 telnet 来向 OpenOCD 发出命令，但是所有从 GDB 发出的命令都需要以 `mon <oocd_command>` 为前缀。

当目标机转储代码覆盖数据时，`.gcda` 文件存储在项目的构建目录中。例如，如果 `main` 组件的 `gcov_example_main.c` 在编译时使用了 `--coverage` 选项，那么转储代码覆盖数据将在 `build/esp-idf/main/CMakeFiles/__idf_main.dir/gcov_example_main.c.gcda` 中生成 `gcov_example_main.gcda` 文件。注意，编译过程中产生的 `.gcno` 文件也放在同一目录下。

代码覆盖数据的转储可以在应用程序的整个生命周期内多次进行。每次转储都会用最新的代码覆盖信息更新 `.gcda` 文件。代码覆盖数据是累积的，因此最新的数据将包含应用程序整个生命周期中每个代码路径的总执行次数。

ESP-IDF 支持两种将代码覆盖数据从目标机转储到主机的方法：

- 运行中实时转储
- 硬编码转储

运行中实时转储 通过 telnet 会话调用 OpenOCD 命令 `ESP32-P4 gcov` 来触发运行时的实时转储。一旦被调用，OpenOCD 将立即抢占 ESP32-P4 的当前状态，并执行内置的 ESP-IDF Gcov 调试存根函数。调试存根函数将数据转储到主机。完成后，ESP32-P4 将恢复当前状态。

硬编码转储 硬编码转储是由应用程序本身从程序内部调用*esp_gcov_dump()* 函数触发的。在调用时，应用程序将停止并等待 OpenOCD 连接，同时检索代码覆盖数据。一旦*esp_gcov_dump()* 函数被调用，主机将通过 telnet 会话执行 `esp gcov dump` OpenOCD 命令，该命令会将 OpenOCD 连接到 ESP32-P4，检索代码覆盖数据，然后断开与 ESP32-P4 的连接，从而恢复应用程序。在应用程序的生命周期中可多次触发硬编码转储。

在必要时（如应用程序初始化后或是应用程序主循环的每次迭代期间）放置*esp_gcov_dump()*，当应用程序在生命周期的某刻需要代码覆盖率数据时，硬编码转储会非常有用。

GDB 可以用来在*esp_gcov_dump()* 上设置断点，然后使用 `gdbinit` 脚本自动调用 `mon esp gcov dump`（关于 GDB 的使用可参考使用命令行调试）。

以下 GDB 脚本将在*esp_gcov_dump()* 处添加一个断点，然后调用 `mon esp gcov dump` OpenOCD 命令。

```
b esp_gcov_dump
commands
mon esp gcov dump
end
```

备注： 注意，所有的 OpenOCD 命令都应该在 GDB 中以 `mon <oocd_command>` 方式调用。

生成代码覆盖报告 一旦代码覆盖数据被转储，`.gcno`、`.gcda` 和源文件可以用来生成代码覆盖报告。该报告会显示源文件中每行被执行的次数。

Gcov 和 Gcovr 都可以用来生成代码覆盖报告。安装 Xtensa 工具链时会一起安装 Gcov，但 Gcovr 可能需要单独安装。关于如何使用 Gcov 或 Gcovr，请参考 Gcov 文档 和 Gcovr 文档。

在工程中添加 Gcovr 构建目标 用户可以在自己的工程中定义额外的构建目标，从而通过一个简单的构建命令即可更方便地生成报告。

请在工程的 `CMakeLists.txt` 文件中添加以下内容：

```
include($ENV{IDF_PATH}/tools/cmake/gcov.cmake)
idf_create_coverage_report(${CMAKE_CURRENT_BINARY_DIR}/coverage_report)
idf_clean_coverage_report(${CMAKE_CURRENT_BINARY_DIR}/coverage_report)
```

可使用以下命令:

- `cmake --build build/ --target gcovr-report`：在 `$(BUILD_DIR_BASE)/coverage_report/html` 目录下生成 HTML 格式代码覆盖报告。
- `cmake --build build/ --target cov-data-clean`：删除所有代码覆盖数据文件。

4.2 应用程序的启动流程

本文将会介绍 ESP32-P4 从上电到运行 `app_main` 函数中间所经历的步骤（即启动流程）。

宏观上，该启动流程可以分为如下 3 个步骤：

1. *一级引导程序* 被固化在了 ESP32-P4 内部的 ROM 中，它会从 flash 的 0x2000 偏移地址处加载二级引导程序至 RAM (IRAM & DRAM) 中。
2. *二级引导程序* 从 flash 中加载分区表和主程序镜像至内存中，主程序中包含了 RAM 段和通过 flash 高速缓存映射的只读段。
3. *应用程序启动阶段* 运行，这时第二个 CPU 和 RTOS 调度器启动，接着运行 `main_task`，从而执行 `app_main`。

下面会对上述过程进行更为详细的阐述。

4.2.1 一级引导程序

SoC 复位后，PRO CPU 会立即开始运行，执行复位向量代码，而 APP CPU 仍然保持复位状态。在启动过程中，PRO CPU 会执行所有的初始化操作。APP CPU 的复位状态会在应用程序启动代码的 `call_start_cpu0` 函数中失效。复位向量代码位于 ESP32-P4 芯片掩膜 ROM 处，且不能被修改。

复位向量调用的启动代码会根据 `GPIO_STRAP_REG` 寄存器的值来确定 ESP32-P4 的启动模式，该寄存器保存着复位后 bootstrap 引脚的电平状态。根据不同的复位原因，程序会执行如下操作：

1. 上电复位、软件 SoC 复位、看门狗 SoC 复位：检查 `GPIO_STRAP_REG` 寄存器，判断是否请求自定义启动模式，如 UART 下载模式。如果是，ROM 会执行此自定义加载模式，否则会像软件 CPU 复位一样继续启动。请参考 ESP32-P4 技术规格书了解 SoC 启动模式以及具体执行过程。
2. 软件 CPU 复位、看门狗 CPU 复位：根据 EFUSE 中的值配置 SPI flash，然后尝试从 flash 中加载代码，这部分将会在后面一小节详细介绍。

备注： 正常启动模式下会使能 RTC 看门狗，因此，如果进程中断或停止，看门狗将自动重置 SOC 并重复启动过程。如果 strapping GPIOs 已更改，则可能导致 SoC 陷入新的启动模式。

4.2.2 二级引导程序

在 ESP-IDF 中，存放在 flash 的 0x2000 偏移地址处的二进制镜像就是二级引导程序。二级引导程序的源码可以在 ESP-IDF 的 components/bootloader 目录下找到。ESP-IDF 使用二级引导程序可以增加 flash 分区的灵活性（使用分区表），并且方便实现 flash 加密，安全引导和空中升级 (OTA) 等功能。

当一级引导程序校验并加载完二级引导程序后，它会从二进制镜像的头部找到二级引导程序的入口点，并跳转过去运行。

二级引导程序默认从 flash 的 0x8000 偏移地址处（*可配置的值*）读取分区表。请参考*分区表* 获取详细信息。引导程序会寻找工厂分区和 OTA 应用程序分区。如果在分区表中找到了 OTA 应用程序分区，引导程序将查询 `otadata` 分区以确定应引导哪个分区。更多信息请参考*空中升级 (OTA)*。

关于 ESP-IDF 引导程序可用的配置选项，请参考*引导加载程序 (Bootloader)*。

对于选定的分区，二级引导程序将从 flash 逐段读取二进制镜像：

- 对于在内部*IRAM（指令 RAM）* 或*DRAM（数据 RAM）* 中具有加载地址的段，将把数据从 flash 复制到它们的加载地址处。
- 对于一些加载地址位于*DROM（数据存储在 flash 中）* 或*IROM（代码从 flash 中运行）* 区域的段，通过配置 flash MMU，可为从 flash 到加载地址提供正确的映射。

一旦处理完所有段（即加载了代码并设置了 flash MMU），二级引导程序将验证应用程序的完整性，并从二进制镜像文件的头部寻找入口地址，然后跳转到该地址处运行。

4.2.3 应用程序启动阶段

应用程序启动包含了从应用程序开始执行到 `app_main` 函数在主任务内部运行前的所有过程。可分为三个阶段：

- 硬件和基本 C 语言运行环境的端口初始化。
- 软件服务和 FreeRTOS 的系统初始化。
- 运行主任务并调用 `app_main`。

备注： 通常不需要了解 ESP-IDF 应用程序初始化的所有阶段。如果需要仅从应用程序开发人员的角度了解初始化，请跳至运行主任务。

端口初始化

ESP-IDF 应用程序的入口是 components/esp_system/port/cpu_start.c 文件中的 `call_start_cpu0` 函数。这个函数由二级引导加载程序执行，并且从不返回。

该端口层的初始化功能会初始化基本的 C 运行环境 (”CRT”)，并对 SoC 的内部硬件进行了初始配置。

- 为应用程序重新配置 CPU 异常（允许应用程序中断处理程序运行，并使用为应用程序配置的选项来处理严重错误，而不是使用 ROM 提供的简易版错误处理程序处理。
- 如果没有设置选项*CONFIG_BOOTLOADER_WDT_ENABLE*，则不使能 RTC 看门狗定时器。
- 初始化内部存储器（数据和 bss）。
- 完成 MMU 高速缓存配置。
- 如果配置了 PSRAM，则使能 PSRAM。
- 将 CPU 时钟设置为项目配置的频率。
- 如果应用程序被配置为在多个内核上运行，则启动另一个内核并等待其初始化（在类似的“端口层”初始化函数 `call_start_cpu1` 内）。

`call_start_cpu0` 完成运行后，将调用在 components/esp_system/startup.c 中找到的“系统层”初始化函数 `start_cpu0`。其他内核也将完成端口层的初始化，并调用同一文件中的 `start_other_cores`。

系统初始化

主要的系统初始化函数是 `start_cpu0`。默认情况下，这个函数与 `start_cpu0_default` 函数弱链接。这意味着可以覆盖这个函数，增加一些额外的初始化步骤。

主要的系统初始化阶段包括：

- 如果默认的日志级别允许，则记录该应用程序的相关信息（项目名称、应用程序版本 等）。
- 初始化堆分配器（在这之前，所有分配必须是静态的或在堆栈上）。
- 初始化 newlib 组件的系统调用和时间函数。
- 配置断电检测器。
- 根据串行控制台配置 设置 libc stdin、stdout、和 stderr。
- 执行与安全有关的检查，包括为该配置烧录 efuse（包括永久限制 *ROM* 下载模式）。
- 初始化 SPI flash API 支持。
- 调用全局 C++ 构造函数和任何标有 `__attribute__((constructor))` 的 C 函数。

二级系统初始化允许单个组件被初始化。如果一个组件有一个用 ESP_SYSTEM_INIT_FN 宏注释的初始化函数，它将作为二级初始化的一部分被调用。

运行主任务

在所有其他组件都初始化后，主任务会被创建，FreeRTOS 调度器开始运行。

做完一些初始化任务后（需要启动调度器），主任务在固件中运行应用程序提供的函数 app_main。

运行 app_main 的主任务有一个固定的 RTOS 优先级（比最小值高）和一个可配置的堆栈大小。

主任务的内核亲和性也是可以配置的，请参考*CONFIG_ESP_MAIN_TASK_AFFINITY*。

与普通的 FreeRTOS 任务（或嵌入式 C 的 main 函数）不同，app_main 任务可以返回。如果 app_main 函数返回，那么主任务将会被删除。系统将继续运行其他的 RTOS 任务。因此可以将 app_main 实现为一个创建其他应用任务然后返回的函数，或主应用任务本身。

APP CPU 的内核启动流程

APP CPU 的启动流程类似但更简单：

当运行系统初始化时，PRO CPU 上的代码会给 APP CPU 设置好入口地址，解除其复位状态，然后等待 APP CPU 上运行的代码设置一个全局标志，以表明 APP CPU 已经正常启动。完成后，APP CPU 跳转到 components/esp_system/port/cpu_start.c 中的 call_start_cpu1 函数。

当 start_cpu0 函数对 PRO CPU 进行初始化的时候，APP CPU 运行 start_cpu_other_cores 函数。与 start_cpu0 函数类似，start_cpu_other_cores 函数是弱链接的，默认为 start_cpu_other_cores_default 函数，但可以由应用程序替换为不同的函数。

start_cpu_other_cores_default 函数做了一些与内核相关的系统初始化，然后等待 PRO CPU 启动 FreeRTOS 的调度器，启动完成后，它会执行 esp_startup_start_app_other_cores 函数，这是另一个默认为 esp_startup_start_app_other_cores_default 的弱链接函数。

默认情况下，esp_startup_start_app_other_cores_default 只会自旋，直到 PRO CPU 上的调度器触发中断，以启动 APP CPU 上的 RTOS 调度器。

4.3 引导加载程序 (Bootloader)

ESP-IDF 软件引导加载程序 (Bootloader) 主要执行以下任务：

1. 内部模块的最小化初始配置；
2. 如果配置了*flash 加密* 和/或*Secure*，则对其进行初始化。
3. 根据分区表和 ota_data（如果存在）选择需要引导的应用程序 (app) 分区；
4. 将此应用程序镜像加载到 RAM（IRAM 和 DRAM）中，最后把控制权转交给此应用程序。

引导加载程序位于 flash 的 0x2000 偏移地址处。

关于启动过程以及 ESP-IDF 引导加载程序的更多信息，请参考应用程序的启动流程。

4.3.1 引导加载程序兼容性

建议使用最新发布的*ESP-IDF 版本*。OTA（空中升级）更新可以在现场烧录新的应用程序，但不能烧录一个新的引导加载程序。因此，引导加载程序支持引导从 ESP-IDF 新版本中构建的应用程序。

但不支持引导从 ESP-IDF 旧版本中构建的程序。如果现有产品可能需要将应用程序降级到旧版本，那么在手动更新 ESP-IDF 时，请继续使用旧版本 ESP-IDF 引导加载程序的二进制文件。

备注： 如果在生产中测试现有产品的 OTA 更新，请确保测试中使用的 ESP-IDF 引导加载程序二进制文件与生产中部署的相同。

配置 SPI flash

每个 ESP-IDF 应用程序或引导加载程序的二进制文件中都包含一个文件头，其中内置了*CONFIG_ESPTOOLPY_FLASHMODE*、*CONFIG_ESPTOOLPY_FLASHFREQ*、和*CONFIG_ESPTOOLPY_FLASHSIZE*。这些是用于在启动时配置 SPI flash。

ROM 中的一级引导程序 从 flash 中读取二级引导程序 文件头中的配置信息，并使用这些信息来加载剩余的二级引导程序。然而，此时系统的时钟速度低于其被配置的速度，并且在这个阶段，只支持部分 flash 模式。因此，当二级引导程序 运行时，它会从当前应用程序的二进制文件头中读取数据（而不是从引导加载程序的文件头中读取数据），并使用这些数据重新配置 flash。这样的配置流程可让 OTA 更新去更改当前使用的 SPI flash 的配置。

4.3.2 日志级别

引导加载程序日志的级别默认为"Info"。通过设置*CONFIG_BOOTLOADER_LOG_LEVEL* 选项，可以增加或减少这个等级。这个日志级别与应用程序中使用的日志级别是分开的（见日志库）。

降低引导加载程序日志的详细程度可以稍微缩短整个项目的启动时间。

4.3.3 恢复出厂设置

在更新出现问题时，最好能有一种方法让设备回到已知的正常状态，这时可选择恢复出厂设置。

要回到原始出厂设置并清除所有用户设置，请在引导加载程序中配置*CONFIG_BOOTLOADER_FACTORY_RESET*。

以下两种方式可以将设备恢复出厂设置。

- 清除一个或多个数据分区。*CONFIG_BOOTLOADER_DATA_FACTORY_RESET* 选项允许用户选择哪些数据分区在恢复出厂设置时需要被擦除。
  用户可以使用以逗号分隔的列表形式指定分区的名称，为了提高可读性，可以选择添加空格（如：`nvs, phy_init, nvs_custom`）。
  请确保选项里指定的分区名称和分区表中的名称相同。此处不能指定"app"类型的分区。
- 从"工厂"应用分区启动。当启用*CONFIG_BOOTLOADER_OTA_DATA_ERASE* 选项，恢复出厂设置后，设备将从默认的"工厂"应用分区启动（如果分区表中没有"工厂"应用分区，则从默认的 OTA 应用分区启动）。这个恢复过程是通过擦除 OTA 数据分区来完成的，OTA 数据分区中保存了当前选择的 OTA 分区槽。"工厂"应用分区槽（如果存在）永远不会通过 OTA 更新，因此重置为从"工厂"应用分区启动则意味着让固件应用程序恢复正常状态。

这两个配置选项都可以独立启用。

此外，以下配置选项用于配置触发恢复出厂设置的条件：

- *CONFIG_BOOTLOADER_NUM_PIN_FACTORY_RESET*- 输入管脚 (GPIO) 的编号，该管脚用于触发恢复出厂设置。必须在重置时将此管脚拉低或拉高（可配置）才能触发出厂重置事件。
- *CONFIG_BOOTLOADER_HOLD_TIME_GPIO*- 管脚电平保持时间（默认为 5 秒）。设备重置后，管脚电平必须保持该设定的时间，才能执行恢复出厂设置或引导测试分区（如适用）。
- *CONFIG_BOOTLOADER_FACTORY_RESET_PIN_LEVEL* - 设置管脚电平高低。设备重置后，根据此设置将管脚拉高或拉低，才能触发出厂重置事件。如果管脚具有内部上拉，则上拉会在管脚采样前生效。有关管脚内部上拉的详细信息，请参考 ESP32-P4 的技术规格书。

如果应用程序需要知道设备是否触发了出厂重置，可以通过调用 `bootloader_common_get_rtc_retain_mem_factory_reset_state()` 函数来确定：

- 如果读取到设备出厂重置状态为 true，会返回状态 true，说明设备已经触发出厂重置。此后会重置状态为 false，以便后续的出厂重置触发判断。

- 如果读取到设备出厂重置状态为 false，会返回状态 false，说明设备并未触发出厂重置，或者保存此状态的内存区域已失效。

同时需要注意该功能需要占用部分 RTC FAST 内存（占用的内存与*CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP* 大小相同）。

4.3.4 从测试固件启动

用户可以编写特殊固件用于生产环境中测试，并在需要的时候运行。此时需要在项目分区表中专门申请一块分区用于保存该测试固件，其类型为 app，子类型为 test（详情请参考分区表）。

实现该测试应用固件需要为测试应用创建一个完全独立的 ESP-IDF 项目（ESP-IDF 中的每个项目仅构建一个应用）。该测试应用可以独立于主项目进行开发和测试，然后在生成测试时作为一个预编译.bin 文件集成到主项目的测试应用程序分区的地址。

要在主项目的引导加载程序中支持这个功能，请设置*CONFIG_BOOTLOADER_APP_TEST* 并配置以下三个选项：

- *CONFIG_BOOTLOADER_NUM_PIN_APP_TEST* - 设置启动 TEST 分区的管脚编号，该管脚将被配置为输入并启用内部上拉。要触发测试应用，必须在重置时将此管脚拉低或拉高（可配置）。
  释放管脚输入并重启设备后，将重新启用默认的启动顺序，即启动工厂分区或任意 OTA 应用分区槽。
- *CONFIG_BOOTLOADER_HOLD_TIME_GPIO* - 设置 GPIO 电平保持的时间，默认为 5 秒。设备重置后，管脚电平必须保持该设定的时间，才能执行恢复出厂设置或引导测试分区（如适用）。
- *CONFIG_BOOTLOADER_APP_TEST_PIN_LEVEL* - 配置应在 GPIO 的高电平还是低电平上触发测试分区启动。若 GPIO 有内部上拉，则该功能在采样管脚前就会被启用。关于管脚内部上拉的详细信息，请参考 ESP32-P4 数据规格书。

4.3.5 回滚

回滚和反回滚功能也必须在引导程序中配置。

请参考*OTA API 参考文档* 中的应用程序回滚 和防回滚 章节。

4.3.6 看门狗

芯片配备两组看门狗定时器：主系统看门狗定时器 (MWDT_WDT) 和 RTC 看门狗定时器 (RTC_WDT)。芯片上电时，两组看门狗定时器都会被启用，但在引导加载程序中，两组看门狗定时器都会被禁用。设置*CONFIG_BOOTLOADER_WDT_ENABLE*（默认设置）可以重新启用 RTC 看门狗定时器，用于跟踪从启用引导加载程序到调用用户主函数的时间。此期间内 RTC 看门狗定时器始终可用，并且如果在 9 秒内没有应用程序成功启动，则 RTC 看门狗定时器会自动重置芯片。这一功能可以有效防止启动过程中由于电源不稳定而导致的死机。

- 可以通过设置*CONFIG_BOOTLOADER_WDT_TIME_MS* 并重新编译引导加载程序来调整超时时间。
- 通过禁用*CONFIG_BOOTLOADER_WDT_ENABLE* 设置并重新编译引导加载程序，可以在引导加载程序中禁用 RTC 看门狗，但并不建议这样做。
- 请参阅硬件看门狗定时器，了解如何在应用程序中使用 RTC_WDT。

4.3.7 引导加载程序大小

当需要启用额外的引导加载程序功能，包括*flash 加密* 或安全启动，尤其是设置高级别*CONFIG_BOOTLOADER_LOG_LEVEL* 时，监控引导加载程序.bin 文件的大小变得非常重要。

当使用默认的*CONFIG_PARTITION_TABLE_OFFSET* 值 0x8000 时，二进制文件最大可为 0x8000 字节。

如果引导加载程序二进制文件过大，则引导加载程序会构建将失败并显示”Bootloader binary size [..] is too large for partition table offset” 的错误。如果此二进制文件已经被烧录，那么 ESP32-P4 将无法启动 - 日志中将记录无效分区表或无效引导加载程序校验和的错误。

可以使用如下方法解决此问题：

- 将*bootloader 编译器优化* 重新设置回默认值 “Size”。
- 降低*引导加载程序日志级别*。将日志级别设置为 Warning, Error 或 None 都会显著减少最终二进制文件的大小（但也可能会让调试变得更加困难）。
- 将*CONFIG_PARTITION_TABLE_OFFSET* 设置为高于 0x8000 的值，以便稍后将分区表放置在 flash 中，这样可以增加引导加载程序的可用空间。如果*分区表* 的 CSV 文件包含明确的分区偏移量，则需要修改这些偏移量，从而保证没有分区的偏移量低于 `CONFIG_PARTITION_TABLE_OFFSET + 0x1000`。（这包括随 ESP-IDF 提供的默认分区 CSV 文件）

当启用 Secure Boot V2 时，由于引导加载程序最先加载到固定大小的缓冲区中进行验证，对二进制文件大小的绝对限制为 64 KB (0x10000 bytes)（不包括 4 KB 签名）。

4.3.8 从深度睡眠中快速启动

引导加载程序有*CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP* 选项，可以减少从深度睡眠中唤醒的时间（有利于降低功耗）。当*CONFIG_SECURE_BOOT* 选项禁用时，该选项可用。由于无需镜像校验，唤醒时间减少。在第一次启动时，引导加载程序将启动的应用程序的地址存储在 RTC FAST 存储器中。而在唤醒过程中，这个地址用于启动而无需任何检查，从而实现了快速加载。

4.3.9 自定义引导加载程序

用户可以扩展或修改当前的引导加载程序，具体有两种方法：使用钩子实现或重写覆盖当前程序。这两种方法在 ESP-IDF 示例的 custom_bootloader 文件夹中都有呈现。

- *bootloader_hooks* 介绍了如何将钩子与引导加载程序初始化连接。
- *bootloader_override* 介绍了如何覆盖引导加载程序的实现。

在引导加载程序的代码中，用户不能使用其他组件提供的驱动和函数，如果确实需要，请将该功能的实现部分放在项目的 *bootloader_components* 目录中（注意，这会增加引导加载程序的大小）。

如果引导加载程序过大，则可能与内存中的分区表重叠，分区表默认烧录在偏移量 0x8000 处。增加*分区表偏移量* ，将分区表放在 flash 中靠后的区域，这样可以增加引导程序的可用空间。

4.4 构建系统

本文档主要介绍 ESP-IDF 构建系统的实现原理以及 组件等相关概念。如需了解如何组织和构建新的 ESP-IDF 项目或组件，请阅读本文档。

4.4.1 概述

一个 ESP-IDF 项目可以看作是多个不同组件的集合，例如一个显示当前湿度的网页服务器会包含以下组件：

- ESP-IDF 基础库，包括 libc、ROM bindings 等
- Wi-Fi 驱动
- TCP/IP 协议栈
- FreeRTOS 操作系统
- 网页服务器
- 湿度传感器的驱动
- 负责将上述组件整合到一起的主程序

ESP-IDF 可以显式地指定和配置每个组件。在构建项目的时候，构建系统会前往 ESP-IDF 目录、项目目录和用户自定义组件目录（可选）中查找所有组件，允许用户通过文本菜单系统配置 ESP-IDF 项目中用到的每个组件。在所有组件配置结束后，构建系统开始编译整个项目。

概念

- 项目特指一个目录，其中包含了构建可执行应用程序所需的全部文件和配置，以及其他支持型文件，例如分区表、数据/文件系统分区和引导程序。
- 项目配置保存在项目根目录下名为 `sdkconfig` 的文件中，可以通过 `idf.py menuconfig` 进行修改，且一个项目只能包含一个项目配置。
- 应用程序是由 ESP-IDF 构建得到的可执行文件。一个项目通常会构建两个应用程序：项目应用程序（可执行的主文件，即用户自定义的固件）和引导程序（启动并初始化项目应用程序）。
- 组件是模块化且独立的代码，会被编译成静态库（.a 文件）并链接到应用程序。部分组件由 ESP-IDF 官方提供，其他组件则来源于其它开源项目。
- 目标特指运行构建后应用程序的硬件设备。运行 *idf.py --list-targets* 可以查看当前 ESP-IDF 版本中支持目标的完整列表。

请注意，以下内容并不属于项目的组成部分：

- `ESP-IDF` 并不是项目的一部分，它独立于项目，通过 `IDF_PATH` 环境变量（保存 `esp-idf` 目录的路径）链接到项目，从而将 IDF 框架与项目分离。
- 交叉编译工具链并不是项目的组成部分，它应该被安装在系统 PATH 环境变量中。

4.4.2 使用构建系统

idf.py

`idf.py` 命令行工具提供了一个前端，可以帮助你轻松管理项目的构建过程，它管理了以下工具：

- CMake，配置待构建的项目
- Ninja，用于构建项目
- esptool.py，烧录目标硬件设备

可通过 `idf.py` 配置构建系统，具体可参考相关文档。

直接使用 CMake

为了方便，*idf.py* 已经封装了 CMake 命令，但是你愿意，也可以直接调用 CMake。

当 `idf.py` 在执行某些操作时，它会打印出其运行的每条命令以便参考。例如运行 `idf.py build` 命令与在 bash shell（或者 Windows Command Prompt）中运行以下命令是相同的：

```
mkdir -p build
cd build
cmake .. -G Ninja    # 或者 'Unix Makefiles'
ninja
```

在上面的命令列表中，`cmake` 命令对项目进行配置，并生成用于最终构建工具的构建文件。在这个例子中，最终构建工具是 Ninja: 运行 `ninja` 来构建项目。

没有必要多次运行 `cmake`。第一次构建后，往后每次只需运行 `ninja` 即可。如果项目需要重新配置，`ninja` 会自动重新调用 `cmake`。

若在 CMake 中使用 `ninja` 或 `make`，则多数 `idf.py` 子命令也会有其对应的目标，例如在构建目录下运行 `make menuconfig` 或 `ninja menuconfig` 与运行 `idf.py menuconfig` 是相同的。

备注： 如果你已经熟悉了 CMake，那么可能会发现 ESP-IDF 的 CMake 构建系统不同寻常，为了减少样板文件，该系统封装了 CMake 的许多功能。请参考*编写纯 CMake 组件* 以编写更多“CMake 风格”的组件。

使用 Ninja/Make 来烧录 可以直接使用 ninja 或 make 运行如下命令来构建项目并烧录:

```
ninja flash
```

或:

```
make app-flash
```

可用的目标还包括：`flash`、`app-flash` （仅用于 app）、`bootloader-flash` （仅用于 bootloader）。

以这种方式烧录时，可以通过设置 `ESPPORT` 和 `ESPBAUD` 环境变量来指定串口设备和波特率。可以在操作系统或 IDE 项目中设置该环境变量，或者直接在命令行中进行设置:

```
ESPPORT=/dev/ttyUSB0 ninja flash
```

备注： 在命令的开头为环境变量赋值属于 Bash shell 的语法，可在 Linux 、macOS 和 Windows 的类 Bash shell 中运行，但在 Windows Command Prompt 中无法运行。

或:

```
make -j3 app-flash ESPPORT=COM4 ESPBAUD=2000000
```

备注： 在命令末尾为变量赋值属于 `make` 的语法，适用于所有平台的 `make`。

在 IDE 中使用 CMake

还可以使用集成了 CMake 的 IDE，仅需将项目 `CMakeLists.txt` 文件的路径告诉 IDE 即可。集成 CMake 的 IDE 通常会有自己的构建工具（CMake 称之为 “生成器”），它是组成 IDE 的一部分，用来构建源文件。

向 IDE 中添加除 `build` 目标以外的自定义目标（如添加 “flash” 目标到 IDE）时，建议调用 `idf.py` 命令来执行这些 “特殊” 的操作。

有关将 ESP-IDF 同 CMake 集成到 IDE 中的详细信息，请参阅*构建系统的元数据*。

设置 Python 解释器

ESP-IDF 适用于 Python 3.8 以上版本。

`idf.py` 和其他的 Python 脚本会使用默认的 Python 解释器运行，如 `python`。你可以通过 `python3 $IDF_PATH/tools/idf.py ...` 命令切换到别的 Python 解释器，或者通过设置 shell 别名或其他脚本来简化该命令。

如果直接使用 CMake，运行 `cmake -D PYTHON=python3 ...`，CMake 会使用传入的值覆盖默认的 Python 解释器。

如果使用集成 CMake 的 IDE，可以在 IDE 的图形用户界面中给名为 `PYTHON` 的 CMake cache 变量设置新的值来覆盖默认的 Python 解释器。

如果想在命令行中更优雅地管理 Python 的各个版本，请查看 pyenv 或 virtualenv 工具，它们会帮助你更改默认的 python 版本。

4.4.3 示例项目

示例项目的目录树结构可能如下所示:

```
- myProject/
             - CMakeLists.txt
             - sdkconfig
             - bootloader_components/ - boot_component/ - CMakeLists.txt
                                                        - Kconfig
                                                        - src1.c
             - components/ - component1/ - CMakeLists.txt
                                         - Kconfig
                                         - src1.c
                           - component2/ - CMakeLists.txt
                                         - Kconfig
                                         - src1.c
                                         - include/ - component2.h
             - main/       - CMakeLists.txt
                           - src1.c
                           - src2.c

             - build/
```

该示例项目"myProject" 包含以下组成部分：

- 顶层项目 CMakeLists.txt 文件，这是 CMake 用于学习如何构建项目的主要文件，可以在这个文件中设置项目全局的 CMake 变量。顶层项目 CMakeLists.txt 文件会导入 /tools/cmake/project.cmake 文件，由它负责实现构建系统的其余部分。该文件最后会设置项目的名称，并定义该项目。
- "sdkconfig" 项目配置文件，执行 `idf.py menuconfig` 时会创建或更新此文件，文件中保存了项目中所有组件（包括 ESP-IDF 本身）的配置信息。`sdkconfig` 文件可能会也可能不会被添加到项目的源码管理系统中。
- 可选的"bootloader_components" 目录中包含了需要在引导加载项目中进行编译和链接的组件。并不是每个项目都需要这种自定义组件，但此类组件在引导加载程序需要修改以嵌入新功能时可能很有用。
- 可选的"components" 目录中包含了项目的部分自定义组件，并不是每个项目都需要这种自定义组件，但它有助于构建可复用的代码或者导入第三方（不属于 ESP-IDF）的组件。或者，你也可以在顶层 CMakeLists.txt 中设置 `EXTRA_COMPONENT_DIRS` 变量以查找其他指定位置处的组件。
- "main" 目录是一个特殊的组件，它包含项目本身的源代码。"main" 是默认名称，CMake 变量 `COMPONENT_DIRS` 默认包含此组件，但你可以修改此变量。有关详细信息，请参阅*重命名 main 组件*。如果项目中源文件较多，建议将其归于组件中，而不是全部放在"main" 中。
- "build" 目录是存放构建输出的地方，如果没有此目录，`idf.py` 会自动创建。CMake 会配置项目，并在此目录下生成临时的构建文件。随后，在主构建进程的运行期间，该目录还会保存临时目标文件、库文件以及最终输出的二进制文件。此目录通常不会添加到项目的源码管理系统中，也不会随项目源码一同发布。

每个组件目录都包含一个 `CMakeLists.txt` 文件，里面会定义一些变量以控制该组件的构建过程，以及其与整个项目的集成。更多详细信息请参阅*组件 CMakeLists 文件*。

每个组件还可以包含一个 `Kconfig` 文件，它用于定义 `menuconfig` 时展示的*组件配置* 选项。某些组件可能还会包含 `Kconfig.projbuild` 和 `project_include.cmake` 特殊文件，它们用于*覆盖项目的部分设置*。

4.4.4 项目 CMakeLists 文件

每个项目都有一个顶层 `CMakeLists.txt` 文件，包含整个项目的构建设置。默认情况下，项目 CMakeLists 文件会非常小。

最小 CMakeLists 文件示例

最小项目：

```
cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(myProject)
```

必要部分

每个项目都要按照上面显示的顺序添加上述三行代码：

- `cmake_minimum_required(VERSION 3.16)` 必须放在 CMakeLists.txt 文件的第一行，它会告诉 CMake 构建该项目所需要的最小版本号。ESP-IDF 支持 CMake 3.16 或更高的版本。
- `include($ENV{IDF_PATH}/tools/cmake/project.cmake)` 会导入 CMake 的其余功能来完成配置项目、检索组件等任务。
- `project(myProject)` 会创建项目本身，并指定项目名称。该名称会作为最终输出的二进制文件的名字，即 `myProject.elf` 和 `myProject.bin`。每个 CMakeLists 文件只能定义一个项目。

可选的项目变量

以下这些变量都有默认值，用户可以覆盖这些变量值以自定义构建行为。更多实现细节，请参阅 /tools/cmake/project.cmake 文件。

- `COMPONENT_DIRS`：组件的搜索目录，默认为 `IDF_PATH/components`、`PROJECT_DIR/components`、和 `EXTRA_COMPONENT_DIRS`。如果你不想在这些位置搜索组件，请覆盖此变量。
- `EXTRA_COMPONENT_DIRS`：用于搜索组件的其它可选目录列表。路径可以是相对于项目目录的相对路径，也可以是绝对路径。
- `COMPONENTS`：要构建进项目中的组件名称列表，默认为 `COMPONENT_DIRS` 目录下检索到的所有组件。使用此变量可以“精简”项目以缩短构建时间。请注意，如果一个组件通过 `COMPONENT_REQUIRES` 指定了它依赖的另一个组件，则会自动将其添加到 `COMPONENTS` 中，所以 `COMPONENTS` 列表可能会非常短。
- `BOOTLOADER_IGNORE_EXTRA_COMPONENT`：引导加载程序编译时应忽略的组件列表，位于 `bootloader_components/` 目录中。使用这一变量可以将一个组件有条件地包含在项目中。

以上变量中的路径可以是绝对路径，或者是相对于项目目录的相对路径。

请使用 cmake 中的 set 命令 来设置这些变量，如 `set(VARIABLE "VALUE")`。请注意，`set()` 命令需放在 `include(...)` 之前，`cmake_minimum(...)` 之后。

重命名 `main` 组件

构建系统会对 `main` 组件进行特殊处理。假如 `main` 组件位于预期的位置（即 *${PROJECT_PATH}/main*），那么它会被自动添加到构建系统中。其他组件也会作为其依赖项被添加到构建系统中，这使用户免于处理依赖关系，并提供即时可用的构建功能。重命名 `main` 组件会减轻上述这些幕后工作量，但要求用户指定重命名后的组件位置，并手动为其添加依赖项。重命名 `main` 组件的步骤如下：

1. 重命名 `main` 目录。
2. 在项目 CMakeLists.txt 文件中设置 `EXTRA_COMPONENT_DIRS`，并添加重命名后的 `main` 目录。
3. 在组件的 CMakeLists.txt 文件中设置 `COMPONENT_REQUIRES` 或 `COMPONENT_PRIV_REQUIRES` 以指定依赖项。

覆盖默认的构建规范

构建系统设置了一些全局的构建规范（编译标志、定义等），这些规范可用于编译来自所有组件的所有源文件。

例如，其中一个默认的构建规范是编译选项 `Wextra`。假设一个用户想用 `Wno-extra` 来覆盖这个选项，应在 `project()` 之后进行：

```
cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(myProject)

idf_build_set_property(COMPILE_OPTIONS "-Wno-error" APPEND)
```

这确保了用户设置的编译选项不会被默认的构建规范所覆盖，因为默认的构建规范是在 `project()` 内设置的。

4.4.5 组件 CMakeLists 文件

每个项目都包含一个或多个组件，这些组件可以是 ESP-IDF 的一部分，可以是项目自身组件目录的一部分，也可以从自定义组件目录添加（见上文）。

组件是 `COMPONENT_DIRS` 列表中包含 `CMakeLists.txt` 文件的任何目录。

搜索组件

搜索 `COMPONENT_DIRS` 中的目录列表以查找项目的组件，此列表中的目录可以是组件自身（即包含 *CMakeLists.txt* 文件的目录），也可以是子目录为组件的顶级目录。

当 CMake 运行项目配置时，它会记录本次构建包含的组件列表，它可用于调试某些组件的添加/排除。

同名组件

ESP-IDF 在搜索所有待构建的组件时，会按照 `COMPONENT_DIRS` 指定的顺序依次进行，这意味着在默认情况下，首先搜索 ESP-IDF 内部组件（`IDF_PATH/components`），然后是 `EXTRA_COMPONENT_DIRS` 中的组件，最后是项目组件（`PROJECT_DIR/components`）。如果这些目录中的两个或者多个包含具有相同名字的组件，则使用搜索到的最后一个位置的组件。这就允许将组件复制到项目目录中再修改以覆盖 ESP-IDF 组件，如果使用这种方式，ESP-IDF 目录本身可以保持不变。

备注： 如果在现有项目中通过将组件移动到一个新位置来覆盖它，项目不会自动看到新组件的路径。请运行 `idf.py reconfigure` 命令后（或删除项目构建文件夹）再重新构建。

最小组件 CMakeLists 文件

最小组件 `CMakeLists.txt` 文件通过使用 `idf_component_register` 将组件添加到构建系统中。

idf_component_register(SRCS "foo.c" "bar.c" INCLUDE_DIRS "include" REQUIRES mbedtls)

- `SRCS` 是源文件列表（`*.c`、`*.cpp`、`*.cc`、`*.S`），里面所有的源文件都将会编译进组件库中。
- `INCLUDE_DIRS` 是目录列表，里面的路径会被添加到所有需要该组件的组件（包括 main 组件）全局 include 搜索路径中。
- `REQUIRES` 实际上并不是必需的，但通常需要它来声明该组件需要使用哪些其它组件，请参考组件依赖。

上述命令会构建生成与组件同名的库，并最终被链接到应用程序中。

上述目录通常设置为相对于 `CMakeLists.txt` 文件的相对路径，当然也可以设置为绝对路径。

还有其它参数可以传递给 `idf_component_register`，具体可参考*here*。

有关更完整的 `CMakeLists.txt` 示例，请参阅组件依赖示例 和组件 *CMakeLists* 示例。

预设的组件变量

以下专用于组件的变量可以在组件 CMakeLists 中使用，但不建议修改：

- COMPONENT_DIR：组件目录，即包含 CMakeLists.txt 文件的绝对路径，它与 CMAKE_CURRENT_SOURCE_DIR 变量一样，路径中不能包含空格。
- COMPONENT_NAME：组件名，与组件目录名相同。
- COMPONENT_ALIAS：库别名，由构建系统在内部为组件创建。
- COMPONENT_LIB：库名，由构建系统在内部为组件创建。

以下变量在项目级别中被设置，但可在组件 CMakeLists 中使用：

- CONFIG_*：项目配置中的每个值在 cmake 中都对应一个以 CONFIG_ 开头的变量。更多详细信息请参阅*Kconfig*。
- ESP_PLATFORM：ESP-IDF 构建系统处理 CMake 文件时，其值设为 1。

构建/项目变量

以下是可作为构建属性的构建/项目变量，可通过组件 CMakeLists.txt 中的 idf_build_get_property 查询其变量值。

- PROJECT_NAME：项目名，在项目 CMakeLists.txt 文件中设置。
- PROJECT_DIR：项目目录（包含项目 CMakeLists 文件）的绝对路径，与 CMAKE_SOURCE_DIR 变量相同。
- COMPONENTS：此次构建中包含的所有组件的名称，具体格式为用分号隔开的 CMake 列表。
- IDF_VER：ESP-IDF 的 git 版本号，由 git describe 命令生成。
- IDF_VERSION_MAJOR、IDF_VERSION_MINOR、IDF_VERSION_PATCH: ESP-IDF 的组件版本，可用于条件表达式。请注意这些信息的精确度不如 IDF_VER 变量，版本号 v4.0-dev-*，v4.0-beta1，v4.0-rc1 和 v4.0 对应的 IDF_VERSION_* 变量值是相同的，但是 IDF_VER 的值是不同的。
- IDF_TARGET：项目的硬件目标名称。
- PROJECT_VER：项目版本号。
  - 如果设置*CONFIG_APP_PROJECT_VER_FROM_CONFIG* 选项，将会使用*CONFIG_APP_PROJECT_VER* 的值。
  - 或者，如果在项目 CMakeLists.txt 文件中设置了 PROJECT_VER 变量，则该变量值可以使用。
  - 或者，如果 PROJECT_DIR/version.txt 文件存在，其内容会用作 PROJECT_VER 的值。
  - 或者，如果在 CMakeLists.txt 文件中将 VERSION 参数传递给 project() 调用，形式为 project(... VERSION x.y.z.w)，那么 VERSION 参数将用作为 PROJECT_VER 的值。VERSION 参数必须符合 cmake 标准。
  - 或者，如果项目位于某个 Git 仓库中，则使用 git describe 命令的输出作为 PROJECT_VER 的值。
  - 否则，PROJECT_VER 的值为 1。
- EXTRA_PARTITION_SUBTYPES：CMake 列表，用于创建额外的分区子类型。子类型的描述由字符串组成，以逗号为分隔，格式为 type_name, subtype_name, numeric_value。组件可通过此列表，添加新的子类型。

其它与构建属性有关的信息请参考这里。

组件编译控制

在编译特定组件的源文件时，可以使用 target_compile_options 函数来传递编译器选项：

```
target_compile_options(${COMPONENT_LIB} PRIVATE -Wno-unused-variable)
```

如果给单个源文件指定编译器标志，可以使用 CMake 的 set_source_files_properties 命令：

```
set_source_files_properties(mysrc.c
    PROPERTIES COMPILE_FLAGS
```

(下页继续)

(续上页)

```
    -Wno-unused-variable
)
```

如果上游代码在编译的时候发出了警告，那这么做可能会很有效。

备注: 如已借助 idf_component_register 中的 SRC_DIRS 变量填充源文件，CMake set_source_files_properties 命令将无法使用，详情请参考文件通配 & 增量构建。

请注意，上述两条命令只能在组件 CMakeLists 文件的 idf_component_register 命令之后调用。

4.4.6 组件配置

每个组件都可以包含一个 Kconfig 文件，和 CMakeLists.txt 放在同一目录下。Kconfig 文件中包含要添加到该组件配置菜单中的一些配置设置信息。

运行 menuconfig 时，可以在 Component Settings 菜单栏下找到这些设置。

创建一个组件的 Kconfig 文件，最简单的方法就是使用 ESP-IDF 中现有的 Kconfig 文件作为模板，在这基础上进行修改。

有关示例请参阅添加条件配置。

4.4.7 预处理器定义

ESP-IDF 构建系统会在命令行中添加以下 C 预处理器定义：

- ESP_PLATFORM：可以用来检测在 ESP-IDF 内发生了构建行为。
- IDF_VER：定义 git 版本字符串，例如：v2.0 用于标记已发布的版本，v1.0-275-g0efaa4f 则用于标记任意某次的提交记录。

4.4.8 组件依赖

编译各个组件时，ESP-IDF 系统会递归评估其依赖项。这意味着每个组件都需要声明它所依赖的组件，即“requires”。

编写组件

```
idf_component_register(...
                       REQUIRES mbedtls
                       PRIV_REQUIRES console spiffs)
```

- REQUIRES 需要包含所有在当前组件的 公共头文件里 *#include* 的头文件所在的组件。
- PRIV_REQUIRES 需要包含被当前组件的源文件 *#include* 的头文件所在的组件（除非已经被设置在了 REQUIRES 中）。以及是当前组件正常工作必须要链接的组件。
- REQUIRES 和 PRIV_REQUIRES 的值不能依赖于任何配置选项（CONFIG_xxx 宏）。这是因为在配置加载之前，依赖关系就已经被展开。其它组件变量（比如包含路径或源文件）可以依赖配置选择。
- 如果当前组件除了通用组件依赖项 中设置的通用组件（比如 RTOS、libc 等）外，并不依赖其它组件，那么对于上述两个 REQUIRES 变量，可以选择其中一个或是两个都不设置。

如果组件仅支持某些硬件目标（IDF_TARGET 的值），则可以在 idf_component_register 中指定 REQUIRED_IDF_TARGETS 来声明这个需求。在这种情况下，如果构建系统导入了不支持当前硬件目标的组件时就会报错。

备注： 在 CMake 中，REQUIRES 和 PRIV_REQUIRES 是 CMake 函数 target_link_libraries(... PUBLIC ...) 和 target_link_libraries(... PRIVATE ...) 的近似包装。

组件依赖示例

假设现在有一个 car 组件，它需要使用 engine 组件，而 engine 组件需要使用 spark_plug 组件：

```
- autoProject/
             - CMakeLists.txt
             - components/ - car/ - CMakeLists.txt
                                  - car.c
                                  - car.h
                           - engine/ - CMakeLists.txt
                                     - engine.c
                                     - include/ - engine.h
                           - spark_plug/  - CMakeLists.txt
                                          - spark_plug.c
                                          - spark_plug.h
```

Car 组件 car.h 头文件是 car 组件的公共接口。该头文件直接包含了 engine.h，这是因为它需要使用 engine.h 中的一些声明：

```
/* car.h */
#include "engine.h"

#ifdef ENGINE_IS_HYBRID
#define CAR_MODEL "Hybrid"
#endif
```

同时 car.c 也包含了 car.h：

```
/* car.c */
#include "car.h"
```

这代表文件 car/CMakeLists.txt 需要声明 car 需要 engine：

```
idf_component_register(SRCS "car.c"
                    INCLUDE_DIRS "."
                    REQUIRES engine)
```

- SRCS 提供 car 组件中源文件列表。
- INCLUDE_DIRS 提供该组件公共头文件目录列表，由于 car.h 是公共接口，所以这里列出了所有包含了 car.h 的目录。
- REQUIRES 给出该组件的公共接口所需的组件列表。由于 car.h 是一个公共头文件并且包含了来自 engine 的头文件，所以我们这里包含 engine。这样可以确保任何包含 car.h 的其他组件也能递归地包含所需的 engine.h。

Engine 组件 engine 组件也有一个公共头文件 include/engine.h，但这个头文件更为简单：

```
/* engine.h */
#define ENGINE_IS_HYBRID

void engine_start(void);
```

在 engine.c 中执行：

```
/* engine.c */
#include "engine.h"
#include "spark_plug.h"

...
```

在该组件中，engine 依赖于 spark_plug，但这是私有依赖关系。编译 engine.c 需要 spark_plug.h 但不需要包含 engine.h。

这代表文件 engine/CMakeLists.txt 可以使用 PRIV_REQUIRES：

```
idf_component_register(SRCS "engine.c"
                    INCLUDE_DIRS "include"
                    PRIV_REQUIRES spark_plug)
```

因此，car 组件中的源文件不需要在编译器搜索路径中添加 spark_plug include 目录。这可以加快编译速度，避免编译器命令行过于的冗长。

Spark Plug 组件 spark_plug 组件没有依赖项，它有一个公共头文件 spark_plug.h，但不包含其他组件的头文件。

这代表 spark_plug/CMakeLists.txt 文件不需要任何 REQUIRES 或 PRIV_REQUIRES：

```
idf_component_register(SRCS "spark_plug.c"
                    INCLUDE_DIRS ".")
```

源文件 Include 目录

每个组件的源文件都是用这些 Include 路径目录编译的，这些路径在传递给 idf_component_register 的参数中指定：

```
idf_component_register(..
                        INCLUDE_DIRS "include"
                        PRIV_INCLUDE_DIRS "other")
```

- 当前组件的 INCLUDE_DIRS 和 PRIV_INCLUDE_DIRS。
- REQUIRES 和 PRIV_REQUIRES 参数指定的所有其他组件（即当前组件的所有公共和私有依赖项）所设置的 INCLUDE_DIRS。
- 递归列出所有组件 REQUIRES 列表中 INCLUDE_DIRS 目录（如递归展开这个组件的所有公共依赖项）。

主要组件依赖项

main 组件比较特别，因为它在构建过程中自动依赖所有其他组件。所以不需要向这个组件传递 REQUIRES 或 PRIV_REQUIRES。有关不再使用 main 组件时需要更改哪些内容，请参考*重命名 main 组件*。

通用组件依赖项

为避免重复性工作，各组件都用自动依赖一些“通用”IDF 组件，即使它们没有被明确提及。这些组件的头文件会一直包含在构建系统中。

通用组件包括：cxx、newlib、freertos、esp_hw_support、heap、log、soc、hal、esp_rom、esp_common、esp_system。

在构建中导入组件

- 默认情况下，每个组件都会包含在构建系统中。
- 如果将 COMPONENTS 变量设置为项目直接使用的最小组件列表，那么构建系统会扩展到包含所有组件。完整的组件列表为：
  - COMPONENTS 中明确提及的组件。
  - 这些组件的依赖项（以及递归运算后的组件）。
  - 每个组件都依赖的通用组件。
- 将 COMPONENTS 设置为所需组件的最小列表，可以显著减少项目的构建时间。

循环依赖

一个项目中可能包含组件 A 和组件 B，而组件 A 依赖（REQUIRES 或 PRIV_REQUIRES）组件 B，组件 B 又依赖组件 A。这就是所谓的依赖循环或循环依赖。

CMake 通常会在链接器命令行上重复两次组件库名称来自动处理循环依赖。然而这种方法并不总是有效，还是可能构建失败并出现关于“Undefined reference to ...”的链接器错误，这通常是由于引用了循环依赖中某一组件中定义的符号。如果存在较大的循环依赖关系，即 A->B->C->D->A，这种情况极有可能发生。

最好的解决办法是重构组件以消除循环依赖关系。在大多数情况下，没有循环依赖的软件架构具有模块化和分层清晰的特性，并且从长远来看更容易维护。然而，移除循环依赖关系并不容易做到。

要绕过由循环依赖引起的链接器错误，最简单的解决方法是增加其中一个组件库的 CMake LINK_INTERFACE_MULTIPLICITY 属性。这会让 CMake 在链接器命令行上对此库及其依赖项重复两次以上。

例如：

```
set_property(TARGET ${COMPONENT_LIB} APPEND PROPERTY LINK_INTERFACE_MULTIPLICITY 3)
```

- 这一行应该放在组件 CMakeLists.txt 文件 idf_component_register 之后。
- 可以的话，将此行放置在因依赖其他组件而造成循环依赖的组件中。实际上，该行可以放在循环内的任何一个组件中，但建议将其放置在拥有链接器错误提示信息中显示的源文件的组件中，或是放置在定义了链接器错误提示信息中所提到的符号的组件，先从这些组件开始是个不错的选择。
- 通常将值增加到 3（默认值是 2）就足够了，但如果不起作用，可以尝试逐步增加这个数字。
- 注意，增加这个选项会使链接器的命令行变长，链接阶段变慢。

高级解决方法：未定义符号 如果只有一两个符号导致循环依赖，而所有其他依赖都是线性的，那么有一种替代方法可以避免链接器错误：在链接时将“反向”依赖所需的特定符号指定为未定义符号。

例如，如果组件 A 依赖于组件 B，但组件 B 也需要引用组件 A 的 reverse_ops（但不依赖组件 A 中的其他内容），那么你可以在组件 B 的 CMakeLists.txt 中添加如下一行，以在链接时避免这出现循环。

```
# 该符号是由“组件 A”在链接时提供
target_link_libraries(${COMPONENT_LIB} INTERFACE "-u reverse_ops")
```

- -u 参数意味着链接器将始终在链接中包含此符号，而不管依赖项顺序如何。
- 该行应该放在组件 CMakeLists.txt 文件中的 idf_component_register 之后。
- 如果“组件 B”不需要访问“组件 A”的任何头文件，只需链接几个符号，那么这一行可以用来代替 B 对 A 的任何“REQUIRES”。这样则进一步简化了构建系统中的组件结构。

请参考 target_link_libraries 文档以了解更多关于此 CMake 函数的信息。

构建系统中依赖处理的实现细节

- 在 CMake 配置进程的早期阶段会运行 expand_requirements.cmake 脚本。该脚本会对所有组件的 CMakeLists.txt 文件进行局部的运算，得到一张组件依赖关系图（此图可能会有闭环）。此图用于在构建目录中生成 component_depends.cmake 文件。

- CMake 主进程会导入该文件，并以此来确定要包含到构建系统中的组件列表（内部使用的 BUILD_COMPONENTS 变量）。BUILD_COMPONENTS 变量已排好序，依赖组件会排在前面。由于组件依赖关系图中可能存在闭环，因此不能保证每个组件都满足该排序规则。如果给定相同的组件集和依赖关系，那么最终的排序结果应该是确定的。
- CMake 会将 BUILD_COMPONENTS 的值以 "Component names:" 的形式打印出来。
- 然后执行构建系统中包含的每个组件的配置。
- 每个组件都被正常包含在构建系统中，然后再次执行 CMakeLists.txt 文件，将组件库加入构建系统。

组件依赖顺序 BUILD_COMPONENTS 变量中组件的顺序决定了构建过程中的其它顺序，包括：

- 项目导入*project_include.cmake* 文件的顺序。
- 生成用于编译（通过 -I 参数）的头文件路径列表的顺序。请注意，对于给定组件的源文件，仅需将该组件的依赖组件的头文件路径告知编译器。

添加链接时依赖项 ESP-IDF 的 CMake 辅助函数 idf_component_add_link_dependency 可以在组件之间添加仅作用于链接时的依赖关系。绝大多数情况下，我们都建议你使用 idf_component_register 中的 PRIV_REQUIRES 功能来构建依赖关系。然而在某些情况下，还是有必要添加另一个组件对当前组件的链接时依赖，即反转 PRIV_REQUIRES 中的依赖关系（参考示例：*Overriding Default Chip Drivers*）。

要使另一个组件在链接时依赖于这个组件：

```
idf_component_add_link_dependency(FROM other_component)
```

请将上述行置于 idf_component_register 行之后。

也可以通过名称指定两个组件：

```
idf_component_add_link_dependency(FROM other_component TO that_component)
```

4.4.9 覆盖项目的部分设置

project_include.cmake

如果组件的某些构建行为需要在组件 CMakeLists 文件之前被执行，你可以在组件目录下创建名为 project_include.cmake 的文件，project.cmake 在运行过程中会导入此 CMake 文件。

project_include.cmake 文件在 ESP-IDF 内部使用，以定义项目范围内的构建功能，比如 esptool.py 的命令行参数和 bootloader 这个特殊的应用程序。

与组件 CMakeLists.txt 文件有所不同，在导入 project_include.cmake 文件的时候，当前源文件目录（即 CMAKE_CURRENT_SOURCE_DIR 和工作目录）为项目目录。如果想获得当前组件的绝对路径，可以使用 COMPONENT_PATH 变量。

请注意，project_include.cmake 对于大多数常见的组件并不是必需的。例如给项目添加 include 搜索目录，给最终的链接步骤添加 LDFLAGS 选项等等都可以通过 CMakeLists.txt 文件来自定义。详细信息请参考可选的项目变量。

project_include.cmake 文件会按照 BUILD_COMPONENTS 变量中组件的顺序（由 CMake 记录）依次导入。即只有在当前组件所有依赖组件的 project_include.cmake 文件都被导入后，当前组件的 project_include.cmake 文件才会被导入，除非两个组件在同一个依赖闭环中。如果某个 project_include.cmake 文件依赖于另一组件设置的变量，则要特别注意上述情况。更多详情请参阅构建系统中依赖处理的实现细节。

在 project_include.cmake 文件中设置变量或目标时要格外小心，这些值被包含在项目的顶层 CMake 文件中，因此他们会影响或破坏所有组件的功能。

KConfig.projbuild

与 project_include.cmake 类似，也可以为组件定义一个 KConfig 文件以实现全局的组件配置。如果要在 menuconfig 的顶层添加配置选项，而不是在 “Component Configuration” 子菜单中，则可以在 CMakeLists.txt 文件所在目录的 KConfig.projbuild 文件中定义这些选项。

在此文件中添加配置时要小心，因为这些配置会包含在整个项目配置中。在可能的情况下，请为组件配置 创建 KConfig 文件。

通过封装对现有函数进行重新定义或扩展

链接器具有封装功能，可以重新定义或扩展现有 ESP-IDF 函数的行为。如需封装函数，你需要在项目的 CMakeLists.txt 文件中提供以下 CMake 声明：

```
target_link_libraries(${COMPONENT_LIB} INTERFACE "-Wl,--wrap=function_to_redefine")
```

其中，function_to_redefine 为需要被重新定义或扩展的函数名称。启用此选项后，链接器将把二进制库中所有对 function_to_redefine 函数的调用改为对 __wrap_function_to_redefine 函数的调用。因此，你必须在应用程序中定义这一符号。

链接器会提供一个名为 __real_function_to_redefine 的新符号，指向将被重新定义的函数的原有实现。由此，可以从新的实现中调用该函数，从而对原有实现进行扩展。

请参考 build_system/wrappers 示例，了解其详细原理。更多细节请参阅 examples/build_system/wrappers/README.md。

覆盖默认引导加载程序

由于 ESP-IDF 中存在可选目录 bootloader_components，因此可以覆盖默认的 ESP-IDF 引导加载程序。覆盖前，应定义一个 bootloader_components/main 组件，使项目目录如下所示：

- **myProject/**
  - CMakeLists.txt
  - sdkconfig
  - **bootloader_components/ - main/ - CMakeLists.txt**
    - Kconfig
    - my_bootloader.c
  - **main/ - CMakeLists.txt**
    - app_main.c
  - build/

此处的 my_bootloader.c 文件会成为新引导加载程序的源代码，这意味着它需要执行所有必要的操作来设置并从 flash 中加载 main 应用程序。

还可以根据特定的条件来替换引导加载程序，例如替换指定目标芯片的引导加载程序。这可以通过 BOOTLOADER_IGNORE_EXTRA_COMPONENT CMake 变量实现，该列表会让 ESP-IDF 引导加载项目忽略 bootloader_components 中的指定组件，不对其进行编译。例如，如果希望使用 ESP32 目标芯片的默认引导加载程序，myProject/CMakeLists.txt 应如下所示：

```
include($ENV{IDF_PATH}/tools/cmake/project.cmake)

if(${IDF_TARGET} STREQUAL "esp32")
    set(BOOTLOADER_IGNORE_EXTRA_COMPONENT "main")
endif()

project(main)
```

值得注意的是，这还可以用于除 main 之外的其他引导加载程序组件。在任何情况下，都不能指定前缀 bootloader_component。

请参考 custom_bootloader/bootloader_override 查看覆盖默认引导加载程序的示例。

4.4.10 仅配置组件

仅配置组件是一类不包含源文件的特殊组件，仅包含 `Kconfig.projbuild`、`KConfig` 和 `CMakeLists.txt` 文件，该 `CMakeLists.txt` 文件仅有一行代码，调用了 `idf_component_register()` 函数。此函数会将组件导入到项目构建中，但不会构建任何库，也不会将头文件添加到任何 include 搜索路径中。

4.4.11 CMake 调试

请查看 CMake v3.16 官方文档 获取更多关于 CMake 和 CMake 命令的信息。

调试 ESP-IDF CMake 构建系统的一些技巧：

- CMake 运行时，会打印大量诊断信息，包括组件列表和组件路径。
- 运行 `cmake -DDEBUG=1`，IDF 构建系统会生成更详细的诊断输出。
- 运行 `cmake` 时指定 `--trace` 或 `--trace-expand` 选项会提供大量有关控制流信息。详情请参考 CMake 命令行文档。

当从项目 CMakeLists 文件导入时，`project.cmake` 文件会定义工具模块和全局变量，并在系统环境中没有设置 `IDF_PATH` 时设置 `IDF_PATH`。

同时还定义了一个自定义版本的内置 CMake `project` 函数，这个函数被覆盖，以添加所有 ESP-IDF 特定的项目功能。

警告未定义的变量

默认情况下，警告未定义的变量这一功能是关闭的。

可通过将 `--warn-uninitialized` 标志传递给 CMake 或通过将 `--cmake-warn-uninitialized` 传递给 `idf.py` 来使能这一功能。这样，如果在构建的过程中引用了未定义的变量，CMake 会打印警告。这对查找有错误的 CMake 文件非常有用。

更多信息，请参考文件 /tools/cmake/project.cmake 以及 /tools/cmake/ 中支持的函数。

4.4.12 组件 CMakeLists 示例

因为构建环境试图设置大多数情况都能工作的合理默认值，所以组件 `CMakeLists.txt` 文件可能非常小，甚至是空的，请参考最小组件 *CMakeLists* 文件。但有些功能往往需要覆盖预设的组件变量 才能实现。

以下是组件 CMakeLists 文件的更高级的示例。

添加条件配置

配置系统可用于根据项目配置中选择的选项有条件地编译某些文件。

`Kconfig`:

```
config FOO_ENABLE_BAR
    bool "Enable the BAR feature."
    help
        This enables the BAR feature of the FOO component.
```

`CMakeLists.txt`:

```
set(srcs "foo.c" "more_foo.c")

if(CONFIG_FOO_ENABLE_BAR)
    list(APPEND srcs "bar.c")
endif()
```

(下页继续)

(续上页)

```
idf_component_register(SRCS "${srcs}"
                       ...)
```

上述示例使用了 CMake 的 if 函数和 list APPEND 函数。

也可用于选择或删除某一实现，如下所示：

Kconfig:

```
config ENABLE_LCD_OUTPUT
    bool "Enable LCD output."
    help
        Select this if your board has a LCD.

config ENABLE_LCD_CONSOLE
    bool "Output console text to LCD"
    depends on ENABLE_LCD_OUTPUT
    help
        Select this to output debugging output to the lcd

config ENABLE_LCD_PLOT
    bool "Output temperature plots to LCD"
    depends on ENABLE_LCD_OUTPUT
    help
        Select this to output temperature plots
```

CMakeLists.txt:

```
if(CONFIG_ENABLE_LCD_OUTPUT)
   set(srcs lcd-real.c lcd-spi.c)
else()
   set(srcs lcd-dummy.c)
endif()

# 如果启用了控制台或绘图功能，则需要加入字体
if(CONFIG_ENABLE_LCD_CONSOLE OR CONFIG_ENABLE_LCD_PLOT)
   list(APPEND srcs "font.c")
endif()

idf_component_register(SRCS "${srcs}"
                       ...)
```

硬件目标的条件判断

CMake 文件可以使用 IDF_TARGET 变量来获取当前的硬件目标。

此外，如果当前的硬件目标是 xyz （即 IDF_TARGET=xyz），那么 Kconfig 变量 CONFIG_IDF_TARGET_XYZ 同样也会被设置。

请注意，组件可以依赖 IDF_TARGET 变量，但不能依赖这个 Kconfig 变量。同样也不可在 CMake 文件的 include 语句中使用 Kconfig 变量，在这种上下文中可以使用 IDF_TARGET。

生成源代码

有些组件的源文件可能并不是由组件本身提供，而必须从另外的文件生成。假设组件需要一个头文件，该文件由 BMP 文件转换后（使用 bmp2h 工具）的二进制数据组成，然后将头文件包含在名为 graphics_lib.c 的文件中：

```
add_custom_command(OUTPUT logo.h
     COMMAND bmp2h -i ${COMPONENT_DIR}/logo.bmp -o log.h
     DEPENDS ${COMPONENT_DIR}/logo.bmp
     VERBATIM)

add_custom_target(logo DEPENDS logo.h)
add_dependencies(${COMPONENT_LIB} logo)

set_property(DIRECTORY "${COMPONENT_DIR}" APPEND PROPERTY
     ADDITIONAL_MAKE_CLEAN_FILES logo.h)
```

这个示例改编自 CMake 的一则 FAQ，其中还包含了一些同样适用于 ESP-IDF 构建系统的示例。

这个示例会在当前目录（构建目录）中生成 logo.h 文件，而 logo.bmp 会随组件一起提供在组件目录中。因为 logo.h 是一个新生成的文件，一旦项目需要清理，该文件也应该要被清除。因此，要将该文件添加到 ADDITIONAL_MAKE_CLEAN_FILES 属性中。

备注： 如果需要生成文件作为项目 CMakeLists.txt 的一部分，而不是作为组件 CMakeLists.txt 的一部分，此时需要使用 `${PROJECT_PATH}` 替代 `${COMPONENT_DIR}`，使用 `${PROJECT_NAME}.elf` 替代 `${COMPONENT_LIB}`。

如果某个源文件是从其他组件中生成，且包含 `logo.h` 文件，则需要调用 `add_dependencies`，在这两个组件之间添加一个依赖项，以确保组件源文件按照正确顺序进行编译。

嵌入二进制数据

有时你的组件希望使用一个二进制文件或者文本文件，但是你又不希望将它们重新格式化为 C 源文件。

这时，你可以在组件注册中指定 `EMBED_FILES` 参数，用空格分隔要嵌入的文件名称:

```
idf_component_register(...
                       EMBED_FILES server_root_cert.der)
```

或者，如果文件是字符串，则可以使用 `EMBED_TXTFILES` 变量，把文件的内容转成以 null 结尾的字符串嵌入:

```
idf_component_register(...
                       EMBED_TXTFILES server_root_cert.pem)
```

文件的内容会被添加到 flash 的.rodata 段，用户可以通过符号名来访问，如下所示:

```
extern const uint8_t server_root_cert_pem_start[] asm("_binary_server_root_cert_
↪pem_start");
extern const uint8_t server_root_cert_pem_end[]   asm("_binary_server_root_cert_
↪pem_end");
```

符号名会根据文件全名生成，如 `EMBED_FILES` 中所示，字符 `/`、`.` 等都会被下划线替代。符号名称中的 _binary 前缀由 objcopy 命令添加，对文本文件和二进制文件都是如此。

如果要将文件嵌入到项目中，而非组件中，可以调用 `target_add_binary_data` 函数:

```
target_add_binary_data(myproject.elf "main/data.bin" TEXT)
```

并将这行代码放在项目 CMakeLists.txt 的 `project()` 命令之后，修改 `myproject.elf` 为你自己的项目名。如果最后一个参数是 `TEXT`，那么构建系统会嵌入以 null 结尾的字符串，如果最后一个参数被设置为 `BINARY`，则将文件内容按照原样嵌入。

有关使用此技术的示例，请查看 file_serving 示例 protocols/http_server/file_serving/main/CMakeLists.txt 中的 main 组件，两个文件会在编译时加载并链接到固件中。

也可以嵌入生成的文件:

```
add_custom_command(OUTPUT my_processed_file.bin
                   COMMAND my_process_file_cmd my_unprocessed_file.bin)
target_add_binary_data(my_target "my_processed_file.bin" BINARY)
```

上述示例中，my_processed_file.bin 是通过命令 my_process_file_cmd 从文件 my_unprocessed_file.bin 中生成，然后嵌入到目标中。

使用 DEPENDS 参数来指明对目标的依赖性:

```
add_custom_target(my_process COMMAND ...)
target_add_binary_data(my_target "my_embed_file.bin" BINARY DEPENDS my_process)
```

target_add_binary_data 的 DEPENDS 参数确保目标首先执行。

代码和数据的存放

ESP-IDF 还支持自动生成链接脚本，它允许组件通过链接片段文件定义其代码和数据在内存中的存放位置。构建系统会处理这些链接片段文件，并将处理后的结果扩充进链接脚本，从而指导应用程序二进制文件的链接过程。更多详细信息与快速上手指南，请参阅链接脚本生成机制。

完全覆盖组件的构建过程

当然，在有些情况下，上面提到的方法不一定够用。如果组件封装了另一个第三方组件，而这个第三方组件并不能直接在 ESP-IDF 的构建系统中工作，在这种情况下，就需要放弃 ESP-IDF 的构建系统，改为使用 CMake 的 ExternalProject 功能。组件 CMakeLists 示例如下:

```
# 用于 quirc 的外部构建过程，在源目录中运行
# 并生成 libquirc.a
externalproject_add(quirc_build
    PREFIX ${COMPONENT_DIR}
    SOURCE_DIR ${COMPONENT_DIR}/quirc
    CONFIGURE_COMMAND ""
    BUILD_IN_SOURCE 1
    BUILD_COMMAND make CC=${CMAKE_C_COMPILER} libquirc.a
    INSTALL_COMMAND ""
    )

# 将 libquirc.a 添加到构建系统中
add_library(quirc STATIC IMPORTED GLOBAL)
add_dependencies(quirc quirc_build)

set_target_properties(quirc PROPERTIES IMPORTED_LOCATION
    ${COMPONENT_DIR}/quirc/libquirc.a)
set_target_properties(quirc PROPERTIES INTERFACE_INCLUDE_DIRECTORIES
    ${COMPONENT_DIR}/quirc/lib)

set_directory_properties( PROPERTIES ADDITIONAL_MAKE_CLEAN_FILES
    "${COMPONENT_DIR}/quirc/libquirc.a")
```

（上述 CMakeLists.txt 可用于创建名为 quirc 的组件，该组件使用自己的 Makefile 构建 quirc 项目。）

- externalproject_add 定义了一个外部构建系统。
  - 设置 SOURCE_DIR、CONFIGURE_COMMAND、BUILD_COMMAND 和 INSTALL_COMMAND。如果外部构建系统没有配置这一步骤，可以将 CONFIGURE_COMMAND 设置为空字符串。在 ESP-IDF 的构建系统中，一般会将 INSTALL_COMMAND 变量设置为空。
  - 设置 BUILD_IN_SOURCE，即构建目录与源目录相同。否则，你也可以设置 BUILD_DIR 变量。
  - 有关 externalproject_add() 命令的详细信息，请参阅 ExternalProject。
- 第二组命令添加了一个目标库，指向外部构建系统生成的库文件。为了添加 include 目录，并告知 CMake 该文件的位置，需要再设置一些属性。

- 最后，生成的库被添加到 ADDITIONAL_MAKE_CLEAN_FILES 中。即执行 `make clean` 后会删除该库。请注意，构建系统中的其他目标文件不会被删除。

ExternalProject 的依赖与构建清理 对于外部项目的构建，CMake 会有一些不同寻常的行为：

- ADDITIONAL_MAKE_CLEAN_FILES 仅在使用 Make 或 Ninja 构建系统时有效。如果使用 IDE 自带的构建系统，执行项目清理时，这些文件不会被删除。
- ExternalProject 会在 clean 运行后自动重新运行配置和构建命令。
- 可以采用以下两种方法来配置外部构建命令：
  1. 将外部 `BUILD_COMMAND` 命令设置为对所有源代码完整的重新编译。如果传递给 `externalproject_add` 命令的 `DEPENDS` 的依赖项发生了改变，或者当前执行的是项目清理操作（即运行了 `idf.py clean`、`ninja clean` 或者 `make clean`），那么就会执行该命令。
  2. 将外部 `BUILD_COMMAND` 命令设置为增量式构建命令，并给 `externalproject_add` 传递 `BUILD_ALWAYS 1` 参数。即不管实际的依赖情况，每次构建时，都会构建外部项目。这种方式仅当外部构建系统具备增量式构建的能力，且运行时间不会很长时才推荐。

构建外部项目的最佳方法取决于项目本身、其构建系统，以及是否需要频繁重新编译项目。

4.4.13 自定义 sdkconfig 的默认值

对于示例工程或者其他你不想指定完整 sdkconfig 配置的项目，但是你确实希望覆盖 ESP-IDF 默认值中的某些键值，则可以在项目中创建 `sdkconfig.defaults` 文件。重新创建新配置时将会用到此文件，另外在 `sdkconfig` 没有设置新配置值时，上述文件也会被用到。

如若需要覆盖此文件的名称或指定多个文件，请设置 `SDKCONFIG_DEFAULTS` 环境变量或在顶层 CMakeLists.txt 文件中设置 `SDKCONFIG_DEFAULTS`。非绝对路径的文件名将以当前项目的相对路径来解析。

在指定多个文件时，使用分号作为分隔符。先列出的文件将会先应用。如果某个键值在多个文件里定义，后面文件的定义会覆盖前面文件的定义。

一些 IDF 示例中包含了 `sdkconfig.ci` 文件。该文件是 CI（持续集成）测试框架的一部分，在正常构建过程中会被忽略。

依赖于硬件目标的 sdkconfig 默认值

当且仅当 `sdkconfig.defaults` 文件存在时，构建系统还将尝试从 `sdkconfig.defaults.TARGET_NAME` 文件中加载默认值，其中 `IDF_TARGET` 的值为 `TARGET_NAME`。例如，对于 `esp32` 这个目标芯片，sdkconfig 的默认值会首先从 `sdkconfig.defaults` 获取，然后再从 `sdkconfig.defaults.esp32` 获取。当没有通用的默认设置时，仍需创建一个空的 `sdkconfig.defaults` 文件，以便构建系统可以识别任何其他与目标芯片相关的 `sdkconfig.defaults.TARGET_NAME` 文件。

如果使用 `SDKCONFIG_DEFAULTS` 覆盖默认文件的名称，则硬件目标的默认文件名也会从 `SDKCONFIG_DEFAULTS` 值中派生。如果 `SDKCONFIG_DEFAULTS` 中有多个文件，硬件目标文件会在引入该硬件目标文件的文件之后应用，而 `SDKCONFIG_DEFAULTS` 中所有其它后续文件则会在硬件目标文件之后应用。

例如，如果 `SDKCONFIG_DEFAULTS="sdkconfig.defaults;sdkconfig_devkit1"`，并且在同一文件夹中有一个 `sdkconfig.defaults.esp32` 文件，那么这些文件将按以下顺序应用：(1) sdkconfig.defaults (2) sdkconfig.defaults.esp32 (3) sdkconfig_devkit1。

4.4.14 flash 参数

有些情况下，我们希望在没有 IDF 时也能烧写目标板，为此，我们希望可以保存已构建的二进制文件、esptool.py 和 esptool write_flash 命令的参数。可以通过编写一段简单的脚本来保存二进制文件和 esptool.py。

运行项目构建之后，构建目录将包含项目二进制输出文件（`.bin` 文件），同时也包含以下烧录数据文件：

- `flash_project_args` 包含烧录整个项目的参数，包括应用程序 (app)、引导程序 (bootloader)、分区表，如果设置了 PHY 数据，也会包含此数据。
- `flash_app_args` 只包含烧录应用程序的参数。
- `flash_bootloader_args` 只包含烧录引导程序的参数。

你可以参照如下命令将任意烧录参数文件传递给 `esptool.py`:

```
python esptool.py --chip esp32p4 write_flash @build/flash_project_args
```

也可以手动复制参数文件中的数据到命令行中执行。

构建目录中还包含生成的 `flasher_args.json` 文件，此文件包含 JSON 格式的项目烧录信息，可用于 `idf.py` 和其它需要项目构建信息的工具。

4.4.15 构建 Bootloader

引导程序是 /components/bootloader/subproject 内部独特的“子项目”，它有自己的项目 CMakeLists.txt 文件，能够构建独立于主项目的 `.ELF` 和 `.BIN` 文件，同时它又与主项目共享配置和构建目录。

子项目通过 /components/bootloader/project_include.cmake 文件作为外部项目插入到项目的顶层，主构建进程会运行子项目的 CMake，包括查找组件（主项目使用的组件的子集），生成引导程序专用的配置文件（从主 `sdkconfig` 文件中派生）。

4.4.16 编写纯 CMake 组件

ESP-IDF 构建系统用“组件”的概念“封装”了 CMake，并提供了很多帮助函数来自动将这些组件集成到项目构建当中。

然而，“组件”概念的背后是一个完整的 CMake 构建系统，因此可以制作纯 CMake 组件。

下面是使用纯 CMake 语法为 `json` 组件编写的最小 CMakeLists 文件的示例:

```
add_library(json STATIC
cJSON/cJSON.c
cJSON/cJSON_Utils.c)

target_include_directories(json PUBLIC cJSON)
```

- 这实际上与 IDF 中的 json 组件 是等效的。
- 因为组件中的源文件不多，所以这个 CMakeLists 文件非常简单。对于具有大量源文件的组件而言，ESP-IDF 支持的组件通配符，可以简化组件 CMakeLists 的样式。
- 每当组件中新增一个与组件同名的库目标时，ESP-IDF 构建系统会自动将其添加到构建中，并公开公共的 include 目录。如果组件想要添加一个与组件不同名的库目标，就需要使用 CMake 命令手动添加依赖关系。

4.4.17 组件中使用第三方 CMake 项目

CMake 在许多开源的 C/C++ 项目中广泛使用，用户可以在自己的应用程序中使用开源代码。CMake 构建系统的一大好处就是可以导入这些第三方的项目，有时候甚至不用做任何改动。这就允许用户使用当前 ESP-IDF 组件尚未提供的功能，或者使用其它库来实现相同的功能。

假设 `main` 组件需要导入一个假想库 `foo`，相应的组件 CMakeLists 文件如下所示:

```
# 注册组件
idf_component_register(...)

# 设置 `foo` 项目中的一些 CMake 变量，以控制 `foo` 的构建过程
set(FOO_BUILD_STATIC OFF)
set(FOO_BUILD_TESTS OFF)
```

(下页继续)

(续上页)

```
# 创建并导入第三方库目标
add_subdirectory(foo)

# 将 `foo` 目标公开链接至 `main` 组件
target_link_libraries(main PUBLIC foo)
```

实际的案例请参考 build_system/cmake/import_lib。请注意，导入第三方库所需要做的工作可能会因库的不同而有所差异。建议仔细阅读第三方库的文档，了解如何将其导入到其它项目中。阅读第三方库的 CMakeLists.txt 文件以及构建结构也会有所帮助。

用这种方式还可以将第三方库封装成 ESP-IDF 的组件。例如 mbedtls 组件就是封装了 mbedtls 项目 得到的。详情请参考 mbedtls 组件的 CMakeLists.txt 文件 。

每当使用 ESP-IDF 构建系统时，CMake 变量 `ESP_PLATFORM` 都会被设置为 1。如果要在通用的 CMake 代码加入 IDF 特定的代码时，可以采用 `if (ESP_PLATFORM)` 的形式加以分隔。

外部库中使用 ESP-IDF 组件

上述示例中假设的是外部库 `foo` （或 `import_lib` 示例中的 `tinyxml` 库）除了常见的 API 如 libc、libstdc++ 等外不需要使用其它 ESP-IDF API。如果外部库需要使用其它 ESP-IDF 组件提供的 API，则需要在外部 CMakeLists.txt 文件中通过添加对库目标 `idf::<componentname>` 的依赖关系。

例如，在 `foo/CMakeLists.txt` 文件:

```
add_library(foo bar.c fizz.cpp buzz.cpp)

if(ESP_PLATFORM)
  # 在 ESP-IDF 中、bar.c 需要包含 spi_flash 组件中的 esp_flash.h
  target_link_libraries(foo PRIVATE idf::spi_flash)
endif()
```

4.4.18 组件中使用预建库

还有一种情况是你有一个由其它构建过程生成预建静态库（`.a` 文件）。

ESP-IDF 构建系统为用户提供了一个实用函数 `add_prebuilt_library`，能够轻松导入并使用预建库:

```
add_prebuilt_library(target_name lib_path [REQUIRES req1 req2 ...] [PRIV_REQUIRES␣
↪req1 req2 ...])
```

其中:

- `target_name`- 用于引用导入库的名称，如链接到其它目标时
- `lib_path`- 预建库的路径，可以是绝对路径或是相对于组件目录的相对路径

可选参数 `REQUIRES` 和 `PRIV_REQUIRES` 指定对其它组件的依赖性。这些参数与 `idf_component_register` 的参数的意义相同。

注意预建库的编译目标需与目前的项目相同。预建库的相关参数也要匹配。如果不特别注意，这两个因素可能会导致应用程序中出现 bug。

请查看示例 build_system/cmake/import_prebuilt。

4.4.19 在自定义 CMake 项目中使用 ESP-IDF

ESP-IDF 提供了一个模板 CMake 项目，可以基于此轻松创建应用程序。然而在有些情况下，用户可能已有一个现成的 CMake 项目，或者想自己创建一个 CMake 项目，此时就希望将 IDF 中的组件以库的形式链接到用户目标（库/可执行文件）。

可以通过 tools/cmake/idf.cmake 提供的*build system APIs* 实现该目标。例如：

```
cmake_minimum_required(VERSION 3.16)
project(my_custom_app C)

# 导入提供 ESP-IDF CMake 构建系统 API 的 CMake 文件
include($ENV{IDF_PATH}/tools/cmake/idf.cmake)

# 在构建中导入 ESP-IDF 组件，可以视作等同 add_subdirectory()
# 但为 ESP-IDF 构建增加额外的构建过程
# 具体构建过程
idf_build_process(esp32)

# 创建项目可执行文件
# 使用其别名 idf::newlib 将其链接到 newlib 组件
add_executable(${CMAKE_PROJECT_NAME}.elf main.c)
target_link_libraries(${CMAKE_PROJECT_NAME}.elf idf::newlib)

# 让构建系统知道项目到可执行文件是什么，从而添加更多的目标以及依赖关系等
idf_build_executable(${CMAKE_PROJECT_NAME}.elf)
```

build_system/cmake/idf_as_lib 中的示例演示了如何在自定义的 CMake 项目创建一个类似于 Hello World 的应用程序。

4.4.20 ESP-IDF CMake 构建系统 API

ESP-IDF 构建命令

```
idf_build_get_property(var property [GENERATOR_EXPRESSION])
```

检索一个构建属性 *property*，并将其存储在当前作用域可访问的 var 中。特定 *GENERATOR_EXPRESSION* 将检索该属性的生成器表达式字符串（不是实际值），它可与支持生成器表达式的 CMake 命令一起使用。

```
idf_build_set_property(property val [APPEND])
```

设置构建属性 *property* 的值为 *val*。特定 *APPEND* 将把指定的值附加到属性当前值之后。如果该属性之前不存在或当前为空，则指定的值将变为第一个元素/成员。

```
idf_build_component(component_dir)
```

向构建系统提交一个包含组件的 *component_dir* 目录。相对路径会被转换为相对于当前目录的绝对路径。所有对该命令的调用必须在 'idf_build_process'之前执行。

该命令并不保证组件在构建过程中会被处理（参见 *idf_build_process* 中 *COMPONENTS* 参数说明）

```
idf_build_process(target
                  [PROJECT_DIR project_dir]
                  [PROJECT_VER project_ver]
                  [PROJECT_NAME project_name]
                  [SDKCONFIG sdkconfig]
                  [SDKCONFIG_DEFAULTS sdkconfig_defaults]
                  [BUILD_DIR build_dir]
                  [COMPONENTS component1 component2 ...])
```

为导入 ESP-IDF 组件执行大量的幕后工作，包括组件配置、库创建、依赖性扩展和解析。在这些功能中，对于用户最重要的可能是通过调用每个组件的 `idf_component_register` 来创建库。该命令为每个组件创建库，这些库可以使用别名来访问，其形式为 idf::*component_name*。这些别名可以用来将组件链接到用户自己的目标、库或可执行文件上。

该调用要求用 *target* 参数指定目标芯片。调用的可选参数包括：

- PROJECT_DIR - 项目目录，默认为 CMAKE_SOURCE_DIR。

- PROJECT_NAME - 项目名称，默认为 CMAKE_PROJECT_NAME。
- PROJECT_VER - 项目的版本/版本号，默认为"1"。
- SDKCONFIG - 生成的 sdkconfig 文件的输出路径，根据是否设置 PROJECT_DIR，默认为 PROJECT_DIR/sdkconfig 或 CMAKE_SOURCE_DIR/sdkconfig。
- SDKCONFIG_DEFAULTS - 包含默认配置的文件列表（列表中必须包含完整的路径），默认为空；对于列表中的每个值 *filename*，如果存在的话，也会加载文件 *filename.target* 中的配置。对于列表中的 *filename* 的每一个值，也会加载文件 *filename.target*（如果存在的话）中的配置。
- BUILD_DIR - 用于放置 ESP-IDF 构建相关工具的目录，如生成的二进制文件、文本文件、组件；默认为 CMAKE_BINARY_DIR。
- COMPONENTS - 从构建系统已知的组件中选择要处理的组件（通过 `idf_build_component` 添加）。这个参数用于精简构建过程。如果在依赖链中需要其它组件，则会自动添加，即自动添加这个列表中组件的公共和私有依赖项，进而添加这些依赖项的公共和私有依赖，以此类推。如果不指定，则会处理构建系统已知的所有组件。

```
idf_build_executable(executable)
```

指定 ESP-IDF 构建的可执行文件 *executable*。这将添加额外的目标，如与 flash 相关的依赖关系，生成额外的二进制文件等。应在 `idf_build_process` 之后调用。

```
idf_build_get_config(var config [GENERATOR_EXPRESSION])
```

获取指定配置的值。就像构建属性一样，特定 *GENERATOR_EXPRESSION* 将检索该配置的生成器表达式字符串，而不是实际值，即可以与支持生成器表达式的 CMake 命令一起使用。然而，实际的配置值只有在调用 `idf_build_process` 后才能知道。

ESP-IDF 构建属性

可以通过使用构建命令 `idf_build_get_property` 来获取构建属性的值。例如，以下命令可以获取构建过程中使用的 Python 解释器的相关信息。

```
idf_build_get_property(python PYTHON)
message(STATUS "The Python interpreter is: ${python}")
```

- BUILD_DIR - 构建目录；由 `idf_build_process` 的 BUILD_DIR 参数设置。
- BUILD_COMPONENTS - 包含在构建中的组件列表；由 `idf_build_process` 设置。
- BUILD_COMPONENT_ALIASES - 包含在构建中的组件的库别名列表；由 `idf_build_process` 设置。
- C_COMPILE_OPTIONS - 适用于所有组件的 C 源代码文件的编译选项。
- COMPILE_OPTIONS - 适用于所有组件的源文件（无论是 C 还是 C++）的编译选项。
- COMPILE_DEFINITIONS - 适用于所有组件源文件的编译定义。
- CXX_COMPILE_OPTIONS - 适用于所有组件的 C++ 源文件的编译选项。
- EXECUTABLE - 项目可执行文件；通过调用 `idf_build_executable` 设置。
- DEPENDENCIES_LOCK - 组件管理器使用的依赖关系锁定文件的路径。默认值为项目路径下的 *dependencies.lock*。
- EXECUTABLE_NAME - 不含扩展名的项目可执行文件的名称；通过调用 `idf_build_executable` 设置。
- EXECUTABLE_DIR - 输出的可执行文件的路径
- IDF_COMPONENT_MANAGER - 默认启用组件管理器，但如果设置这个属性为 `0`，则会被 IDF_COMPONENT_MANAGER 环境变量禁用。
- IDF_PATH - ESP-IDF 路径；由 IDF_PATH 环境变量设置，或者从 `idf.cmake` 的位置推断。
- IDF_TARGET - 构建的目标芯片；由 `idf_build_process` 的目标参数设置。
- IDF_VER - ESP-IDF 版本；由版本文件或 IDF_PATH 仓库的 Git 版本设置。
- INCLUDE_DIRECTORIES - 包含所有组件源文件的目录。
- KCONFIGS - 构建过程中组件里的 Kconfig 文件的列表；由 `idf_build_process` 设置。
- KCONFIG_PROJBUILDS - 构建过程中组件中的 Kconfig.projbuild 文件的列表；由 `idf_build_process` 设置。
- PROJECT_NAME - 项目名称；由 `idf_build_process` 的 PROJECT_NAME 参数设置。
- PROJECT_DIR - 项目的目录；由 `idf_build_process` 的 PROJECT_DIR 参数设置。

- PROJECT_VER - 项目的版本；由 idf_build_process 的 PROJECT_VER 参数设置。
- PYTHON - 用于构建的 Python 解释器；如果有则从 PYTHON 环境变量中设置，如果没有，则使用"python"。
- SDKCONFIG - 输出的配置文件的完整路径；由 idf_build_process SDKCONFIG 参数设置。
- SDKCONFIG_DEFAULTS - 包含默认配置的文件列表；由 idf_build_process SDKCONFIG_DEFAULTS 参数设置。
- SDKCONFIG_HEADER - 包含组件配置的 C/C++ 头文件的完整路径；由 idf_build_process 设置。
- SDKCONFIG_CMAKE - 包含组件配置的 CMake 文件的完整路径；由 idf_build_process 设置。
- SDKCONFIG_JSON - 包含组件配置的 JSON 文件的完整路径；由 idf_build_process 设置。
- SDKCONFIG_JSON_MENUS - 包含配置菜单的 JSON 文件的完整路径；由 idf_build_process 设置。

ESP-IDF 组件命令

```
idf_component_get_property(var component property [GENERATOR_EXPRESSION])
```

检索一个指定的 *component* 的组件属性 *property*，并将其存储在当前作用域可访问的 *var* 中。指定 *GENERATOR_EXPRESSION* 将检索该属性的生成器表达式字符串（不是实际值），它可以在支持生成器表达式的 CMake 命令中使用。

```
idf_component_set_property(component property val [APPEND])
```

设置指定的 *component* 的组件属性，*property* 的值为 *val*。特定 *APPEND* 将把指定的值追加到属性的当前值后。如果该属性之前不存在或当前为空，指定的值将成为第一个元素/成员。

```
idf_component_register([[SRCS src1 src2 ...] | [[SRC_DIRS dir1 dir2 ...] [EXCLUDE_
↪SRCS src1 src2 ...]]
                       [INCLUDE_DIRS dir1 dir2 ...]
                       [PRIV_INCLUDE_DIRS dir1 dir2 ...]
                       [REQUIRES component1 component2 ...]
                       [PRIV_REQUIRES component1 component2 ...]
                       [LDFRAGMENTS ldfragment1 ldfragment2 ...]
                       [REQUIRED_IDF_TARGETS target1 target2 ...]
                       [EMBED_FILES file1 file2 ...]
                       [EMBED_TXTFILES file1 file2 ...]
                       [KCONFIG kconfig]
                       [KCONFIG_PROJBUILD kconfig_projbuild]
                       [WHOLE_ARCHIVE])
```

将一个组件注册到构建系统中。就像 project() CMake 命令一样，该命令应该直接从组件的 CMakeLists.txt 中调用（而不是通过函数或宏），且建议在其他命令之前调用该命令。下面是一些关于在 idf_component_register 之前 不能调用哪些命令的指南：

- 在 CMake 脚本模式下无效的命令。
- 在 project_include.cmake 中定义的自定义命令。
- 除了 idf_build_get_property 之外，构建系统的 API 命令；但要考虑该属性是否有被设置。

对变量进行设置和操作的命令，一般可在 idf_component_register 之前调用。

idf_component_register 的参数包括：

- SRCS - 组件的源文件，用于为组件创建静态库；如果没有指定，组件将被视为仅配置组件，从而创建接口库。
- SRC_DIRS、EXCLUDE_SRCS - 用于通过指定目录来 glob 源文件 (.c、.cpp、.S)，而不是通过 SRCS 手动指定源文件。请注意，这受*CMake 中通配符的限制*。在 EXCLUDE_SRCS 中指定的源文件会从被 glob 的文件中移除。
- INCLUDE_DIRS - 相对于组件目录的路径，该路径将被添加到需要当前组件的所有其他组件的 include 搜索路径中。
- PRIV_INCLUDE_DIRS - 必须是相对于组件目录的目录路径，它仅被添加到这个组件源文件的 include 搜索路径中。

- REQUIRES - 组件的公共组件依赖项。
- PRIV_REQUIRES - 组件的私有组件依赖项；在仅用于配置的组件上会被忽略。
- LDFRAGMENTS - 组件链接器片段文件。
- REQUIRED_IDF_TARGETS - 指定该组件唯一支持的目标。
- KCONFIG - 覆盖默认的 Kconfig 文件。
- KCONFIG_PROJBUILD - 覆盖默认的 Kconfig.projbuild 文件。
- WHOLE_ARCHIVE - 如果指定了此参数，链接时会在组件库的前后分别添加 `-Wl,--whole-archive` 和 `-Wl,--no-whole-archive`。这与设置 `WHOLE_ARCHIVE` 组件属性的效果一致。

以下内容用于将数据嵌入到组件中，并在确定组件是否仅用于配置时被视为源文件。这意味着，即使组件没有指定源文件，如果组件指定了以下其中之一，仍然会在内部为组件创建一个静态库。

- EMBED_FILES - 嵌入组件的二进制文件
- EMBED_TXTFILES - 嵌入组件的文本文件

ESP-IDF 组件属性

组件的属性值可以通过使用构建命令 `idf_component_get_property` 来获取。例如，以下命令可以获取 `freertos` 组件的目录。

```
idf_component_get_property(dir freertos COMPONENT_DIR)
message(STATUS "The 'freertos' component directory is: ${dir}")
```

- COMPONENT_ALIAS - COMPONENT_LIB 的别名，用于将组件链接到外部目标；由 `idf_build_component` 设置，别名库本身由 `idf_component_register` 创建。
- COMPONENT_DIR - 组件目录；由 `idf_build_component` 设置。
- COMPONENT_OVERRIDEN_DIR - 如果这个组件覆盖了另一个组件，则包含原组件的目录。
- COMPONENT_LIB - 所创建的组件静态/接口库的名称；由 `idf_build_component` 设置，库本身由 `idf_component_register` 创建。
- COMPONENT_NAME - 组件的名称；由 `idf_build_component` 根据组件的目录名设置。
- COMPONENT_TYPE - 组件的类型（LIBRARY 或 CONFIG_ONLY）。如果一个组件指定了源文件或嵌入了一个文件，那么它的类型就是 LIBRARY。
- EMBED_FILES - 要嵌入组件的文件列表；由 `idf_component_register` EMBED_FILES 参数设置。
- EMBED_TXTFILES - 要嵌入组件的文本文件列表；由 `idf_component_register` EMBED_TXTFILES 参数设置。
- INCLUDE_DIRS - 组件 include 目录列表；由 `idf_component_register` INCLUDE_DIRS 参数设置。
- KCONFIG - 组件 Kconfig 文件；由 `idf_build_component` 设置。
- KCONFIG_PROJBUILD - 组件 Kconfig.projbuild；由 `idf_build_component` 设置。
- LDFRAGMENTS - 组件链接器片段文件列表；由 `idf_component_register` LDFRAGMENTS 参数设置。
- MANAGED_PRIV_REQUIRES - IDF 组件管理器从 `idf_component.yml` 清单文件中的依赖关系中添加的私有组件依赖关系列表。
- MANAGED_REQUIRES - IDF 组件管理器从 `idf_component.yml` 清单文件的依赖关系中添加的公共组件依赖关系列表。
- PRIV_INCLUDE_DIRS - 组件私有 include 目录列表；在 LIBRARY 类型的组件 `idf_component_register` PRIV_INCLUDE_DIRS 参数中设置。
- PRIV_REQUIRES - 私有组件依赖关系列表；根据 `idf_component_register` PRIV_REQUIRES 参数的值以及 `idf_component.yml` 清单文件中的依赖关系设置。
- REQUIRED_IDF_TARGETS - 组件支持的目标列表；由 `idf_component_register` REQUIRED_IDF_TARGETS 参数设置。
- REQUIRES - 公共组件依赖关系列表；根据 `idf_component_register` REQUIRES 参数的值以及 `idf_component.yml` 清单文件中的依赖关系设置。
- SRCS - 组件源文件列表；由 `idf_component_register` 的 SRCS 或 SRC_DIRS/EXCLUDE_SRCS 参数设置。
- WHOLE_ARCHIVE - 如果该属性被设置为 `TRUE`（或是其他 CMake 布尔“真”值：1、`ON`、`YES`、`Y` 等），链接时会在组件库的前后分别添加 `-Wl,--whole-archive` 和 `-Wl,--no-whole-archive` 选

项。这可以强制链接器将每个目标文件包含到可执行文件中，即使该目标文件没有解析来自应用程序其余部分的任何引用。当组件中包含依赖链接时注册的插件或模块时，通常会使用该方法。默认情况下，此属性为 FALSE。可以从组件的 CMakeLists.txt 文件中将其设置为 TRUE。

4.4.21 文件通配 & 增量构建

在 ESP-IDF 组件中添加源文件的首选方法是在 COMPONENT_SRCS 中手动列出它们:

```
idf_component_register(SRCS library/a.c library/b.c platform/platform.c
                       ...)
```

这是在 CMake 中手动列出源文件的 最佳实践。然而，当有许多源文件都需要添加到构建中时，这种方法就会很不方便。ESP-IDF 构建系统因此提供了另一种替代方法，即使用 SRC_DIRS 来指定源文件:

```
idf_component_register(SRC_DIRS library platform
                       ...)
```

后台会使用通配符在指定的目录中查找源文件。但是请注意，在使用这种方法的时候，如果组件中添加了一个新的源文件，CMake 并不知道重新运行配置，最终该文件也没有被加入构建中。

如果是自己添加的源文件，这种折衷还是可以接受的，因为用户可以触发一次干净的构建，或者运行 idf.py reconfigure 来手动重启 CMake。但是，如果你需要与其他使用 Git 等版本控制工具的开发人员共享项目时，问题就会变得更加困难，因为开发人员有可能会拉取新的版本。

ESP-IDF 中的组件使用了第三方的 Git CMake 集成模块(/tools/cmake/third_party/GetGitRevisionDescription.cmake)，任何时候源码仓库的提交记录发生了改变，该模块就会自动重新运行 CMake。即只要拉取了新的 ESP-IDF 版本，CMake 就会重新运行。

对于不属于 ESP-IDF 的项目组件，有以下几个选项供参考:

- 如果项目文件保存在 Git 中，ESP-IDF 会自动跟踪 Git 修订版本，并在它发生变化时重新运行 CMake。
- 如果一些组件保存在第三方 Git 仓库中（不在项目仓库或 ESP-IDF 仓库），则可以在组件 CMakeLists 文件中调用 git_describe 函数，以便在 Git 修订版本发生变化时自动重启 CMake。
- 如果没有使用 Git，请记住在源文件发生变化时手动运行 idf.py reconfigure。
- 使用 idf_component_register 的 SRCS 参数来列出项目组件中的所有源文件则可以完全避免这一问题。

具体选择哪一方式，就要取决于项目本身，以及项目用户。

4.4.22 构建系统的元数据

为了将 ESP-IDF 集成到 IDE 或者其它构建系统中，CMake 在构建的过程中会在 build/ 目录下生成大量元数据文件。运行 cmake 或 idf.py reconfigure （或任何其它 idf.py 构建命令），可以重新生成这些元数据文件。

- compile_commands.json 是标准格式的 JSON 文件，它描述了在项目中参与编译的每个源文件。CMake 其中的一个功能就是生成此文件，许多 IDE 都知道如何解析此文件。
- project_description.json 包含有关 ESP-IDF 项目、已配置路径等的一些常规信息。
- flasher_args.json 包含 esptool.py 工具用于烧录项目二进制文件的参数，此外还有 flash_*_args 文件，可直接与 esptool.py 一起使用。更多详细信息请参阅*flash 参数*。
- CMakeCache.txt 是 CMake 的缓存文件，包含 CMake 进程、工具链等其它信息。
- config/sdkconfig.json 包含 JSON 格式的项目配置结果。
- config/kconfig_menus.json 是在 menuconfig 中显示菜单的 JSON 格式版本，用于外部 IDE 的 UI。

JSON 配置服务器

`kconfserver` 工具可以帮助 IDE 轻松地与配置系统的逻辑进行集成，它运行在后台，通过使用 stdin 和 stdout 读写 JSON 文件的方式与调用进程交互。

你可以通过 `idf.py confserver` 或 `ninja kconfserver` 从项目中运行 `kconfserver`，也可以使用不同的构建生成器来触发类似的目标。

有关 kconfserver 的更多信息，请参阅 esp-idf-kconfig 文档。

4.4.23 构建系统内部

构建脚本

ESP-IDF 构建系统的列表文件位于 /tools/cmake 中。实现构建系统核心功能的模块如下

- build.cmake - 构建相关命令，即构建初始化、检索/设置构建属性、构建处理。
- component.cmake - 组件相关的命令，如添加组件、检索/设置组件属性、注册组件。
- kconfig.cmake - 从 Kconfig 文件中生成配置文件（sdkconfig、sdkconfig.h、sdkconfig.cmake 等）。
- ldgen.cmake - 从链接器片段文件生成最终链接器脚本。
- target.cmake - 设置构建目标和工具链文件。
- utilities.cmake - 其它帮助命令。

除了这些文件，还有两个重要的 CMake 脚本在 /tools/cmake 中：

- idf.cmake - 设置构建参数并导入上面列出的核心模块。之所以包括在 CMake 项目中，是为了方便访问 ESP-IDF 构建系统功能。
- project.cmake - 导入 `idf.cmake`，并提供了一个自定义的 `project()` 命令，该命令负责处理建立可执行文件时所有的繁重工作。包含在标准 ESP-IDF 项目的顶层 CMakeLists.txt 中。

/tools/cmake 中的其它文件都是构建过程中的支持性文件或第三方脚本。

构建过程

本节介绍了标准的 ESP-IDF 应用构建过程。构建过程可以大致分为四个阶段：

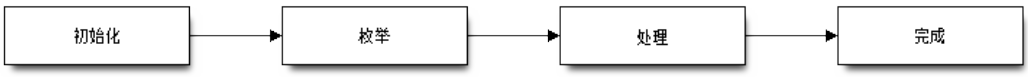


图 2: ESP-IDF Build System Process

初始化

该阶段为构建设置必要的参数。

- **在将 `idf.cmake` 导入 `project.cmake` 后，将执行以下步骤：**
  - 在环境变量中设置 `IDF_PATH` 或从顶层 CMakeLists.txt 中包含的 `project.cmake` 路径推断相对路径。
  - 将 /tools/cmake 添加到 `CMAKE_MODULE_PATH` 中，并导入核心模块和各种辅助/第三方脚本。
  - 设置构建工具/可执行文件，如默认的 Python 解释器。
  - 获取 ESP-IDF git 修订版，并存储为 `IDF_VER`。
  - 设置全局构建参数，即编译选项、编译定义、包括所有组件的 include 目录。
  - 将 components 中的组件添加到构建中。

- **自定义 `project()` 命令的初始部分执行以下步骤：**
  - 在环境变量或 CMake 缓存中设置 `IDF_TARGET` 以及设置相应要使用的 `CMAKE_TOOLCHAIN_FILE`。
  - 添加 `EXTRA_COMPONENT_DIRS` 中的组件至构建中
  - 从 `COMPONENTS`/`EXCLUDE_COMPONENTS`、`SDKCONFIG`、`SDKCONFIG_DEFAULTS` 等变量中为调用命令 `idf_build_process()` 准备参数。

调用 `idf_build_process()` 命令标志着这个阶段的结束。

枚举

这个阶段会建立一个需要在构建过程中处理的组件列表，该阶段在 `idf_build_process()` 的前半部分进行。

- 检索每个组件的公共和私有依赖。创建一个子进程，以脚本模式执行每个组件的 CMakeLists.txt。`idf_component_register` REQUIRES 和 PRIV_REQUIRES 参数的值会返回给父进程。这就是所谓的早期扩展。在这一步中定义变量 `CMAKE_BUILD_EARLY_EXPANSION`。
- 根据公共和私有的依赖关系，递归地导入各个组件。

处理

该阶段处理构建中的组件，是 `idf_build_process()` 的后半部分。

- 从 sdkconfig 文件中加载项目配置，并生成 sdkconfig.cmake 和 sdkconfig.h 头文件。这两个文件分别定义了可以从构建脚本和 C/C++ 源文件/头文件中访问的配置变量/宏。
- 导入各组件的 `project_include.cmake`。
- 将每个组件添加为一个子目录，处理其 CMakeLists.txt。组件 CMakeLists.txt 调用注册命令 `idf_component_register` 添加源文件、导入目录、创建组件库、链接依赖关系等。

完成

该阶段是 `idf_build_process()` 剩余的步骤。

- 创建可执行文件并将其链接到组件库中。
- 生成 project_description.json 等项目元数据文件并且显示所建项目等相关信息。

请参考 /tools/cmake/project.cmake 获取更多信息。

4.4.24 从 ESP-IDF GNU Make 构建系统迁移到 CMake 构建系统

ESP-IDF CMake 构建系统与旧版的 GNU Make 构建系统在某些方面非常相似，开发者都需要提供 include 目录、源文件等。然而，有一个语法上的区别，即对于 ESP-IDF CMake 构建系统，开发者需要将这些作为参数传递给注册命令 `idf_component_register`。

自动转换工具

在 ESP-IDF v4.x 版本中，*tools/cmake/convert_to_cmake.py* 提供了项目自动转换工具。由于该脚本依赖于 *make* 构建系统，所以 v5.0 版本中不包含该脚本。

CMake 中不可用的功能

有些功能已从 CMake 构建系统中移除，或者已经发生很大改变。GNU Make 构建系统中的以下变量已从 CMake 构建系统中删除：

- `COMPONENT_BUILD_DIR`：由 `CMAKE_CURRENT_BINARY_DIR` 替代。

- COMPONENT_LIBRARY：默认为 $(COMPONENT_NAME).a 但是库名可以被组件覆盖。在 CMake 构建系统中，组件库名称不可再被组件覆盖。
- CC、LD、AR、OBJCOPY：gcc xtensa 交叉工具链中每个工具的完整路径。CMake 使用 CMAKE_C_COMPILER、CMAKE_C_LINK_EXECUTABLE 和 CMAKE_OBJCOPY 进行替代。完整列表请参阅 CMake 语言变量。
- HOSTCC、HOSTLD、HOSTAR：宿主机本地工具链中每个工具的全名。CMake 系统不再提供此变量，外部项目需要手动检测所需的宿主机工具链。
- COMPONENT_ADD_LDFLAGS：用于覆盖链接标志。CMake 中使用 target_link_libraries 命令替代。
- COMPONENT_ADD_LINKER_DEPS：链接过程依赖的文件列表。target_link_libraries 通常会自动推断这些依赖。对于链接脚本，可以使用自定义的 CMake 函数 target_linker_scripts。
- COMPONENT_SUBMODULES：不再使用。CMake 会自动枚举 ESP-IDF 仓库中所有的子模块。
- COMPONENT_EXTRA_INCLUDES：曾是 COMPONENT_PRIV_INCLUDEDIRS 变量的替代版本，仅支持绝对路径。CMake 系统中统一使用 COMPONENT_PRIV_INCLUDEDIRS（可以是相对路径，也可以是绝对路径）。
- COMPONENT_OBJS：以前，可以以目标文件列表的方式指定组件源，现在，可以通过 COMPONENT_SRCS 以源文件列表的形式指定组件源。
- COMPONENT_OBJEXCLUDE：已被 COMPONENT_SRCEXCLUDE 替换。用于指定源文件（绝对路径或组件目录的相对路径）。
- COMPONENT_EXTRA_CLEAN：已被 ADDITIONAL_MAKE_CLEAN_FILES 属性取代，注意，*CMake 对此项功能有部分限制*。
- COMPONENT_OWNBUILDTARGET & COMPONENT_OWNCLEANTARGET：已被 CMake 外部项目 *<ExternalProject_>* 替代，详细内容请参阅完全覆盖组件的构建过程。
- COMPONENT_CONFIG_ONLY：已被 register_config_only_component() 函数替代，请参阅仅配置组件。
- CFLAGS、CPPFLAGS、CXXFLAGS：已被相应的 CMake 命令替代，请参阅组件编译控制。

无默认值的变量

以下变量不再具有默认值：

- 源目录（Make 中的 COMPONENT_SRCDIRS 变量，CMake 中 idf_component_register 的 SRC_DIRS 参数）
- include 目录（Make 中的 COMPONENT_ADD_INCLUDEDIRS 变量，CMake 中 idf_component_register 的 INCLUDE_DIRS 参数）

不再需要的变量

在 CMake 构建系统中，如果设置了 COMPONENT_SRCS，就不需要再设置 COMPONENT_SRCDIRS。实际上，CMake 构建系统中如果设置了 COMPONENT_SRCDIRS，那么 COMPONENT_SRCS 就会被忽略。

从 Make 中烧录

仍然可以使用 make flash 或者类似的目标来构建和烧录，但是项目 sdkconfig 不能再用来指定串口和波特率。可以使用环境变量来覆盖串口和波特率的设置，详情请参阅使用 *Ninja/Make* 来烧录。

4.5 C Support

ESP-IDF is primarily written in C and provides C APIs. Newlib is used as standard C library (the Newlib version can be found in newlib/sbom.yml). In general, all C features supported by the compiler, currently GCC, should be available in ESP-IDF, unless specified in *Unsupported C Features* below.

4.5.1 C Version

GNU dialect of ISO C17 (`--std=gnu17`) is the current default C version in ESP-IDF.

To compile the source code of a certain component using a different language standard, set the desired compiler flag in the component's `CMakeLists.txt` file:

```
idf_component_register( ... )
target_compile_options(${COMPONENT_LIB} PRIVATE -std=gnu11)
```

If the public header files of the component also need to be compiled with the same language standard, replace the flag `PRIVATE` with `PUBLIC`.

4.5.2 Unsupported C Features

The following features are not supported in ESP-IDF.

Nested Function Pointers

The **GNU dialect of ISO C17** supports nested functions. However, they do not work in ESP-IDF when referenced as function pointer because the compiler generates a trampoline on the stack, while the stack is not executable in ESP-IDF. Hence, do not use function pointers to nested functions.

4.6 C++ 支持

ESP-IDF 主要使用 C 语言编写，并提供 C 语言 API。但 ESP-IDF 也支持使用 C++ 开发应用程序，与 C++ 开发相关的各种主题在本文档中列出。

ESP-IDF 支持以下 C++ 功能：

- 异常处理
- 多线程
- 运行时类型信息 *(RTTI)*
- 线程局部存储 （`thread_local` 关键字）
- 除部分限制，所有由 GCC 实现的 C++ 功能均受支持。有关由 GCC 所实现功能的详细信息，请参阅 GCC 文档。

4.6.1 `esp-idf-cxx` 组件

esp-idf-cxx 组件为一些 ESP-IDF 中的功能提供了更高级别的 C++ API，该组件可以从 ESP-IDF 组件注册表 中获取。

4.6.2 C++ 语言标准

默认情况下，ESP-IDF 使用 C++23 语言标准和 GNU 扩展 (`-std=gnu++23`) 编译 C++ 代码。

要使用其他语言标准编译特定组件的源代码，请按以下步骤，在组件的 CMakeLists.txt 文件中设置所需的编译器标志：

```
idf_component_register( ... )
target_compile_options(${COMPONENT_LIB} PRIVATE -std=gnu++11)
```

如果组件的公共头文件也需要以该语言标准编译，请使用 `PUBLIC` 而非 `PRIVATE`。

4.6.3 多线程

支持 C++ 线程，互斥锁和条件变量。C++ 线程基于 pthread 构建，而 pthread 封装了 FreeRTOS 任务。

有关在 C++ 中创建线程的示例，请参阅 cxx/pthread。

备注： std::jthread 的析构函数只能从线程 *API* 或 C++ 线程库 API 创建的任务中安全地调用。

4.6.4 异常处理

ESP-IDF 默认禁用对 C++ 异常处理的支持，可以用*CONFIG_COMPILER_CXX_EXCEPTIONS* 选项启用该支持。

如果抛出了异常处理，却没有相应的 `catch` 块，程序将由 `abort` 函数终止，并打印回溯信息。有关回溯信息的更多信息，请参见严重错误。

C++ 异常处理应 **仅**应用于异常情况，即意外情况及罕见情况，如发生频率低于 1% 的事件。**请勿**将 C++ 异常处理用于流程控制，详情请参阅下文的资源使用部分。有关使用 C++ 异常处理的更多详情，请参阅 ISO C++ FAQ 和 CPP 核心指南。

有关 C++ 异常处理的示例，请参阅 cxx/exceptions。

C++ 异常处理及所需资源

启用异常处理后，应用程序的二进制文件通常会增加几个 KB。

此外，可能需要为异常处理应急内存池保留一部分 RAM。如果无法从堆内存中分配异常处理对象，则会使用该池中的内存。

使用*CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE* 变量可以设置异常处理应急内存池的内存量。

当且仅当 C++ 异常抛出时，会使用额外的栈内存（约 200 字节），从而从栈内存顶部调用函数，启动异常处理。

使用 C++ 异常处理的代码的运行时间取决于运行时实际发生的情况。

- 如果没有抛出异常，则异常处理的代码运行速度会更快，因为无需检查错误代码。
- 如果抛出异常，异常处理代码的运行时间会比返回错误代码的代码长几个数量级。

如果抛出异常，解开栈代码的速度要比返回错误代码慢好几个数量级。所增加的运行时长取决于应用程序的要求和错误处理的实现方式（例如，是否需要用户输入或发送消息到云端）。因此，在实时关键的代码路径中，不应使用会抛出异常的代码。

4.6.5 运行时类型信息 (RTTI)

ESP-IDF 默认禁用对 RTTI 的支持，可以用*CONFIG_COMPILER_CXX_RTTI* 选项启用该支持。

启用此选项，将以启用了 RTTI 支持的方式编译所有的 C++ 文件，并支持使用 `dynamic_cast` 转换和 `typeid` 运算符。启用此选项通常会增加几十 KB 的二进制文件大小。

有关在 ESP-IDF 中使用 RTTI 的示例，请参阅 cxx/rtti。

4.6.6 在 C++ 中进行开发

以下部分提供了在 C++ 中开发 ESP-IDF 应用程序的一些技巧。

组合 C 和 C++ 代码

当应用程序的不同部分使用 C 和 C++ 开发时，理解 语言链接性 的概念非常重要。

为了能够从 C 代码中调用 C++ 函数，该 C++ 函数必须使用 C 链接 (`extern "C"`) 进行 **声明**和 **定义**：

```
// 在 .h 文件中声明：
#ifdef __cplusplus
extern "C" {
#endif

void my_cpp_func(void);

#ifdef __cplusplus
}
#endif

// 在 .cpp 文件中进行定义：
extern "C" void my_cpp_func(void) {
    // ...
}
```

为了能够从 C++ 中调用 C 函数，该 C 函数必须使用 C 链接 **声明**：

```
// 在 .h 文件中声明：
#ifdef __cplusplus
extern "C" {
#endif

void my_c_func(void);

#ifdef __cplusplus
}
#endif

// 在 .c 文件中进行定义：
void my_c_func(void) {
    // ...
}
```

在 C++ 中定义 app_main

ESP-IDF 希望应用程序入口点 `app_main` 以 C 链接定义。当 `app_main` 在.cpp 源文件中定义时，必须以 `extern "C"` 标识：

```
extern "C" void app_main()
{
}
```

指定初始化器

许多 ESP-IDF 组件会以`配置结构体` 作为初始化函数的参数。用 C 编写的 ESP-IDF 示例通常使用 指定初始化器，以可读且可维护的方式填充有关结构体。

C 和 C++ 语言对于指定初始化器有不同的规则。例如，C++23（当前在 ESP-IDF 中默认使用）不支持无序指定初始化、嵌套指定初始化、混合使用指定初始化器和常规初始化器，而对数组进行指定初始化。因此，当将 ESP-IDF 的 C 示例移植到 C++ 时，可能需要对结构体初始化器进行一些更改。详细信息请参阅 C++ aggregate initialization reference。

iostream

ESP-IDF 支持 iostream 功能，但应注意：

1. ESP-IDF 在构建过程中通常会删除未使用的代码。然而，在使用 iostreams 的情况下，仅在其中一个源文件包含 <iostream> 头文件就会使二进制文件增加大约 200 kB。
2. ESP-IDF 默认使用简单的非阻塞机制来处理标准输入流 (stdin)。要获得 std::cin 的常规行为，应用程序必须初始化 UART 驱动程序，并启用阻塞模式，详情请参阅 common_components/protocol_examples_common/stdin_out.c。

4.6.7 限制

- 链接脚本生成器不支持将具有 C++ 链接的函数单独放置在内存的特定位置。
- 当与模板函数一起使用时，会忽略各种节属性（例如 IRAM_ATTR）。
- vtable 位于 flash 中，在禁用 flash 缓存时无法访问。因此，在*IRAM 安全中断处理程序* 中应避免调用虚拟函数。目前尚无法使用链接器脚本生成器调整 vtable 的放置位置。
- 不支持 C++ 文件系统 (std::filesystem) 功能。

4.6.8 注意事项

请勿在 C++ 中使用 setjmp/longjmp。longjmp 会在不调用任何析构函数的情况下盲目跳出堆栈，容易引起未定义的行为和内存泄漏。请改用 C++ 异常处理，这类程序可以确保正确调用析构函数。如果无法使用 C++ 异常处理，请使用其他替代方案（setjmp/longjmp 除外），如简单的返回码。

4.7 核心转储

4.7.1 概述

核心转储是软件发生致命错误时，由紧急处理程序自动保存的一组软件状态信息。核心转储有助于对故障进行事后分析，了解软件状态。ESP-IDF 支持生成核心转储。

核心转储包含了系统中所有任务在发生故障时的快照，每个快照都包括任务的控制块 (TCB) 和栈信息。通过分析任务快照，可以确定是哪个任务、在哪个指令（代码行），以及该任务的哪个调用栈导致了系统崩溃。如果将某些变量赋予特殊的核心转储属性，还可以转储这些变量的内容。

核心转储数据会按照特定格式保存在核心转储文件中，详情请参阅*核心转储镜像文件详解*。然而，ESP-IDF 的 idf.py 命令提供了专门的子命令，用于解码和分析核心转储文件。

4.7.2 配置

目标

选项*CONFIG_ESP_COREDUMP_TO_FLASH_OR_UART* 可以启用或禁用核心转储，并在启用时选择核心转储的目标。发生崩溃时，生成的核心转储文件可以保存到 flash 中，也可以通过 UART 输出到连接的主机上。

格式和大小

选项*CONFIG_ESP_COREDUMP_DATA_FORMAT* 控制核心转储文件格式，即 ELF 格式或二进制格式。

ELF 格式具备扩展特性，支持在发生崩溃时保存更多关于错误任务和崩溃软件的信息，但使用 ELF 格式会使核心转储文件变大。建议在新的软件设计中使用此格式，该格式足够灵活，可以在未来的修订版本中进行扩展，保存更多信息。

出于兼容性考虑，核心转储文件保留二进制格式。二进制格式的核心转储文件更小，性能更优。

选项*CONFIG_ESP_COREDUMP_MAX_TASKS_NUM* 配置核心转储保存的任务快照数量。

通过 `Components > Core dump > Core dump data integrity check` 选项可进行核心转储数据完整性检查。

保留栈大小

核心转储例程需要解析并保存所有其他任务的栈，因此会从单独的栈中运行。选项*CONFIG_ESP_COREDUMP_STACK_SIZE* 控制核心转储栈大小，以字节数表示。

将此选项设置为 0 字节将使核心转储例程从 ISR 栈中运行，从而节省内存。将选项设置为大于零的值将创建一个独立的栈。

备注： 如果使用了独立的栈，建议栈大小应大于 1300 字节，确保核心转储例程本身不会导致栈溢出。

核心转储内存区域

核心转储默认保存 CPU 寄存器、任务数据和崩溃原因。选择*CONFIG_ESP_COREDUMP_CAPTURE_DRAM* 选项后，`.bss` 段和 `.data` 段以及 `heap` 数据也将保存到转储中。

推荐将上面提到的几个数据段都保存到核心转储中，以方便调试。但这会导致核心转储文件变大，具体所需的额外存储空间取决于应用程序使用的 DRAM 大小。

备注： 除了崩溃任务的 TCB 和栈外，位于外部 RAM 中的数据不会保存到核心转储文件中，包括使用 `EXT_RAM_BSS_ATTR` 或 `EXT_RAM_NOINIT_ATTR` 属性定义的变量，以及存储在 `extram_bss` 段中的任何数据。

备注： 该功能仅在使用 ELF 文件格式时可用。

4.7.3 将核心转储保存到 flash

将核心转储文件保存至 flash 时，这些文件会保存到 flash 上的特殊分区。指定核心转储分区可以在 flash 芯片上预留空间来存储核心转储文件。

使用 ESP-IDF 提供的默认分区表时，核心转储分区会自动声明。但使用自定义分区表时，请按如下示例进行核心转储分区声明：

```
# 名称,     类型, 子类型,     偏移量,     大小
# 注意: 如果增加了引导加载程序大小, 请及时更新偏移量, 避免产生重叠
nvs,       data, nvs,      0x9000,  0x6000
phy_init,  data, phy,      0xf000,  0x1000
factory,   app,  factory,  0x10000, 1M
coredump,  data, coredump,,         64K
```

重要： 如果设备启用了*flash 加密*，请在核心转储分区中添加 `encrypted` 标志。请注意，使用 `idf.py coredump-info` 或 `idf.py coredump-debug` 命令无法从加密分区读取核心转储。建议使用 `idf.py coredump-info -c <path-to-core-dump>` 命令从 ESP 设备侧读取核心转储，ESP 设备会自动解密分区并发送到相应位置用于分析。

```
coredump, data, coredump,,        64K, encrypted
```

分区命名没有特殊要求，可以根据应用程序的需要选择。但分区类型应为 `data`，子类型应为 `coredump`。此外，在选择分区大小时需注意，核心转储的数据结构会产生 20 字节的固定开销和 12 字节的单任务开销，此开销不包括每个任务的 TCB 和栈的大小。因此，分区大小应至少为 `20 + 最大任务数 x (12 + TCB 大小 + 最大任务栈大小)` 字节。

用于分析 flash 中核心转储的常用命令，可参考以下示例：

```
idf.py coredump-info
```

或

```
idf.py coredump-debug
```

备注： `idf.py coredump-info` 命令和 `idf.py coredump-debug` 命令对 *esp-coredump* 工具进行了封装，可以在 ESP-IDF 环境中轻松使用。更多信息，请参考核心转储命令。

4.7.4 将核心转储保存到 UART

当核心转储文件输出到 UART 时，输出文件会以 Base64 编码方式呈现。通过*CONFIG_ESP_COREDUMP_DECODE* 选项，可以选择 ESP-IDF 监视器对输出文件自动解码，或保持编码状态等待手动解码。

自动解码

如果设置*CONFIG_ESP_COREDUMP_DECODE*，使其自动解码 UART 核心转储文件，ESP-IDF 监视器会自动解码数据，将所有函数地址转换为源代码行，并在监视器中显示相应信息。ESP-IDF 监视器会输出类似以下内容：

此外，选项*CONFIG_ESP_COREDUMP_UART_DELAY* 支持在将核心转储文件输出到 UART 前添加延迟。

```
================================================================
==================== ESP32 CORE DUMP START ====================

Crashed task handle: 0x3ffafba0, name: 'main', GDB name: 'process 1073413024'
Crashed task is not in the interrupt context
Panic reason: abort() was called at PC 0x400d66b9 on core 0

================== CURRENT THREAD REGISTERS ===================
exccause       0x1d (StoreProhibitedCause)
excvaddr       0x0
epc1           0x40084013
epc2           0x0
...
==================== CURRENT THREAD STACK =====================
#0  0x4008110d in panic_abort (details=0x3ffb4f0b "abort() was called at PC␣
↪0x400d66b9 on core 0") at /builds/espressif/esp-idf/components/esp_system/panic.
↪c:472
#1  0x4008510c in esp_system_abort (details=0x3ffb4f0b "abort() was called at PC␣
↪0x400d66b9 on core 0") at /builds/espressif/esp-idf/components/esp_system/port/
↪esp_system_chip.c:93
...
======================== THREADS INFO =========================
  Id   Target Id          Frame
* 1    process 1073413024 0x4008110d in panic_abort (details=0x3ffb4f0b "abort()␣
↪was called at PC 0x400d66b9 on core 0") at /builds/espressif/esp-idf/components/
↪esp_system/panic.c:472
```

(下页继续)

(续上页)

```
  2    process 1073413368 vPortTaskWrapper (pxCode=0x0, pvParameters=0x0) at /
↪builds/espressif/esp-idf/components/freertos/FreeRTOS-Kernel/portable/xtensa/
↪port.c:133
...
       TCB              NAME PRIO C/B  STACK USED/FREE
---------- ---------------- -------- ----------------
0x3ffafba0              main      1/1         368/3724
0x3ffafcf8             IDLE0      0/0         288/1240
0x3ffafe50             IDLE1      0/0         416/1108
...
==================== THREAD 1 (TCB: 0x3ffafba0, name: 'main') ====================
#0  0x4008110d in panic_abort (details=0x3ffb4f0b "abort() was called at PC␣
↪0x400d66b9 on core 0") at /builds/espressif/esp-idf/components/esp_system/panic.
↪c:472
#1  0x4008510c in esp_system_abort (details=0x3ffb4f0b "abort() was called at PC␣
↪0x400d66b9 on core 0") at /builds/espressif/esp-idf/components/esp_system/port/
↪esp_system_chip.c:93
...
==================== THREAD 2 (TCB: 0x3ffafcf8, name: 'IDLE0')␣
↪====================
#0  vPortTaskWrapper (pxCode=0x0, pvParameters=0x0) at /builds/espressif/esp-idf/
↪components/freertos/FreeRTOS-Kernel/portable/xtensa/port.c:133
#1  0x40000000 in ?? ()
...
======================== ALL MEMORY REGIONS ========================
Name   Address   Size   Attrs
...
.iram0.vectors 0x40080000 0x403 R XA
.iram0.text 0x40080404 0xb8ab R XA
.dram0.data 0x3ffb0000 0x2114 RW A
...
====================== ESP32 CORE DUMP END ======================
================================================================
```

手动解码

如果设置*CONFIG_ESP_COREDUMP_DECODE* 为不解码，则在以下 UART 输出的页眉和页脚之间，将输出核心转储的原始 Base64 编码正文：

```
================= CORE DUMP START =================
<将 Base64 编码的核心转储内容解码，并将其保存到磁盘文件中>
================= CORE DUMP END ===================
```

建议将核心转储文本主体手动保存到文件，`CORE DUMP START` 和 `CORE DUMP END` 行不应包含在核心转储文本文件中。随后，可以使用以下命令解码保存的文本：

```
idf.py coredump-info -c </path/to/saved/base64/text>
```

或

```
idf.py coredump-debug -c </path/to/saved/base64/text>
```

4.7.5 核心转储命令

ESP-IDF 提供了一些特殊命令，有助于检索和分析核心转储：

- `idf.py coredump-info` - 打印崩溃任务的寄存器、调用栈、系统可用任务列表、内存区域以及核心转储中存储的内存内容（包括 TCB 和栈）。

- `idf.py coredump-debug` - 创建核心转储 ELF 文件，并使用该文件运行 GDB 调试会话。你可以手动检查内存、变量和任务状态。请注意，由于并未将所有内存保存在核心转储中，因此只有在栈上分配的变量的值才有意义。

高阶用户如果需要传递额外参数或使用自定义 ELF 文件，可直接使用 esp-coredump 工具。如果在 ESP-IDF 环境中使用该工具，可运行如下命令查询更多信息：

```
esp-coredump --help
```

4.7.6 回溯中的 ROM 函数

程序崩溃时，某些任务和/或崩溃任务本身的调用栈中可能包含一或多个 ROM 函数。由于 ROM 不是程序 ELF 的一部分，而 GDB 需要分析函数序言来解码回溯，因此 GDB 无法解析这些调用栈。因此，在遇到第一个 ROM 函数时，调用栈解析将中断并报错。

为解决这一问题，ESP-IDF 监视器会根据目标芯片及其修订版本自动加载乐鑫提供的 ROM ELF。有关 ROM ELF 的详细信息，请参阅 esp-rom-elfs。

4.7.7 按需转储变量

通过读取变量的最后一个值，可以了解崩溃发生的根本原因。核心转储支持通过为已声明的变量添加特殊标记，在 GDB 上检索变量数据。

支持的标记和 RAM 区域

- `COREDUMP_DRAM_ATTR` 将变量放置在 DRAM 区域，该区域包含在转储中。
- `COREDUMP_RTC_ATTR` 将变量放置在 RTC 区域，该区域包含在转储中。
- `COREDUMP_RTC_FAST_ATTR` 将变量放置在 RTC_FAST 区域，该区域包含在转储中。

示例

1. 在 *项目配置菜单* 中启用 *COREDUMP TO FLASH*，随后保存并退出。
2. 在项目中，创建如下全局变量，放置在 DRAM 区域：

```
// uint8_t global_var;
COREDUMP_DRAM_ATTR uint8_t global_var;
```

3. 在主应用程序中，将该变量设置为任意值，并以 `assert(0)` 引发崩溃。

```
global_var = 25;
assert(0);
```

4. 在目标设备上构建、烧写并运行应用程序，等待转储信息。
5. 运行以下命令，在 GDB 中开始核心转储，其中 `PORT` 是设备的 USB 端口：

```
idf.py coredump-debug
```

6. 在 GDB shell 中，输入 `p global_var` 获取变量内容：

```
(gdb) p global_var
$1 = 25 '\031'
```

4.7.8 运行 idf.py coredump-info 和 idf.py coredump-debug

要获取更多有关使用方法的详情，请运行 `idf.py coredump-info --help` 和 `idf.py coredump-debug --help` 命令。

相关文档

核心转储镜像文件详解

核心转储文件的格式可以配置为使用 ELF 格式或传统的二进制格式。建议在所有新设计中使用 ELF 格式，该格式在发生崩溃时会提供更多关于软件状态的信息，例如 CPU 寄存器和内存内容。

内存状态包含程序内存空间中映射的所有任务的快照；CPU 状态包含核心转储生成时的寄存器值。核心转储文件使用 ELF 结构的子集注册这些信息。

可加载的 ELF 段用于存储进程的内存状态，ELF 注释 (`ELF.PT_NOTE`) 用于存储进程的元数据（如 pid、寄存器、信号等）。CPU 状态则存储在一个具有特殊名称 (`CORE`) 和类型 (`NT_PRSTATUS type`) 的注释中。

下图展示了核心转储的结构：

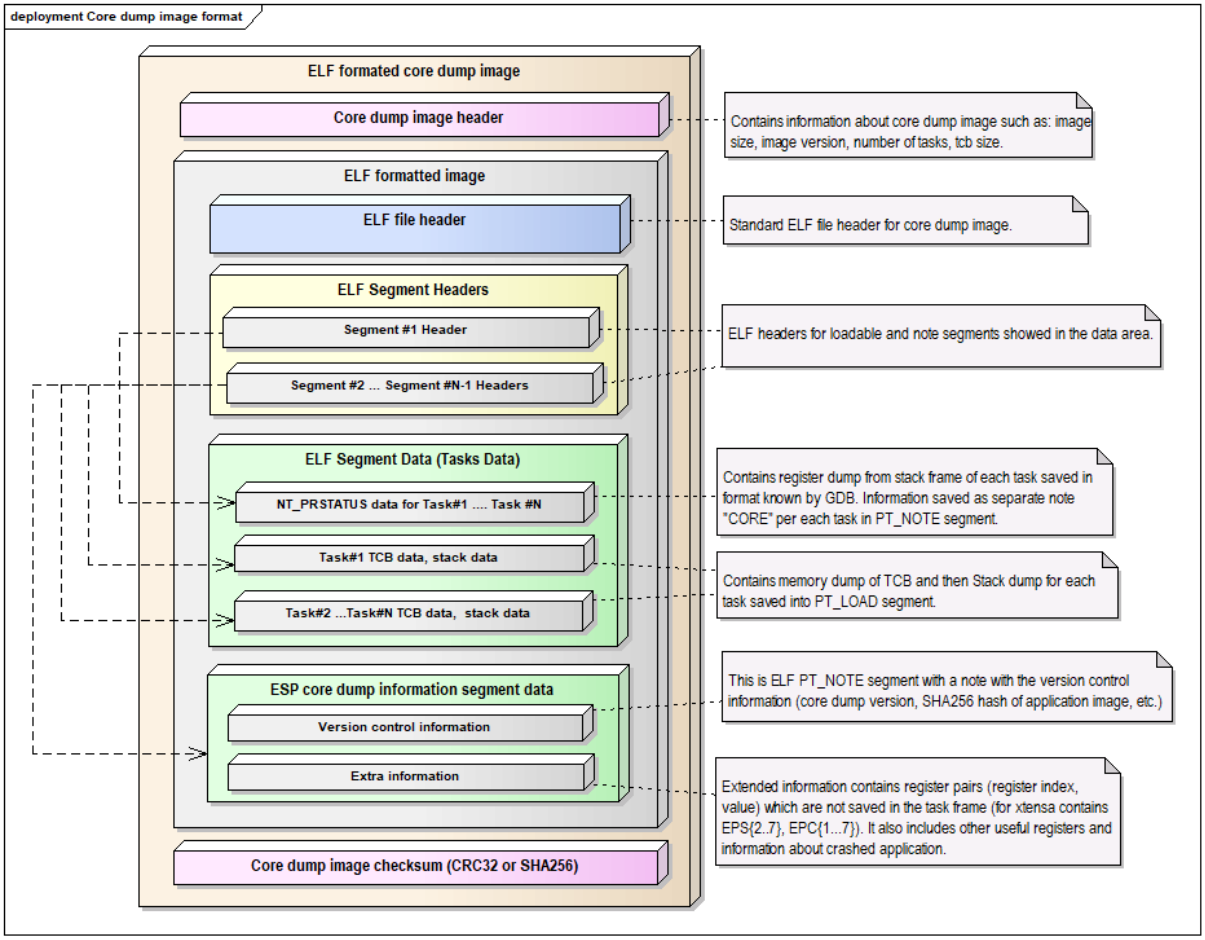


图 3: 核心转储 ELF 镜像文件格式

备注： 上图仅展示当前版本镜像的文件格式，在未来的发布版本中可能会发生变化。

核心转储实现 下图展示了核心转储实现的基本情况：

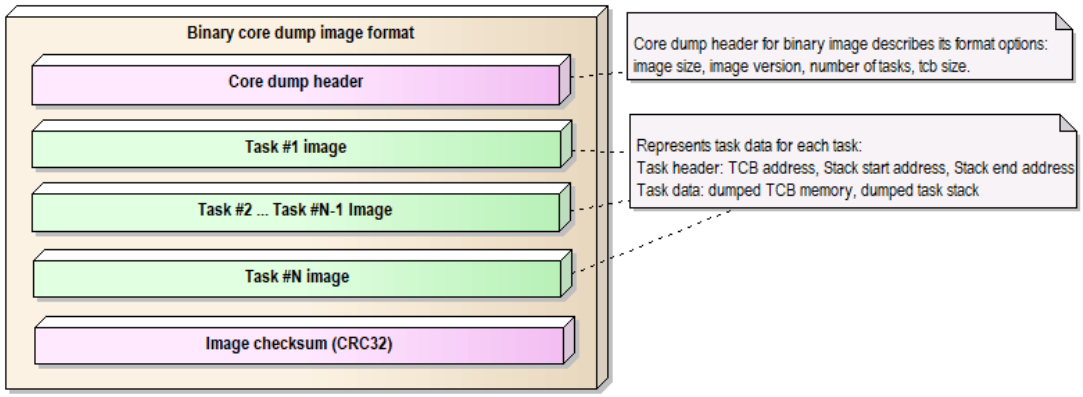


图 4: 核心转储二进制镜像文件格式

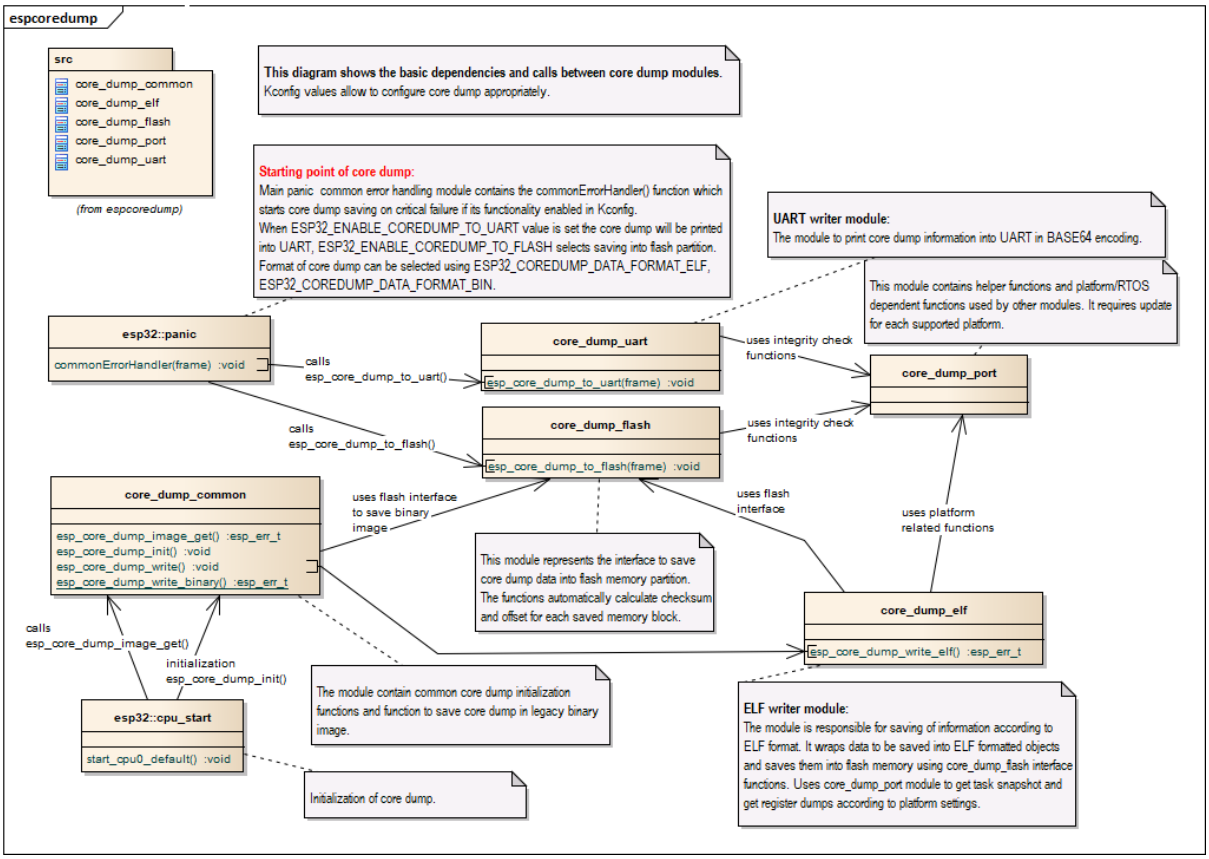


图 5: 核心转储实现

备注： 上图隐藏了部分细节，仅展示当前版本的核心转储实现，在未来的发布版本中可能会发生变化。

4.8 Current Consumption Measurement of Modules

You may want to know the current consumption of a module in deep-sleep mode, *other power-saving modes*, and active mode to develop some applications sensitive to power consumption. This section introduces how to measure the current consumption of a module running such an application.

4.8.1 Notes to Measurement

Can We Use a Development Board?

For ESP32-P4, using a development board directly to measure current consumption of the corresponding module is not recommended, as some circuits still consume power on the board even when you flash the chip with the deep_sleep example. Therefore, you need to cut off the power supply circuit to the module to measure the module's current. This method is inconvenient and increases measurement costs.

How to Choose an Appropriate Ammeter?

In the deep_sleep example, the module will be woken up every 20 seconds. In deep-sleep mode, the current in the module is just several microamps (μA), while in active mode, the current is in the order of milliamps (mA). The high dynamic current range makes accurate measurement difficult. Ordinary ammeters cannot dynamically switch the measurement range fast enough.

Additionally, ordinary ammeters have a relatively high internal resistance, resulting in a significant voltage drop. This may cause the module to enter an unstable state, as it is powered by a voltage smaller than the minimum required voltage supply.

Therefore, an ammeter suitable for measuring current in deep-sleep mode should have low internal resistance and, ideally, switch current ranges dynamically. We recommend two options: the Joulescope ammeter and the Power Profiler Kit II from Nordic.

Joulescope Ammeter The Joulescope ammeter combines high-speed sampling and rapid dynamic current range switching to provide accurate and seamless current and energy measurements, even for devices with rapidly varying current consumption. Joulescope accurately measures electrical current over nine orders of magnitude from amps down to nanoamps. This wide range allows for accurate and precise current measurements for devices. Additionally, Joulescope has a total voltage drop of 25 mV at 1 A, which keeps the module running normally. These two features make Joulescope a perfect option for measuring the module switching between deep-sleep mode and wake-up mode.

Joulescope has no display screen. You need to connect it to a PC to visualize the current waveforms of the measured module. For specific instructions, please follow the documentation provided by the manufacturer.

Nordic Power Profiler Kit II The Nordic Power Profiler Kit II has an advanced analog measurement unit with a high dynamic measurement range. This allows for accurate power consumption measurements for the entire range typically seen in low-power embedded applications, all the way from single μAs to 1 A. The resolution varies between 100 nA and 1 mA, depending on the measurement range, and is high enough to detect small spikes often seen in low-power optimized systems.

4.8.2 Hardware Connection

To measure the power consumption of a bare module, you need an ESP-Prog to flash the deep_sleep example to the module and power the module during measurement, a suitable ammeter (here we use the Joulescope ammeter), a

computer, and of course a bare module with necessary jumper wires. For the connection, please refer to the following figure.

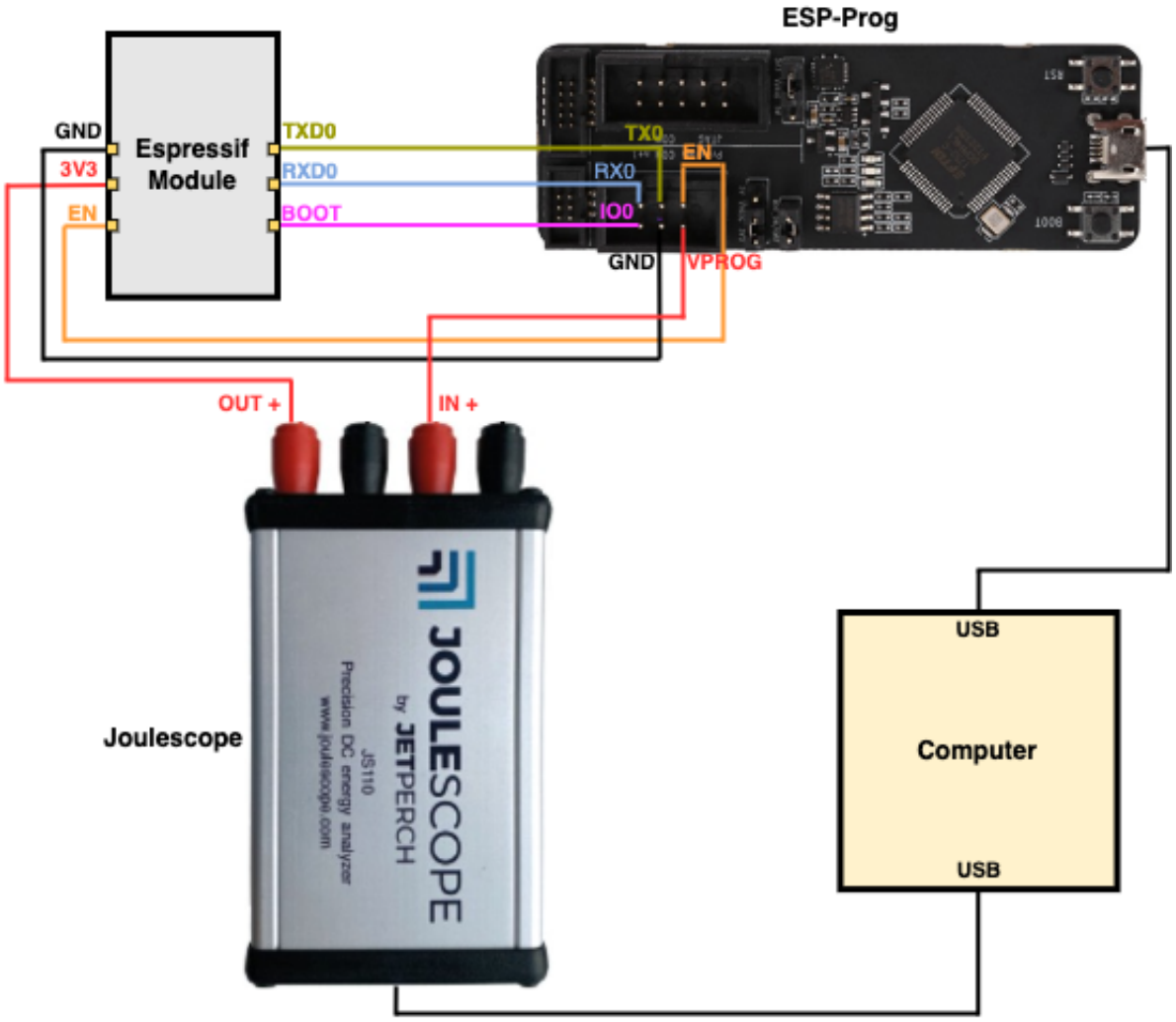


图 6: Hardware Connection (click to enlarge)

Please connect the pins of **UART TX**, **UART RX**, **SPI Boot**, **Enable**, and **Ground** on the measured module with corresponding pins on ESP-Prog, and connect the **VPROG** pin on ESP-Prog with the **IN+** port on the Joulescope ammeter and connect its **OUT+** port with the **Power supply (3V3)** pin on the measured module. For the specific names of these pins in different modules, please refer to the list below.

表 1: Pin Names of Modules Based on ESP32-P4 Chip

Function of Module Pin	Pin Name
UART TX	TXD0
UART RX	RXD0
SPI Boot	IO35
Enable	EN
Power Supply	3V3
Ground	GND

For details of the pin names, please refer to the datasheet of specific module.

4.8.3 Measurement Steps

ESP32-S3-WROOM-1 is used as an example in the measurement, and other modules can be measured similarly. For the specific current consumption of chips in different modes, please refer to the Current Consumption subsection in the corresponding chip datasheet.

You can refer to the following steps to measure the current in deep-sleep mode.

- Connect the aforementioned devices according to the hardware connection.
- Flash the deep_sleep example to the module. For details, please refer to Start a Project on Linux and macOS for a computer with Linux or macOS system or Start a Project on Windows for a computer with Windows system.
- By default, the module will be woken up every 20 seconds (you can change the timing by modifying the code of this example). To check if the example runs as expected, you can monitor the module operation by running `idf.py -p PORT monitor` (please replace PORT with your serial port name).
- Open the Joulescope software to see the current waveform as shown in the image below.

From the waveforms, you can obtain that the current of the module in deep-sleep mode is 8.14 µA. In addition, you can also see the current of the module in active mode, which is about 23.88 mA. The waveforms also show that the average power consumption during deep-sleep mode is 26.85 µW, and the average power consumption during active mode is 78.32 mW.

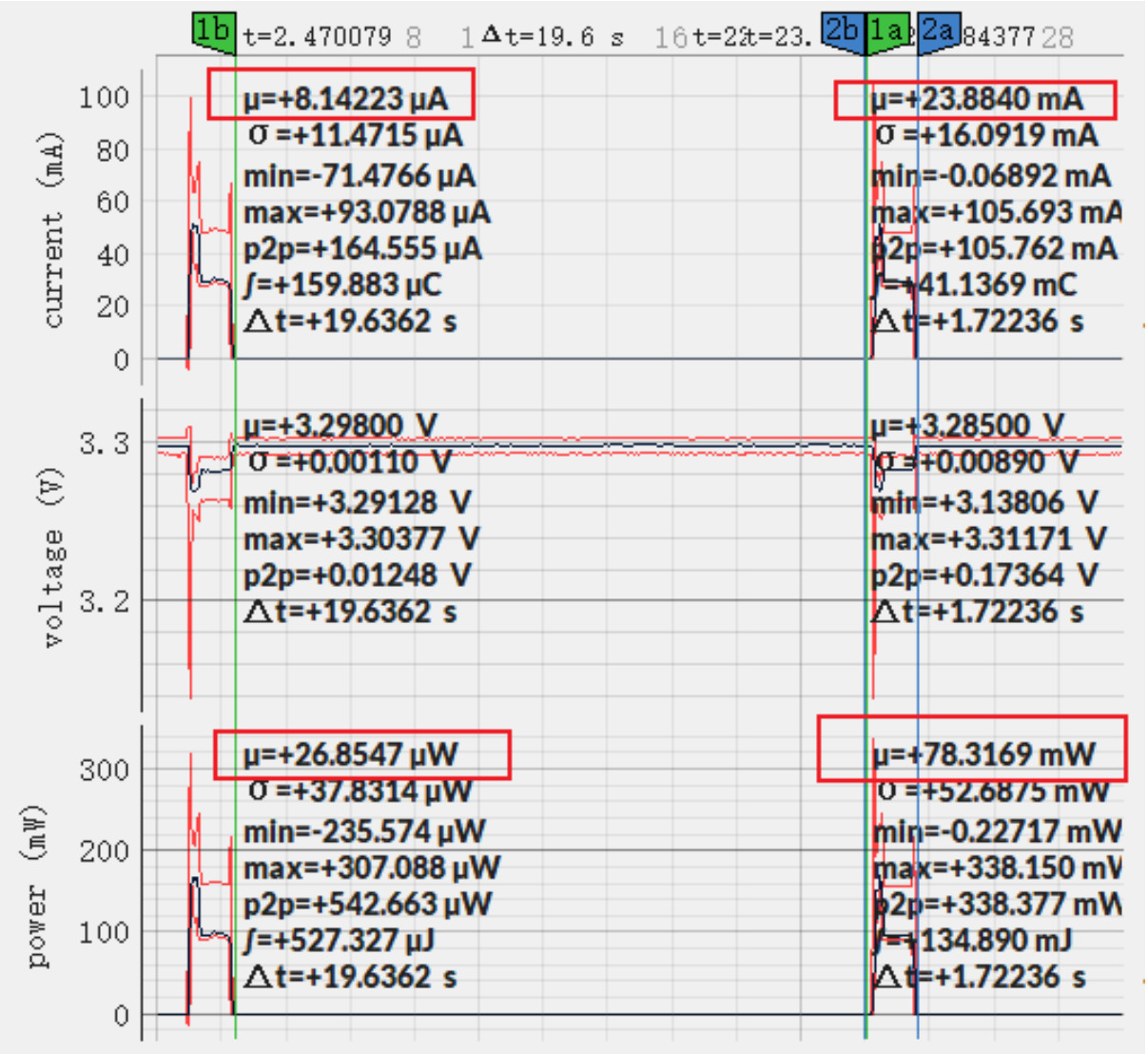


图 7: Current Waveform of ESP32-S3-WROOM-1 (click to enlarge)

The figure below shows the total power consumption of one cycle is 6.37 mW.

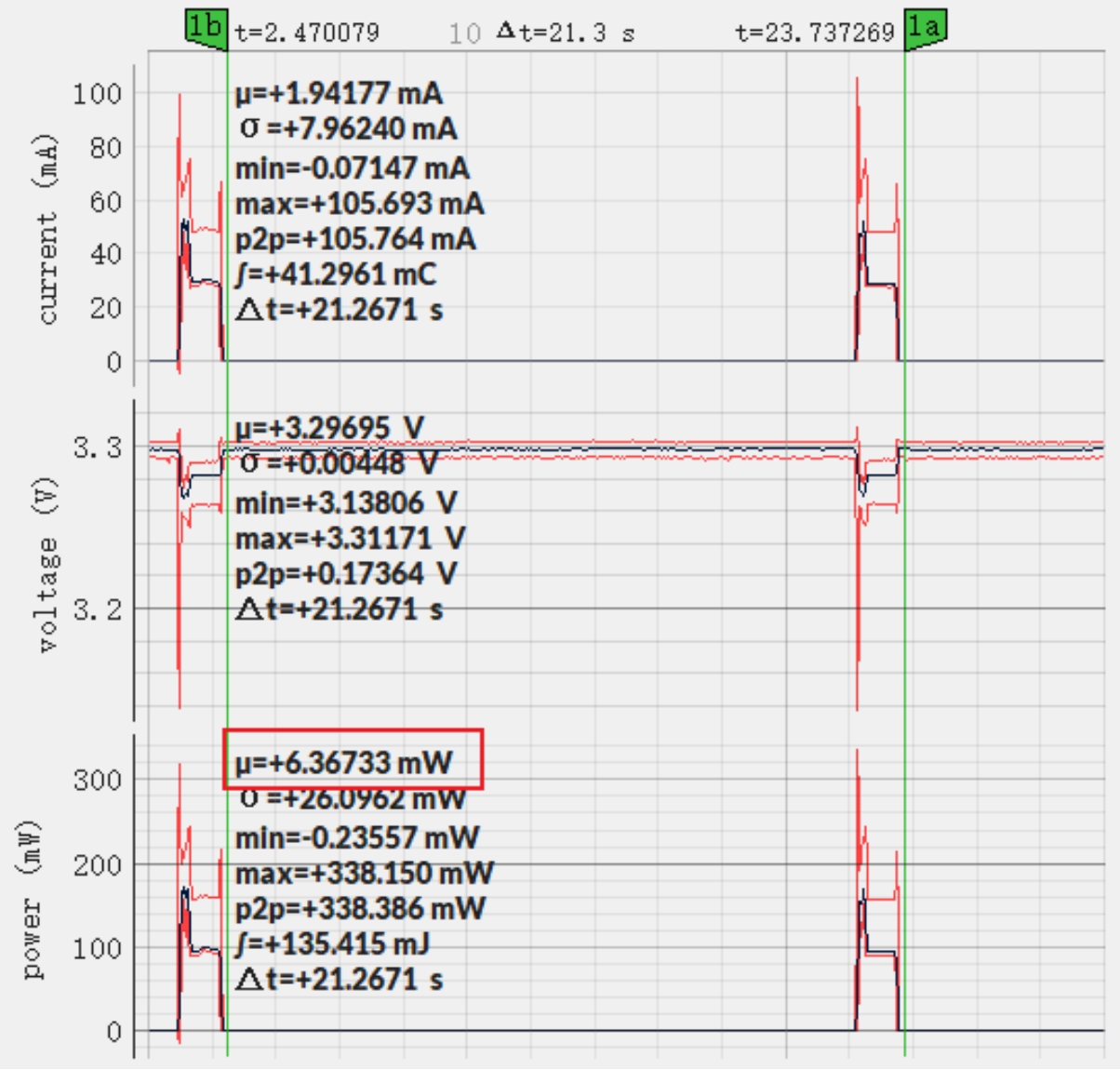


图 8: Power Consumption of ESP32-S3-WROOM-1 (click to enlarge)

By referring to these power consumption in different modes, you can estimate the power consumption of your applications and choose the appropriate power source.

4.9 错误处理

4.9.1 概述

在应用程序开发中，及时发现并处理在运行时期的错误，对于保证应用程序的健壮性非常重要。常见的运行时错误有如下几种：

- 可恢复的错误：
  - 通过函数的返回值（错误码）表示的错误
  - 使用 `throw` 关键字抛出的 C++ 异常

- 不可恢复（严重）的错误：
  - 断言失败（使用 assert 宏或者其它类似方法，可参考*Assertions*）或者直接调用 abort() 函数造成的错误
  - CPU 异常：访问受保护的内存区域、非法指令等
  - 系统级检查：看门狗超时、缓存访问错误、堆栈溢出、堆栈粉碎、堆栈损坏等

本文将介绍 ESP-IDF 中针对可恢复错误的错误处理机制，并提供一些常见错误的处理模式。

关于如何处理不可恢复的错误，请查阅不可恢复错误。

4.9.2 错误码

ESP-IDF 中大多数函数会返回*esp_err_t* 类型的错误码，*esp_err_t* 实质上是带符号的整型，ESP_OK 代表成功（没有错误），具体值定义为 0。

在 ESP-IDF 中，许多头文件都会使用预处理器，定义可能出现的错误代码。这些错误代码通常均以 ESP_ERR_ 前缀开头，一些常见错误（比如内存不足、超时、无效参数等）的错误代码则已经在 esp_err.h 文件中定义好了。此外，ESP-IDF 中的各种组件 (component) 也都可以针对具体情况，自行定义更多错误代码。

完整错误代码列表，请见错误代码参考 中查看完整的错误列表。

4.9.3 错误码到错误消息

错误代码并不直观，因此 ESP-IDF 还可以使用*esp_err_to_name()* 或者*esp_err_to_name_r()* 函数，将错误代码转换为具体的错误消息。例如，我们可以向*esp_err_to_name()* 函数传递错误代码 0x101，可以得到返回字符串 “ESP_ERR_NO_MEM”。这样一来，我们可以在日志中输出更加直观的错误消息，而不是简单的错误码，从而帮助研发人员更快理解发生了何种错误。

此外，如果出现找不到匹配的 ESP_ERR_ 值的情况，函数*esp_err_to_name_r()* 则会尝试将错误码作为一种 标准 POSIX 错误代码 进行解释。具体过程为：POSIX 错误代码（例如 ENOENT，ENOMEM）定义在 errno.h 文件中，可以通过 errno 变量获得，进而调用 strerror_r 函数实现。在 ESP-IDF 中，errno 是一个基于线程的局部变量，即每个 FreeRTOS 任务都有自己的 errno 副本，通过函数修改 errno 也只会作用于当前任务中的 errno 变量值。

该功能（即在无法匹配 ESP_ERR_ 值时，尝试用标准 POSIX 解释错误码）默认启用。用户也可以禁用该功能，从而减小应用程序的二进制文件大小，详情可见*CONFIG_ESP_ERR_TO_NAME_LOOKUP*。注意，该功能对禁用并不影响*esp_err_to_name()* 和*esp_err_to_name_r()* 函数的定义，用户仍可调用这两个函数转化错误码。在这种情况下，*esp_err_to_name()* 函数在遇到无法匹配错误码的情况会返回 UNKNOWN ERROR，而*esp_err_to_name_r()* 函数会返回 Unknown error 0xXXXX(YYYYY)，其中 0xXXXX 和 YYYYY 分别代表错误代码的十六进制和十进制表示。

4.9.4 ESP_ERROR_CHECK 宏

宏*ESP_ERROR_CHECK* 的功能和 assert 类似，不同之处在于：这个宏会检查*esp_err_t* 的值，而非判断 bool 条件。如果传给*ESP_ERROR_CHECK* 的参数不等于*ESP_OK* ，则会在控制台上打印错误消息，然后调用 abort() 函数。

错误消息通常如下所示:

```
ESP_ERROR_CHECK failed: esp_err_t 0x107 (ESP_ERR_TIMEOUT) at 0x400d1fdf

file: "/Users/user/esp/example/main/main.c" line 20
func: app_main
expression: sdmmc_card_init(host, &card)

Backtrace: 0x40086e7c:0x3ffb4ff0 0x40087328:0x3ffb5010 0x400d1fdf:0x3ffb5030␣
↪0x400d0816:0x3ffb5050
```

备注： 如果使用*IDF 监视器*，则最后一行回溯结果中的地址将会被自动解析为相应的文件名和行号。

- 第一行打印错误代码的十六进制表示，及该错误在源代码中的标识符。这个标识符取决于*CONFIG_ESP_ERR_TO_NAME_LOOKUP* 选项的设定。最后，第一行还会打印程序中该错误发生的具体位置。
- 下面几行显示了程序中调用*ESP_ERROR_CHECK* 宏的具体位置，以及传递给该宏的参数。
- 最后一行打印回溯结果。对于所有不可恢复错误，这里在应急处理程序中打印的内容都是一样的。更多有关回溯结果的详细信息，请参阅*不可恢复错误* 。

4.9.5 ESP_ERROR_CHECK_WITHOUT_ABORT 宏

宏*ESP_ERROR_CHECK_WITHOUT_ABORT* 的功能和 ESP_ERROR_CHECK 类似，不同之处在于它不会调用 abort().

4.9.6 ESP_RETURN_ON_ERROR 宏

宏*ESP_RETURN_ON_ERROR* 用于错误码检查，如果错误码不等于*ESP_OK*，该宏会打印错误信息，并使原函数立刻返回。

4.9.7 ESP_GOTO_ON_ERROR 宏

宏*ESP_GOTO_ON_ERROR* 用于错误码检查，如果错误码不等于*ESP_OK*，该宏会打印错误信息，将局部变量 *ret* 赋值为该错误码，并使原函数跳转至给定的 *goto_tag*.

4.9.8 ESP_RETURN_ON_FALSE 宏

宏*ESP_RETURN_ON_FALSE* 用于条件检查，如果给定条件不等于 *true*，该宏会打印错误信息，并使原函数立刻返回，返回值为给定的 *err_code*.

4.9.9 ESP_GOTO_ON_FALSE 宏

宏*ESP_GOTO_ON_FALSE* 用于条件检查，如果给定条件不等于 *true*，该宏会打印错误信息，将局部变量 *ret* 赋值为给定的 *err_code*，并使原函数跳转至给定的 *goto_tag*.

4.9.10 CHECK 宏使用示例

示例：

```
static const char* TAG = "Test";

esp_err_t test_func(void)
{
    esp_err_t ret = ESP_OK;

    ESP_ERROR_CHECK(x);                                                 // err message␣
↪printed if `x` is not `ESP_OK`, and then `abort()`.
    ESP_ERROR_CHECK_WITHOUT_ABORT(x);                                   // err message␣
↪printed if `x` is not `ESP_OK`, without `abort()`.
    ESP_RETURN_ON_ERROR(x, TAG, "fail reason 1");                       // err message␣
↪printed if `x` is not `ESP_OK`, and then function returns with code `x`.
    ESP_GOTO_ON_ERROR(x, err, TAG, "fail reason 2");                    // err message␣
↪printed if `x` is not `ESP_OK`, `ret` is set to `x`, and then jumps to `err`.
```

(下页继续)

(续上页)

```
    ESP_RETURN_ON_FALSE(a, err_code, TAG, "fail reason 3");      // err message␣
↪printed if `a` is not `true`, and then function returns with code `err_code`.
    ESP_GOTO_ON_FALSE(a, err_code, err, TAG, "fail reason 4");  // err message␣
↪printed if `a` is not `true`, `ret` is set to `err_code`, and then jumps to␣
↪`err`.

err:
    // clean up
    return ret;
}
```

备注： 如果 Kconfig 中的*CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT* 选项被打开, CHECK 宏将不会打印错误信息，其他功能不变。

ESP_RETURN_xx 和 ESP_GOTO_xx 宏不可以在中断服务程序里被调用。如需要在中断中使用类似功能，请使用 xx_ISR 宏，如 ESP_RETURN_ON_ERROR_ISR 等。

4.9.11 错误处理模式

1. 尝试恢复。根据具体情况不同，我们具体可以：
   - 在一段时间后，重新调用该函数；
   - 尝试删除该驱动，然后重新进行 “初始化”；
   - 采用其他带外机制，修改导致错误发生的条件（例如，对一直没有响应的外设进行复位等）。

   示例:

```
esp_err_t err;
do {
    err = sdio_slave_send_queue(addr, len, arg, timeout);
    // 如果发送队列已满就不断重试
} while (err == ESP_ERR_TIMEOUT);
if (err != ESP_OK) {
    // 处理其他错误
}
```

2. 将错误传递回调用程序。在某些中间件组件中，采用此类处理模式代表函数必须以相同的错误码退出，这样才能确保所有分配的资源都能得到释放。
   示例:

```
sdmmc_card_t* card = calloc(1, sizeof(sdmmc_card_t));
if (card == NULL) {
    return ESP_ERR_NO_MEM;
}
esp_err_t err = sdmmc_card_init(host, &card);
if (err != ESP_OK) {
    // 释放内存
    free(card);
    // 将错误码传递给上层（例如通知用户）
    // 或者，应用程序可以自定义错误代码并返回
    return err;
}
```

3. 转为不可恢复错误，比如使用 ESP_ERROR_CHECK。详情请见*ESP_ERROR_CHECK 宏* 章节。
   对于中间件组件而言，通常并不希望在发生错误时中止应用程序。不过，有时在应用程序级别，这种做法是可以接受的。
   在 ESP-IDF 的示例代码中，很多都会使用 ESP_ERROR_CHECK 来处理各种 API 引发的错误，虽然这不是应用程序的最佳做法，但可以让示例代码看起来更加简洁。
   示例:

```
ESP_ERROR_CHECK(spi_bus_initialize(host, bus_config, dma_chan));
```

4.9.12 C++ 异常

请参考异常处理。

4.10 片外 RAM

4.10.1 简介

ESP32-P4 提供了好几百 KB 的片上 RAM，可以满足大部分需求。但有些场景可能需要更多 RAM，因此 ESP32-P4 另外提供了高达 Value not updated 的虚拟地址，供片外 PSRAM（伪静态随机存储器）存储器使用。片外 RAM 已经集成到内存映射中，在某些范围内与片上 RAM 使用方式相同。

4.10.2 硬件

ESP32-P4 支持与 SPI flash 芯片并联的 PSRAM。虽然 ESP32-P4 支持多种类型的 RAM 芯片，但 ESP-IDF 当前仅支持乐鑫品牌的 PSRAM 芯片，如 ESP-PSRAM32、ESP-PSRAM64 等。

备注： 请查询相应 PSRAM 芯片以及 ESP32-P4 的技术规格书获取准确的工作电压。

PSRAM 默认由片上 LDO2 供电。可设置*CONFIG_ESP_LDO_CHAN_PSRAM_DOMAIN* 来切换相应的 LDO 输出通道，将该值设为 -1 表示使用外部电源，即不使用片上 LDO。默认情况下，连接到 LDO 的 PSRAM 会基于所使用的乐鑫模组设置正确电压。如果未使用乐鑫模组，仍可设置*CONFIG_ESP_LDO_VOLTAGE_PSRAM_DOMAIN* 来选择 LDO 输出电压。使用外部电源时，该选项不存在。

备注： 乐鑫同时提供模组和系统级封装芯片，集成了兼容的 PSRAM 和 flash，可直接用于终端产品 PCB 中。如需了解更多信息，请前往乐鑫官网。注意，ESP-IDF SDK 可能与定制的 PSRAM 芯片不兼容。

有关将 SoC 或模组管脚连接到片外 PSRAM 芯片的具体细节，请查阅 SoC 或模组技术规格书。

4.10.3 配置片外 RAM

ESP-IDF 完全支持将片外 RAM 集成到你的应用程序中。在启动并完成片外 RAM 初始化后，可以将 ESP-IDF 配置为用多种方式处理片外 RAM：

- *集成片外 RAM 到 ESP32-P4 内存映射*
- *添加片外 RAM 到堆内存分配器*
- *调用 malloc() 分配片外 RAM* (default)
- *允许.bss 段放入片外存储器*
- *允许.noinit 段放入片外存储器*
- *在 PSRAM 中直接执行代码*

集成片外 RAM 到 ESP32-P4 内存映射

在*CONFIG_SPIRAM_USE* 中选择 `Integrate RAM into memory map` 选项，以集成片外 RAM 到 ESP32-P4 内存映射。

这是集成片外 RAM 最基础的设置选项，大多数用户需要用到其他更高级的选项。

ESP-IDF 启动过程中，片外 RAM 被映射到数据虚拟地址空间，该地址空间是动态分配的，其长度为 PSRAM 大小和可用数据虚拟地址空间大小之间的最小值。

应用程序可以创建指向该区域的指针，手动将数据放入片外存储器，并全权负责管理片外 RAM，包括协调缓存占用、防止发生损坏等。

建议通过 ESP-IDF 堆内存分配器访问 PSRAM（见下一小节）。

添加片外 RAM 到堆内存分配器

在*CONFIG_SPIRAM_USE* 中选择 `Make RAM allocatable using heap_caps_malloc(..., MALLOC_CAP_SPIRAM)` 选项。

启用上述选项后，片外 RAM 被映射到数据虚拟地址空间，并将这个区域添加到携带 `MALLOC_CAP_SPIRAM` 标志的堆内存分配器。

程序如果想从片外存储器分配存储空间，则需要调用 `heap_caps_malloc(size, MALLOC_CAP_SPIRAM)`，之后可以调用 `free()` 函数释放这部分存储空间。

调用 `malloc()` 分配片外 RAM

在*CONFIG_SPIRAM_USE* 中选择 `Make RAM allocatable using malloc() as well` 选项，该选项为默认选项。

启用此选项后，片外存储器将被添加到内存分配程序（与上一选项相同），同时也将被添加到由标准 `malloc()` 函数返回的 RAM 中。

应用程序因此可以使用片外 RAM，无需重写代码就能使用 `heap_caps_malloc(..., MALLOC_CAP_SPIRAM)`。

如果某次内存分配偏向于片外存储器，也可以使用*CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL* 设置分配空间的大小阈值，控制分配结果：

- 如果分配的空间小于或等于阈值，分配程序将首先选择内部存储器。
- 如果分配的空间大于阈值，分配程序将首先选择外部存储器。

如果优先考虑的内部或外部存储器中没有可用的存储块，分配程序则会选择其他类型存储。

由于有些内存缓冲器仅可在内部存储器中分配，因此需要使用第二个配置项*CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL* 定义一个内部内存池，仅限显式的内部存储器分配使用（例如用于 DMA 的存储器）。常规 `malloc()` 将不会从该池中分配，但可以使用*MALLOC_CAP_DMA* 和 `MALLOC_CAP_INTERNAL` 标志从该池中分配存储器。

允许.bss 段放入片外存储器

通过勾选*CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY* 启用该选项。

启用该选项后，PSRAM 被映射到的数据虚拟地址空间将用于存储来自 lwip、net80211、libpp、wpa_supplicant 和 bluedroid ESP-IDF 库中零初始化的数据（BSS 段）。

通过将宏 `EXT_RAM_BSS_ATTR` 应用于任何静态声明（未初始化为非零值），可以将附加数据从内部 BSS 段移到片外 RAM。

也可以使用链接器片段方案 `extram_bss` 将组件或库的 BSS 段放到片外 RAM 中。

启用此选项可以减少 BSS 段占用的内部静态存储。

剩余的片外 RAM 也可以通过上述方法添加到堆分配器中。

允许.noinit 段放入片外存储器

通过勾选*CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY* 启用该选项。启用该选项后，PSRAM 被映射到的数据虚拟地址空间将用于存储未初始化的数据。即使在启动或重新启动期间，放置在该段中的值也不会被初始化或修改。

通过应用 `EXT_RAM_NOINIT_ATTR` 宏，可以将数据从内部 NOINIT 段移到片外 RAM。剩余的片外 RAM 也可以通过上述方法添加到堆分配器中，具体请参考*添加片外 RAM 到堆内存分配器*。

在 PSRAM 中直接执行代码 启用*CONFIG_SPIRAM_XIP_FROM_PSRAM* 选项后能在 PSRAM 中直接执行代码。通常放置在 flash 中的段，如 `.text` 部分的数据（用于指令）和 `.rodata` 部分的数据（用于只读数据），将被加载到 PSRAM 中。

启用此选项后，SPI1 flash 操作期间 cache 保持启用状态，因此需要执行的代码在此期间不必放置在内部 RAM 中。由于 ESP32-P4 flash 和 PSRAM 使用两个独立的 SPI 总线，将 flash 内容移动到 PSRAM 实际上增加了 PSRAM MSPI 总线的负载，因此整体性能将取决于应用程序对 PSRAM 的使用。例如，因为 PSRAM 的总线速度可能远大于 flash 总线速度，如果那些之前存放在 flash 中的指令和只读数据（现在存放于 PSRAM 中）不被经常访问，那么整体性能将有可能更好。因此，建议先进行性能分析以确定启用此选项是否会显著影响应用程序性能。

4.10.4 片外 RAM 使用限制

使用片外 RAM 有下面一些限制：

- flash cache 禁用时（比如，正在写入 flash），片外 RAM 将无法访问；同样，对片外 RAM 的读写操作也将导致 cache 访问异常。因此，ESP-IDF 不会在片外 RAM 中分配任务堆栈（详见下文）。
- 片外 RAM 与片外 flash 使用相同的 cache 区域，这意味着频繁在片外 RAM 访问的变量可以像在片上 RAM 中一样快速读取和修改。但访问大块数据时（大于 32 KB），cache 空间可能会不足，访问速度将降低到片外 RAM 的访问速度。此外，访问大块数据会挤出 flash cache，可能在之后降低代码的执行速度。
- 一般来说，片外 RAM 不会用作任务堆栈存储器。*`xTaskCreate()`* 及类似函数始终会为堆栈和任务 TCB 分配片上储存器。

可以使用*CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY* 选项将任务堆栈放入片外存储器。这时，必须使用*`xTaskCreateStatic()`* 指定从片外存储器分配的任务堆栈缓冲区，否则任务堆栈将仍从片上存储器分配。

4.10.5 初始化失败

默认情况下，片外 RAM 初始化失败将终止 ESP-IDF 启动。如果想禁用此功能，可启用*CONFIG_SPIRAM_IGNORE_NOTFOUND* 配置选项。

4.10.6 加密

可以为存储在外部 RAM 中的数据启用自动加密功能。启用该功能后，通过缓存读写的任何数据将被外部存储器加密硬件自动加密、解密。

只要启用了 flash 加密功能，就会启用这个功能。关于如何启用 flash 加密以及其工作原理，请参考*flash 加密*。

4.11 严重错误

4.11.1 概述

在某些情况下，程序并不会按照我们的预期运行，在 ESP-IDF 中，这些情况包括：

- CPU 异常：非法指令，加载/存储时的内存对齐错误，加载/存储时的访问权限错误。
- 系统级检查错误：
  - 中断看门狗 超时
  - 任务看门狗 超时（只有开启*CONFIG_ESP_TASK_WDT_PANIC* 后才会触发严重错误）
  - 高速缓存访问错误
  - 掉电检测事件
  - 堆栈溢出
  - 堆栈粉碎保护检查
  - 堆完整性检查
  - 未定义行为清理器 (UBSAN) 检查
- 使用 `assert`、`configASSERT` 等类似的宏断言失败。

本指南会介绍 ESP-IDF 中这类错误的处理流程，并给出对应的解决建议。

4.11.2 紧急处理程序

概述 中列举的所有错误都会由 紧急处理程序 *(Panic Handler)* 负责处理。

紧急处理程序首先会将出错原因打印到控制台，例如 CPU 异常的错误信息通常会类似于

```
Guru Meditation Error: Core 0 panic'ed (IllegalInstruction). Exception was␣
↪unhandled.
```

对于一些系统级检查错误（如中断看门狗超时，高速缓存访问错误等），错误信息会类似于

```
Guru Meditation Error: Core 0 panic'ed (Cache error). Exception was␣
↪unhandled.
```

不管哪种情况，错误原因都会被打印在括号中。请参阅*Guru Meditation 错误* 以查看所有可能的出错原因。

紧急处理程序接下来的行为将取决于*CONFIG_ESP_SYSTEM_PANIC* 的设置，支持的选项包括：

- 打印 CPU 寄存器，然后重启（`CONFIG_ESP_SYSTEM_PANIC_PRINT_REBOOT`）- 默认选项
  打印系统发生异常时 CPU 寄存器的值，打印回溯，最后重启芯片。
- 打印 CPU 寄存器，然后暂停（`CONFIG_ESP_SYSTEM_PANIC_PRINT_HALT`）
  与上一个选项类似，但不会重启，而是选择暂停程序的运行。重启程序需要外部执行复位操作。
- 静默重启（`CONFIG_ESP_SYSTEM_PANIC_SILENT_REBOOT`）
  不打印 CPU 寄存器的值，也不打印回溯，立即重启芯片。
- 调用 GDB Stub（`CONFIG_ESP_SYSTEM_PANIC_GDBSTUB`）
  启动 GDB 服务器，通过控制台 UART 接口与 GDB 进行通信。该选项只提供只读调试或者事后调试，详细信息请参阅*GDB Stub*。

备注： 仅当构建中包含组件 `esp_gdbstub` 时，配置选项*CONFIG_ESP_SYSTEM_PANIC* 中的 `CONFIG_ESP_SYSTEM_PANIC_GDBSTUB` 选项可用。

紧急处理程序的行为还受到另外两个配置项的影响：

- 如果使能了*CONFIG_ESP_DEBUG_OCDAWARE*（默认），紧急处理程序会检测 ESP32-P4 是否已经连接 JTAG 调试器。如果检测成功，程序会暂停运行，并将控制权交给调试器。在这种情况下，寄存器和回溯不会被打印到控制台，并且也不会使用 GDB Stub 和 Core Dump 的功能。

- 如果使能了内核转储 功能，系统状态（任务堆栈和寄存器）会被转储到 flash 或者 UART 以供后续分析。
- 如果*CONFIG_ESP_PANIC_HANDLER_IRAM* 被禁用（默认情况下禁用），紧急处理程序的代码会放置在 flash 而不是 IRAM 中。这意味着，如果 ESP-IDF 在 flash 高速缓存禁用时崩溃，在运行 GDB Stub 和内核转储之前紧急处理程序会自动重新使能 flash 高速缓存。如果 flash 高速缓存也崩溃了，这样做会增加一些小风险。
  如果使能了该选项，紧急处理程序的代码（包括所需的 UART 函数）会放置在 IRAM 中，导致 SRAM 中的可用内存空间变小。当禁用 flash 高速缓存（如写入 SPI flash）时或触发异常导致 flash 高速缓存崩溃时，可用此选项调试一些复杂的崩溃问题。
- 如果启用*CONFIG_ESP_SYSTEM_PANIC_REBOOT_DELAY_SECONDS* （默认为禁用）并将其配置为大于 0 的数字，紧急处理程序将基于该数字延迟重启的时间，单位为秒。如果用于监测串行输出的工具不支持停止和检查串行输出，可启用该选项。在这种情况下，借助延迟重启，用户可以在延迟期间检查和调试紧急处理程序的输出（例如回溯）。延迟结束后，设备将重新启动，并记录重置原因。

下图展示了紧急处理程序的行为：

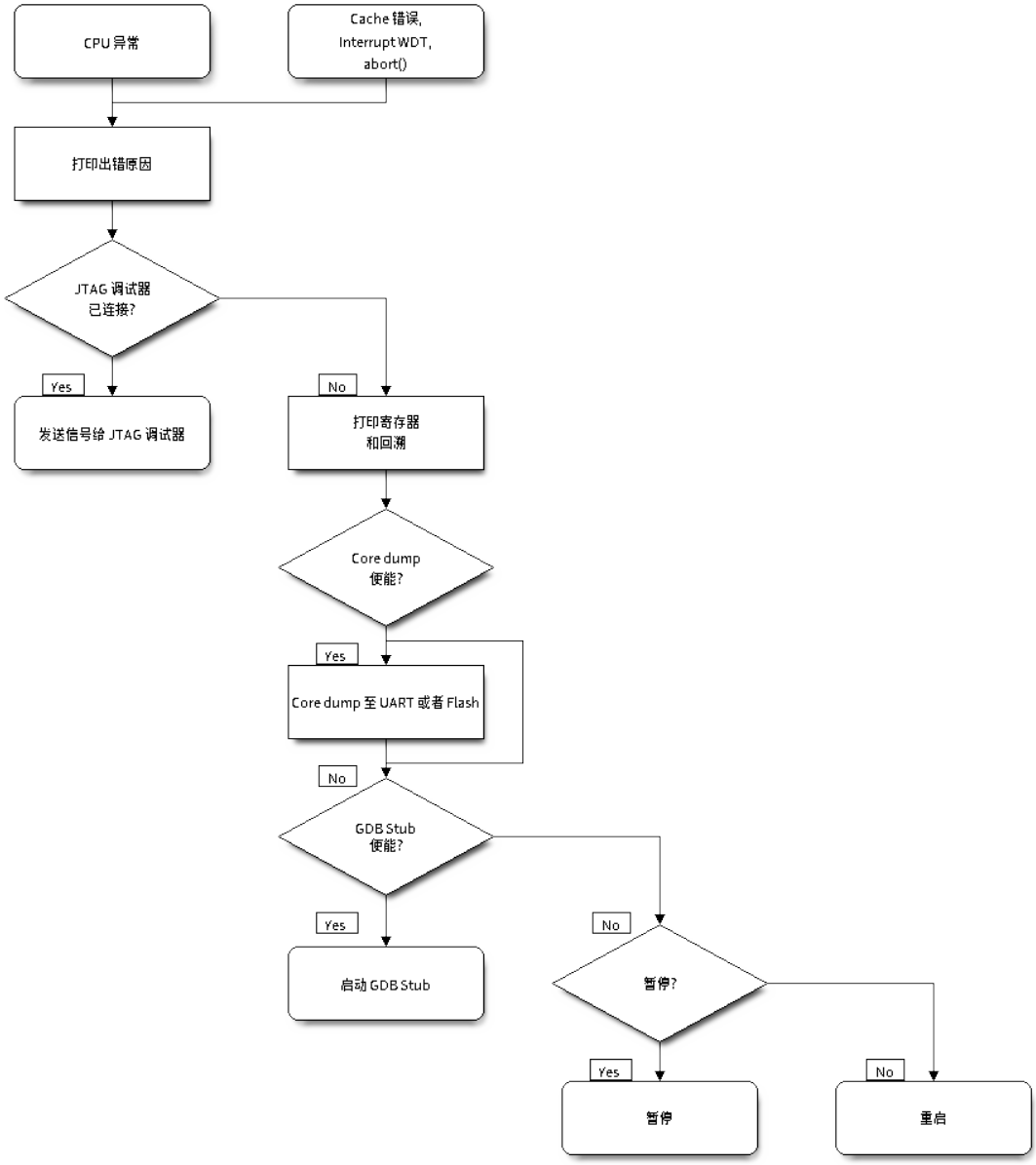


图 9: 紧急处理程序流程图（点击放大）

4.11.3 寄存器转储与回溯

除非启用了 CONFIG_ESP_SYSTEM_PANIC_SILENT_REBOOT 否则紧急处理程序会将 CPU 寄存器和回溯打印到控制台

```
Core  0 register dump:
MEPC    : 0x420048b4  RA      : 0x420048b4  SP      : 0x3fc8f2f0  GP      :␣
↪0x3fc8a600
TP      : 0x3fc8a2ac  T0      : 0x40057fa6  T1      : 0x0000000f  T2      :␣
↪0x00000000
S0/FP   : 0x00000000  S1      : 0x00000000  A0      : 0x00000001  A1      :␣
↪0x00000001
A2      : 0x00000064  A3      : 0x00000004  A4      : 0x00000001  A5      :␣
↪0x00000000
A6      : 0x42001fd6  A7      : 0x00000000  S2      : 0x00000000  S3      :␣
↪0x00000000
S4      : 0x00000000  S5      : 0x00000000  S6      : 0x00000000  S7      :␣
↪0x00000000
S8      : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     :␣
↪0x00000000
T3      : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      :␣
↪0x00000000
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000007  MTVAL   :␣
↪0x00000000
MHARTID : 0x00000000
```

仅会打印异常帧中 CPU 寄存器的值，即引发 CPU 异常或者其它严重错误时刻的值。

紧急处理程序如果是因 abort() 而调用，则不会打印寄存器转储。

如果使用了*IDF 监视器*，该工具会将程序计数器的值转换为对应的代码位置（函数名，文件名，行号），并加以注释：

```
Core  0 register dump:
MEPC    : 0x420048b4  RA      : 0x420048b4  SP      : 0x3fc8f2f0  GP      :␣
↪0x3fc8a600
0x420048b4: app_main at /Users/user/esp/example/main/hello_world_main.c:20

0x420048b4: app_main at /Users/user/esp/example/main/hello_world_main.c:20

TP      : 0x3fc8a2ac  T0      : 0x40057fa6  T1      : 0x0000000f  T2      :␣
↪0x00000000
S0/FP   : 0x00000000  S1      : 0x00000000  A0      : 0x00000001  A1      :␣
↪0x00000001
A2      : 0x00000064  A3      : 0x00000004  A4      : 0x00000001  A5      :␣
↪0x00000000
A6      : 0x42001fd6  A7      : 0x00000000  S2      : 0x00000000  S3      :␣
↪0x00000000
0x42001fd6: uart_write at /Users/user/esp/esp-idf/components/vfs/vfs_uart.c:201

S4      : 0x00000000  S5      : 0x00000000  S6      : 0x00000000  S7      :␣
↪0x00000000
S8      : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     :␣
↪0x00000000
T3      : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      :␣
↪0x00000000
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000007  MTVAL   :␣
↪0x00000000
MHARTID : 0x00000000
```

此外，由于紧急处理程序中提供了堆栈转储，因此*IDF 监视器* 也可以生成并打印回溯。输出结果如下：

```
Backtrace:

0x42006686 in bar (ptr=ptr@entry=0x0) at ../main/hello_world_main.c:18
18      *ptr = 0x42424242;
#0  0x42006686 in bar (ptr=ptr@entry=0x0) at ../main/hello_world_main.c:18
#1  0x42006692 in foo () at ../main/hello_world_main.c:22
#2  0x420066ac in app_main () at ../main/hello_world_main.c:28
#3  0x42015ece in main_task (args=<optimized out>) at /Users/user/esp/components/
↪freertos/port/port_common.c:142
#4  0x403859b8 in vPortEnterCritical () at /Users/user/esp/components/freertos/
↪port/riscv/port.c:130
#5  0x00000000 in ?? ()
Backtrace stopped: frame did not save the PC
```

虽然以上的回溯信息非常方便，但要求用户使用*IDF 监视器*。因此，如果用户希望使用其它的串口监控软件也能显示堆栈回溯信息，则需要在 menuconfig 中启用*CONFIG_ESP_SYSTEM_USE_EH_FRAME* 选项。

该选项会让编译器为项目的每个函数生成 DWARF 信息。然后，当 CPU 异常发生时，紧急处理程序将解析这些数据并生成出错任务的堆栈回溯信息。输出结果如下：

```
Backtrace: 0x42009e9a:0x3fc92120 0x42009ea6:0x3fc92120 0x42009ec2:0x3fc92130␣
↪0x42024620:0x3fc92150 0x40387d7c:0x3fc92160 0xfffffffe:0x3fc92170
```

这些 `PC:SP` 对代表当前任务每一个栈帧的程序计数器值 (Program Counter) 和栈顶地址 (Stack Pointer)。

CONFIG_ESP_SYSTEM_USE_EH_FRAME 选项的主要优点是，回溯信息可以由程序自己解析生成并打印（而不依靠*IDF 监视器*）。但是该选项会导致编译后的二进制文件更大（增幅可达 20% 甚至 100%）。此外，该选项会将调试信息也保存在二进制文件里。因此，强烈不建议用户在量产/生产版本中启用该选项。

若要查找发生严重错误的代码位置，请查看"Backtrace" 的后面几行，发生严重错误的代码显示在顶行，后续几行显示的是调用堆栈。

4.11.4 GDB Stub

如果启用了 `CONFIG_ESP_SYSTEM_PANIC_GDBSTUB` 选项，在发生严重错误时，紧急处理程序不会复位芯片，相反，它将启动 GDB 远程协议服务器，通常称为 GDB Stub。发生这种情况时，可以让主机上运行的 GDB 实例通过 UART 端口连接到 ESP32。

如果使用了*IDF 监视器*，该工具会在 UART 端口检测到 GDB Stub 提示符后自动启动 GDB，输出会类似于：

```
Entering gdb stub now.
$T0b#e6GNU gdb (crosstool-NG crosstool-ng-1.22.0-80-gff1f415) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-build_apple-darwin16.3.0 --
↪target=riscv32-esp-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /Users/user/esp/example/build/example.elf...done.
Remote debugging using /dev/cu.usbserial-31301
0x400e1b41 in app_main ()
    at /Users/user/esp/example/main/main.cpp:36
```

（下页继续）

(续上页)

```
36      *((int*) 0) = 0;
(gdb)
```

在 GDB 会话中，我们可以检查 CPU 寄存器，本地和静态变量以及内存中任意位置的值。但是不支持设置断点，改变 PC 值或者恢复程序的运行。若要复位程序，请退出 GDB 会话，在 IDF 监视器中连续输入 Ctrl-T Ctrl-R，或者按下开发板上的复位按键也可以重新运行程序。

4.11.5 RTC 看门狗超时

RTC 看门狗在启动代码中用于跟踪执行时间，也有助于防止由于电源不稳定引起的锁定。RTC 看门狗默认启用，参见*CONFIG_BOOTLOADER_WDT_ENABLE*。如果执行时间超时，RTC 看门狗将自动重启系统。此时，ROM 引导加载程序将打印消息 `RTC Watchdog Timeout` 说明重启原因。

```
rst:0x10 (LP_WDT_SYS)
```

RTC 看门狗涵盖了从一级引导程序（ROM 引导程序）到应用程序启动的执行时间，最初在 ROM 引导程序中设置，而后在引导程序中使用*CONFIG_BOOTLOADER_WDT_TIME_MS* 选项进行配置（默认 9000 ms）。在应用初始化阶段，由于慢速时钟源可能已更改，RTC 看门狗将被重新配置，最后在调用 `app_main()` 之前被禁用。可以使用选项*CONFIG_BOOTLOADER_WDT_DISABLE_IN_USER_CODE* 以保证 RTC 看门狗在调用 `app_main` 之前不被禁用，而是保持运行状态，用户需要在应用代码中定期“喂狗”。

4.11.6 Guru Meditation 错误

本节将对打印在 `Guru Meditation Error: Core panic'ed` 后面括号中的致错原因进行逐一解释。

备注： 想要了解”Guru Meditation” 的历史渊源，请参阅 维基百科 。

IllegalInstruction

此 CPU 异常表示当前执行的指令不是有效指令，引起此错误的常见原因包括：

- FreeRTOS 中的任务函数已返回。在 FreeRTOS 中，如果想终止任务函数，需要调用 *`vTaskDelete()`* 函数释放当前任务的资源，而不是直接返回。
- 无法从 SPI flash 中读取下一条指令，这通常发生在：
  - 应用程序将 SPI flash 的管脚重新配置为其它功能（如 GPIO、UART 等）。有关 SPI flash 管脚的详细信息，请参阅硬件设计指南和芯片/模组的数据手册。
  - 某些外部设备意外连接到 SPI flash 的管脚上，干扰了 ESP32-P4 和 SPI flash 之间的通信。
- 在 C++ 代码中，退出 non-void 函数而无返回值被认为是未定义的行为。启用优化后，编译器通常会忽略此类函数的结尾，导致 IllegalInstruction 异常。默认情况下，ESP-IDF 构建系统启用 `-Werror=return-type`，这意味着缺少返回语句会被视为编译时错误。但是，如果应用程序项目禁用了编译器警告，可能就无法检测到该问题，在运行时就会出现 IllegalInstruction 异常。

Instruction address misaligned

此 CPU 异常表示要执行的指令地址非 2 字节对齐。

Instruction access fault, Load access fault, Store access fault

当应用程序尝试读取或写入无效的内存位置时，会发生此类 CPU 异常。此类无效内存地址可以在寄存器转储的 `MTVAL` 中找到。如果该地址为零，通常意味着应用程序正尝试解引用一个 NULL 指针。如果该地址接近于零，则通常意味着应用程序尝试访问某个结构体的成员，但是该结构体的指针为 NULL。如果

该地址是其它非法值（不在 0x3fxxxxxx - 0x6xxxxxxx 的范围内），则可能意味着用于访问数据的指针未初始化或者已经损坏。

Breakpoint

执行 EBREAK 指令时，会发生此 CPU 异常。请参见*FreeRTOS 任务堆栈末尾监视点*。

Load address misaligned, Store address misaligned

应用程序尝试读取/写入的内存位置不符合加载/存储指令对字节对齐大小的要求，例如，32 位加载指令只能访问 4 字节对齐的内存地址，而 16 位加载指令只能访问 2 字节对齐的内存地址。

Interrupt wdt timeout on CPU0 / CPU1

这表示发生了中断看门狗超时，详细信息请查阅*看门狗* 文档。

Cache error

在某些情况下，ESP-IDF 会暂时禁止通过高速缓存访问外部 SPI flash 和 SPI RAM，例如在使用 spi_flash API 读取/写入/擦除/映射 SPI flash 的时候。在这些情况下，任务会被挂起，并且未使用 ESP_INTR_FLAG_IRAM 注册的中断处理程序会被禁用。请确保任何使用此标志注册的中断处理程序所访问的代码和数据分别位于 IRAM 和 DRAM 中。更多详细信息请参阅*SPI flash API 文档*。

4.11.7 其他严重错误

掉电

ESP32-P4 内部集成掉电检测电路，并且会默认启用。如果电源电压低于安全值，掉电检测器可以触发系统复位。掉电检测器可以使用*CONFIG_ESP_BROWNOUT_DET* 和*CONFIG_ESP_BROWNOUT_DET_LVL_SEL* 这两个选项进行设置。

当掉电检测器被触发时，会打印如下信息:

```
Brownout detector was triggered
```

芯片会在该打印信息结束后复位。

请注意，如果电源电压快速下降，则只能在控制台上看到部分打印信息。

堆不完整

ESP-IDF 堆的实现包含许多运行时的堆结构检查，可以在 menuconfig 中开启额外的检查（“Heap Poisoning”）。如果其中的某项检查失败，则会打印类似如下信息:

```
CORRUPT HEAP: Bad tail at 0x3ffe270a. Expected 0xbaad5678 got 0xbaac5678
assertion "head != NULL" failed: file "/Users/user/esp/esp-idf/components/heap/
↪multi_heap_poisoning.c", line 201, function: multi_heap_free
abort() was called at PC 0x400dca43 on core 0
```

更多详细信息，请查阅*堆内存调试* 文档。

堆栈溢出

硬件堆栈保护 ESP32-P4 集成了辅助调试模块，支持监测堆栈指针 (SP) 寄存器，确保其值位于已分配给堆栈的内存范围内。发生中断处理或 FreeRTOS 切换上下文时，辅助调试模块都会设置新的堆栈监测范围。注意，该操作会对性能产生一定影响。

以下为辅助调试模块的部分相关特性：

- 采用硬件实现
- 支持监测堆栈指针寄存器的值
- 无需占用额外 CPU 时间或内存，即可监测堆栈内存范围

当辅助调试模块检测到堆栈溢出时，将触发紧急处理程序并打印类似如下信息：

```
Guru Meditation Error: Core 0 panic'ed (Stack protection fault).
```

可以通过*CONFIG_ESP_SYSTEM_HW_STACK_GUARD* 选项，禁用硬件堆栈保护。

FreeRTOS 任务堆栈末尾监视点 ESP-IDF 支持基于监视点的 FreeRTOS 堆栈溢出检测机制。每次 FreeRTOS 切换任务上下文时，都会设置一个监视点，用于监视堆栈的最后 32 字节。

通常，该设置会提前触发监视点，触发点可能会比预期提前多达 28 字节。基于 FreeRTOS 中堆栈金丝雀的大小为 20 字节，故将观察范围设置为 32 字节，确保可以在堆栈金丝雀遭到破坏前及时触发监测点。

备注： 并非每次堆栈溢出都能触发监视点。如果任务绕过堆栈金丝雀的位置访问堆栈，则无法触发监视点。

监视点触发后，将打印类似如下信息：

```
Guru Meditation Error: Core  0 panic'ed (Breakpoint). Exception was unhandled.
```

可以通过*CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK* 选项启用该功能。

FreeRTOS 堆栈检查 请参见*CONFIG_FREERTOS_CHECK_STACKOVERFLOW*。

堆栈粉碎

堆栈粉碎保护（基于 GCC `-fstack-protector*` 标志）可以通过 ESP-IDF 中的*CONFIG_COMPILER_STACK_CHECK_MODE* 选项来开启。如果检测到堆栈粉碎，则会打印类似如下的信息：

```
Stack smashing protect failure!

abort() was called at PC 0x400d2138 on core 0

Backtrace: 0x4008e6c0:0x3ffc1780 0x4008e8b7:0x3ffc17a0 0x400d2138:0x3ffc17c0␣
↪0x400e79d5:0x3ffc17e0 0x400e79a7:0x3ffc1840 0x400e79df:0x3ffc18a0␣
↪0x400e2235:0x3ffc18c0 0x400e1916:0x3ffc18f0 0x400e19cd:0x3ffc1910␣
↪0x400e1a11:0x3ffc1930 0x400e1bb2:0x3ffc1950 0x400d2c44:0x3ffc1a80
0
```

回溯信息会指明发生堆栈粉碎的函数，建议检查函数中是否有代码访问局部数组时发生了越界。

未定义行为清理器 (UBSAN) 检查

未定义行为清理器 (UBSAN) 是一种编译器功能，它会为可能不正确的操作添加运行时检查，例如：

- 溢出（乘法溢出、有符号整数溢出）

- 移位基数或指数错误（如移位超过 32 位）
- 整数转换错误

请参考 GCC 文档 中的 `-fsanitize=undefined` 选项，查看支持检查的完整列表。

使能 UBSAN 默认情况下未启用 UBSAN。可以通过在构建系统中添加编译器选项 `-fsanitize=undefined` 在文件、组件或项目级别上使能 UBSAN。

在对使用 SoC 硬件寄存器头文件（`soc/xxx_reg.h`）的代码使能 UBSAN 时，建议使用 `-fno-sanitize=shift-base` 选项禁用移位基数清理器。这是由于 ESP-IDF 寄存器头文件目前包含的模式会对这个特定的清理器选项造成误报。

要在项目级使能 UBSAN，请在项目 CMakeLists.txt 文件的末尾添加以下内容:

```
idf_build_set_property(COMPILE_OPTIONS "-fsanitize=undefined" "-fno-sanitize=shift-
↪base" APPEND)
```

或者，通过 `EXTRA_CFLAGS` 和 `EXTRA_CXXFLAGS` 环境变量来传递这些选项。

使能 UBSAN 会明显增加代码量和数据大小。当为整个应用程序使能 UBSAN 时，微控制器的可用 RAM 无法容纳大多数应用程序（除了一些微小程序）。因此，建议为特定的待测组件使能 UBSAN。

要为项目 CMakeLists.txt 文件中的特定组件（`component_name`）启用 UBSAN，请在文件末尾添加以下内容:

```
idf_component_get_property(lib component_name COMPONENT_LIB)
target_compile_options(${lib} PRIVATE "-fsanitize=undefined" "-fno-sanitize=shift-
↪base")
```

注意: 关于构建属性 和组件属性 的更多信息，请查看构建系统文档。

要为同一组件的 CMakeLists.txt 中的特定组件（`component_name`）使能 UBSAN，在文件末尾添加以下内容:

```
target_compile_options(${COMPONENT_LIB} PRIVATE "-fsanitize=undefined" "-fno-
↪sanitize=shift-base")
```

UBSAN 输出 当 UBSAN 检测到一个错误时，会打印一个信息和回溯，例如:

```
Undefined behavior of type out_of_bounds

Backtrace:0x4008b383:0x3ffcd8b0 0x4008c791:0x3ffcd8d0 0x4008c587:0x3ffcd8f0␣
↪0x4008c6be:0x3ffcd950 0x400db74f:0x3ffcd970 0x400db99c:0x3ffcd9a0
```

当使用*IDF 监视器* 时，回溯会被解码为函数名以及源代码位置，并指向问题发生的位置（这里是 `main.c:128`）：

```
0x4008b383: panic_abort at /path/to/esp-idf/components/esp_system/panic.c:367

0x4008c791: esp_system_abort at /path/to/esp-idf/components/esp_system/system_api.
↪c:106

0x4008c587: __ubsan_default_handler at /path/to/esp-idf/components/esp_system/
↪ubsan.c:152

0x4008c6be: __ubsan_handle_out_of_bounds at /path/to/esp-idf/components/esp_system/
↪ubsan.c:223

0x400db74f: test_ub at main.c:128
```

(下页继续)

(续上页)

```
0x400db99c: app_main at main.c:56 (discriminator 1)
```

UBSAN 报告的错误类型为以下几种：

名称	含义
type_mismatch、type_mismatch_v1	指针值不正确：空、未对齐、或与给定类型不兼容
add_overflow、sub_overflow、mul_overflow、negate_overflow	加法、减法、乘法、求反过程中的整数溢出
divrem_overflow	整数除以 0 或 INT_MIN
shift_out_of_bounds	左移或右移运算符导致的溢出
out_of_bounds	访问超出数组范围
unreachable	执行无法访问的代码
missing_return	Non-void 函数已结束而没有返回值（仅限 C++）
vla_bound_not_positive	可变长度数组的大小不是正数
load_invalid_value	bool 或 enum（仅 C++）变量的值无效（超出范围）
nonnull_arg	对于 nonnull 属性的函数，传递给函数的参数为空
nonnull_return	对于 returns_nonnull 属性的函数，函数返回值为空
builtin_unreachable	调用 __builtin_unreachable 函数
pointer_overflow	指针运算过程中的溢出

4.12 硬件抽象

ESP-IDF 提供了一组用于硬件抽象的 API，支持以不同抽象级别控制外设，相比仅使用 ESP-IDF 驱动程序与硬件进行交互，使用更加灵活。ESP-IDF 硬件抽象适用于编写高性能裸机驱动程序，或尝试将 ESP 芯片移植到另一个平台。

本指南分为以下三个小节：

1. 架构
2. *LL 层（低级层）*
3. *HAL（硬件抽象层）*

警告： 硬件抽象 API（不包括驱动程序和 xxx_types.h）尚处于试验阶段，因此不能算作公共 API。硬件抽象 API 不遵守 ESP-IDF 版本控制方案的 API 名称更改规范。换言之，非主要 ESP-IDF 版本迭代时，硬件抽象 API 的名称可能会更改。

备注： 尽管本文档主要关注外设的硬件抽象，如 UART、SPI、I2C 等，但硬件抽象可以扩展到外设以外其他的硬件部分，如某些 CPU 功能也进行了部分抽象。

4.12.1 架构

ESP-IDF 的硬件抽象由以下层级各组成，从接近硬件的低层级抽象，到远离硬件的高层级抽象。

- 低级层 (LL)
- 硬件抽象层 (HAL)
- 驱动层

LL 层和 HAL 完全包含在 hal 组件中，每一层都依赖于其下方的层级，即驱动层依赖于 HAL 层，HAL 层依赖于 LL 层，LL 层依赖于寄存器头文件。

对于特定外设 xxx，其硬件抽象通常由下表中的头文件组成。其中 **特定目标**指的是文件对于不同目标（即芯片）有不同的实现。然而，对于不同的目标，#include 指令相同，构建系统会自动包含正确版本的头文件和源文件。

表 2: 硬件抽象头文件

包含指令	特定目标	描述
#include 'soc/xxx_caps.h"	是	此头文件包含了 C 宏列表，指明 ESP32-P4 外设 xxx 的各种功能。外设的硬件功能包括通道数量、DMA 支持、硬件 FIFO/缓冲区长度等。
#include "soc/xxx_struct.h" #include "soc/xxx_reg.h"	是	这两个头文件分别以 C 结构体和 C 宏的形式表示外设寄存器，支持通过其中任一头文件，在寄存器级别上操作外设。
#include "soc/xxx_pins.h"	是	如果某些外设的信号映射到 ESP32-P4 的特定管脚上，则该头文件中以 C 宏的形式定义了它们的映射关系。
#include "soc/xxx_periph.h"	否	此头文件主要是为了方便，可以自动包含 xxx_caps.h、xxx_struct.h 和 xxx_reg.h。
#include "hal/xxx_types.h	否	此头文件包含了在 LL、HAL 和驱动层间共享的类型定义和宏。此外，作为公共 API，该头文件可以包含在应用层中。共享的类型和定义通常与具体的实现无关，例如： • 协议相关的类型/宏，如帧、模式、常见总线速度等。 • xxx 外设可能存在的特性/特点，可能存在于任何实现上（与实现无关），例如通道、工作模式、信号放大或衰减强度等。
#include "hal/xxx_ll.h"	是	此头文件包含了硬件抽象的 LL 层。LL 层 API 主要用于将寄存器操作抽象成可读的函数。
#include "hal/xxx_hal.h"	是	HAL 层用于将外设操作步骤抽象成函数，如读取缓冲区、启动传输、处理事件等。HAL 层构建在 LL 层之上。
#include "driver/xxx.h"	否	驱动层是 ESP-IDF 硬件抽象的最高级别。驱动层 API 旨在从 ESP-IDF 应用程序中调用，并在内部使用操作系统的基本功能。因此，驱动层 API 由事件驱动，并可在多线程环境中使用。

4.12.2 LL 层（低级层）

LL 层主要目的是将寄存器字段访问抽象为更容易理解的函数。LL 函数本质是将各种输入/输出参数转换为外设寄存器的寄存器字段，并以获取/设置函数的形式呈现。所有必要的位移、掩码、偏移和寄存器字段的字节顺序都应由 LL 函数处理。

```
//在 xxx_ll.h 内

static inline void xxx_ll_set_baud_rate(xxx_dev_t *hw,
                                        xxx_ll_clk_src_t clock_source,
                                        uint32_t baud_rate) {
    uint32_t src_clk_freq = (source_clk == XXX_SCLK_APB) ? APB_CLK_FREQ : REF_CLK_
↪FREQ;
    uint32_t clock_divider = src_clk_freq / baud;
    // 设置时钟选择字段
    hw->clk_div_reg.divider = clock_divider >> 4;
    // 设置时钟分频器字段
    hw->config.clk_sel = (source_clk == XXX_SCLK_APB) ? 0 : 1;
}
```

(下页继续)

(续上页)

```
static inline uint32_t xxx_ll_get_rx_byte_count(xxx_dev_t *hw) {
    return hw->status_reg.rx_cnt;
}
```

以上代码片段展示了外设 xxx 的典型 LL 函数。LL 函数通常具有以下特点：

- 所有 LL 函数均定义为 `static inline`，因此，由于编译器优化而调用这些函数时，开销最小。这些函数不保证由编译器内联，因此在禁用缓存时（例如从 IRAM ISR 上下文调用）调用的任何 LL 函数都应标记为 `__attribute__((always_inline))`。
- 第一个参数应为指向 `xxx_dev_t` 类型的指针。`xxx_dev_t` 类型表示外设寄存器的结构体，因此第一个参数始终是指向外设寄存器起始地址的指针。请注意，在某些情况下，如果外设具有多个相同寄存器布局的通道，`xxx_dev_t *hw` 可能指向特定通道的寄存器。
- LL 函数应尽可能简短，并且在大多数情况下是确定性的。换句话说，在最糟糕的情况下，LL 函数的运行时间可以在编译时确定。因此，LL 函数中的任何循环都应该是有限的；然而，目前也存在一些例外。
- LL 函数并非线程安全，其上层（驱动层）有责任确保不会同时访问寄存器和寄存器字段。

4.12.3 HAL（硬件抽象层）

HAL 将外设的操作过程建模成一组通用步骤，其中每个步骤都有一个相关联的函数。对于每个步骤，HAL 隐藏（抽象）了外设寄存器的实现细节（即需要设置/读取的寄存器）。通过将外设操作过程建模为一组功能步骤，HAL 可以抽象化（即透明处理）不同目标或芯片版本间的微小硬件实现差异。换句话说，特定外设的 HAL API 在多个目标/芯片版本之间基本保持相同。

以下 HAL 函数示例选自看门狗定时器 (WDT) HAL，每个函数都映射到了 WDT 操作生命周期的某个步骤，从而展示了 HAL 如何将外设的操作抽象为功能步骤。

```
// 初始化某个 WDT
void wdt_hal_init(wdt_hal_context_t *hal, wdt_inst_t wdt_inst, uint32_t prescaler,␣
↪bool enable_intr);

// 配置 WDT 的特定超时阶段
void wdt_hal_config_stage(wdt_hal_context_t *hal, wdt_stage_t stage, uint32_t␣
↪timeout, wdt_stage_action_t behavior);

// 启动 WDT
void wdt_hal_enable(wdt_hal_context_t *hal);

// 喂养（即重置）WDT
void wdt_hal_feed(wdt_hal_context_t *hal);

// 处理 WDT 超时
void wdt_hal_handle_intr(wdt_hal_context_t *hal);

// 停止 WDT
void wdt_hal_disable(wdt_hal_context_t *hal);

// 去初始化 WDT
void wdt_hal_deinit(wdt_hal_context_t *hal);
```

禁用 RTC_WDT

```
wdt_hal_context_t rtc_wdt_ctx = RWDT_HAL_CONTEXT_DEFAULT();
wdt_hal_write_protect_disable(&rtc_wdt_ctx);
wdt_hal_disable(&rtc_wdt_ctx);
wdt_hal_write_protect_enable(&rtc_wdt_ctx);
```

重置 RTC_WDT 计数器

```
wdt_hal_context_t rtc_wdt_ctx = RWDT_HAL_CONTEXT_DEFAULT();
wdt_hal_write_protect_disable(&rtc_wdt_ctx);
wdt_hal_feed(&rtc_wdt_ctx);
wdt_hal_write_protect_enable(&rtc_wdt_ctx);
```

HAL 函数通常具有以下特点：

- HAL 函数的第一个参数是 `xxx_hal_context_t *` 类型。HAL 上下文类型用于存储信息，这些信息与特定外设实例（即上下文实例）相关。HAL 上下文通过 `xxx_hal_init()` 函数初始化，可以存储以下信息：
  - 该实例的通道编号
  - 指向外设（或通道）寄存器的指针（即 `xxx_dev_t *` 类型）
  - 进行中的事务的信息（例如使用中的 DMA 描述符列表的指针）
  - 实例的一些配置值（例如通道配置）
  - 维护实例状态信息的变量（例如表明实例是否正在等待事务完成的标志）
- HAL 函数不应包含任何操作系统原语，如队列、信号量、互斥锁等。所有同步/并发操作应在更高层次（如驱动程序）处理。
- 某些外设的某些步骤可能无法由 HAL 进一步抽象，因此最终成为对 LL 函数的直接封装（或宏）。
- 某些 HAL 函数可能会放置在 IRAM 中，因此可能带有 `IRAM_ATTR` 或放置在单独的 `xxx_hal_iram.c` 源文件中。

4.13 JTAG 调试

本文将介绍如何安装 ESP32-P4 的 OpenOCD 调试环境，以及如何使用 GDB 来调试 ESP32-P4 的应用程序。

备注： 也可以使用 `idf.py monitor` 来调试 ESP32-P4，免于设置 JTAG 或 OpenOCD。请参阅*IDF 监视器* 和*CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME*。

本文档结构如下：

引言 介绍本指南主旨。
工作原理 介绍 ESP32-P4、JTAG (Joint Test Action Group) 接口、OpenOCD 和 GDB 如何相互连接，从而实现 ESP32-P4 的调试功能。
选择 *JTAG* 适配器 介绍有关 JTAG 硬件适配器的选择及参照标准。
安装 *OpenOCD* 介绍如何安装官方预编译好的 OpenOCD 软件包并验证是否安装成功。
配置 *ESP32-P4* 目标板 介绍如何设置 OpenOCD 软件并安装 JTAG 硬件，两项共同构成调试目标.
启动调试器 介绍如何从*Eclipse 集成开发环境* 和*命令行终端* 启动 GDB 调试会话。
调试范例 如果你不熟悉 GDB，请查看此小节以获取*Eclipse 集成开发环境* 以及*命令行终端* 提供的调试示例。
从源码构建 *OpenOCD* 介绍如何在*Windows*，*Linux* 和*macOS* 操作系统上从源码构建 OpenOCD。
注意事项和补充内容 介绍使用 OpenOCD 和 GDB 通过 JTAG 接口调试 ESP32-P4 时的注意事项和补充内容。

4.13.1 引言

乐鑫已完成 OpenOCD 移植，以支持 ESP32-P4 处理器和多核 FreeRTOS 架构（大多数 ESP32-P4 应用程序的基础）。此外，乐鑫还提供了一些 OpenOCD 本身并不支持的工具，以进一步丰富调试功能。

本文将介绍如何在 Linux、Windows 和 macOS 环境下为 ESP32-P4 安装 OpenOCD，并使用 GDB 进行软件调试。除部分安装流程有所不同外，所有操作系统的软件用户界面和使用流程都是相同的。

备注： 本文使用的图片素材来自于 Ubuntu 16.04 LTS 上 Eclipse Neon 3 软件的截图，不同的操作系统（Windows、macOS 或 Linux）或不同的 Eclipse 软件版本在用户界面上可能会有细微差别。

4.13.2 工作原理

通过 JTAG (Joint Test Action Group) 接口使用 OpenOCD 调试 ESP32-P4 时所需要的关键软件和硬件包括 **riscv32-esp-elf-gdb 调试器**、**OpenOCD 片上调试器**和连接到 **ESP32-P4** 目标的 **JTAG 适配器**，如下图“Application Loading and Monitoring”标志所示。

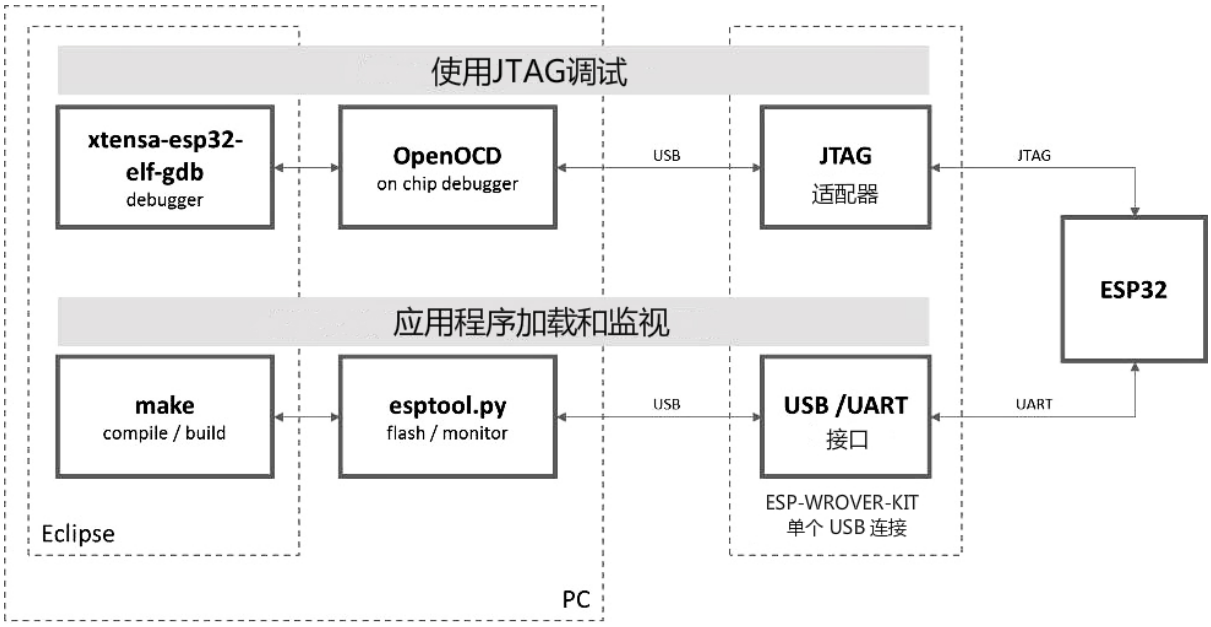


图 10: JTAG 调试 - 概述图

“Application Loading and Monitoring”标志显示一组关键的软件和硬件组件，可用于编译、构建和烧写应用程序到 ESP32-P4 上，以及监视来自 ESP32-P4 的运行诊断信息。

Eclipse 环境集成了 JTAG 调试和应用程序加载、监视的功能，使得软件从编写、编译、加载到调试的迭代过程变得更加快速简单。Eclipse IDE 及其集成的调试软件均适用于 Windows、Linux 和 macOS 平台。根据用户喜好，除了使用 Eclipse 集成开发环境，还可以直接在命令行终端运行 *debugger* 和 *idf.py build*。

仅需一根 USB 线即可高效连接 PC 与 ESP32-P4，因为 ESP32-P4 芯片本身提供了两路 USB 通道，一路连接到 JTAG，另一路连接到 USB 终端。应将 USB 线连接到 ESP32-P4 的 D+/D- USB 管脚，而非通过 USB-UART 芯片连接到串行 RxD/TxD。后文中配置 *ESP32-P4* 目标板 小节将对此进行解释。

4.13.3 选择 JTAG 适配器

上手 JTAG 最快速便捷的方式是将一根 USB 线连接到 ESP32-P4 的 D+/D- USB 管脚，无需使用外部 JTAG 适配器和额外线缆。

如果想使用单独的 JTAG 适配器，请确保其与 ESP32-P4 的电平电压和 OpenOCD 软件都兼容。ESP32-P4 使用的是业界标准的 JTAG 接口，它未使用（实际上也并不需要）TRST 信号脚。JTAG 使用的 IO 管脚由 VDD_3P3_RTC 电源管脚供电（通常连接到外部 3.3 V 的电源轨），因此 JTAG 硬件适配器的管脚需要能够在该电压范围内正常工作。

在软件方面，OpenOCD 支持相当多数量的 JTAG 适配器，请参阅 OpenOCD 支持的适配器列表（请注意这一列表并不完整），其中还列出了兼容 SWD 接口的适配器，但请注意，ESP32-P4 目前并不支持 SWD。此外，硬编码为只支持特定产品线的 JTAG 适配器也无法在 ESP32-P4 上工作，例如仅针对 STM32 系列产品的 ST-LINK 适配器。

保证 JTAG 正常工作需要连接的信号线包括：TDI、TDO、TCK、TMS 和 GND。一些 JTAG 适配器还需要 ESP32-P4 提供一路电源到适配器的某个管脚上（比如 Vtar），用于设置适配器的工作电压。也可以选择将 SRST 信号线连接到 ESP32-P4 的 CH_PD 管脚上，但请注意，目前 OpenOCD 对该信号线提供的支持相当有限。

ESP-Prog 中展示了使用外部电路板进行调试的实例，方法是将其连接到 ESP32-P4 的 JTAG 管脚上。

4.13.4 安装 OpenOCD

如果已经按照快速入门 完成了 ESP-IDF 及其 CMake 构建系统的安装，那么 OpenOCD 已经被默认安装到了你的开发系统中。在设置开发环境 结束后，应该能够在终端中运行如下 OpenOCD 命令:

```
openocd --version
```

终端会输出以下信息（实际版本号可能会更新）：

```
Open On-Chip Debugger v0.12.0-esp32-20240318 (2024-03-18-18:25)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
```

你还可以检查 `OPENOCD_SCRIPTS` 环境变量的值，以确认 OpenOCD 配置文件的路径，Linux 和 macOS 用户可以在终端输入 `echo $OPENOCD_SCRIPTS`，Windows 用户需要输入 `echo %OPENOCD_SCRIPTS%`。如果终端输出了有效路径，则表明已经正确安装 OpenOCD。

如果无法执行上述步骤，请再次阅读快速入门手册，Linux 和 macOS 用户请参考设置安装工具 章节，Windows 用户请参考*ESP-IDF 工具安装器*。

备注： 另外也可以从源代码编译 OpenOCD 工具，详细信息请参阅*从源码构建 OpenOCD* 章节。

4.13.5 配置 ESP32-P4 目标板

OpenOCD 安装完成后就可以配置 ESP32-P4 目标（即带 JTAG 接口的 ESP32-P4 板），具体分为以下三个步骤：

- *配置并连接 JTAG 接口*
- *运行 OpenOCD*
- *上传待调试的应用程序*

配置并连接 JTAG 接口

此步骤取决于使用的 JTAG 和 ESP32-P4 板，请参考以下两种情况。

配置 ESP32-P4 内置 JTAG 接口

ESP32-P4 配备了内置的 JTAG 电路，无需额外芯片，即可进行调试。只需将 USB 电缆连接到 D+/D- 管脚即可。必要的连接如下所示。

配置硬件

表 3: ESP32-P4 管脚及 USB 信号

ESP32-P4 管脚	USB 信号
未更新!	D-
未更新!	D+
5V	V_BUS
GND	接地

请不要将用于 USB 通信的 ESP32-P4 管脚连接到可能会干扰 JTAG 操作的其他硬件上。

配置 USB 驱动程序 JTAG 通信在所有支持的平台上都可工作。Windows 用户可能会遇到 *LIBUSB_ERROR_NOT_FOUND* 错误。请使用 2.8 版本（或更新版本）的*ESP-IDF 工具安装器*，并选择驱动程序 “Espressif - WinUSB 支持 JTAG (ESP32-C3/S3)” 以解决此问题。如果您不想重新运行安装程序，则可以通过 idf-env 实现相同的效果。请在 PowerShell 运行以下命令:

```
Invoke-WebRequest 'https://dl.espressif.com/dl/idf-env/idf-env.exe' -OutFile .\idf-
↪env.exe; .\idf-env.exe driver install --espressif
```

在 Linux 上，需要添加 OpenOCD udev 规则。方法是将以下 udev 规则文件 放置在 `/etc/udev/rules.d` 文件夹中。

配置其他 JTAG 接口

关于适配 OpenOCD 和 ESP32-P4 的 JTAG 接口选择问题，请参考*选择 JTAG 适配器* 章节。然后按照以下步骤进行设置，使其正常工作。

配置 eFuse ESP32-P4 JTAG 接口默认连接至*内置 USB_SERIAL_JTAG 外设*。要使用外部 JTAG 适配器，需将 JTAG 接口切换至 GPIO 管脚。你可以使用 `espefuse.py` 工具来烧录 eFuse，以完成接口转换。

- 烧录 `DIS_USB_JTAG` eFuse 后，USB_SERIAL_JTAG 和 ESP32-P4 的 JTAG 接口之间的连接将被永久禁用，此后你可以将 JTAG 接口连接到 GPIO2-GPIO5。注意，烧录后，USB_SERIAL_JTAG 的 USB CDC 功能仍然可用，即仍然可以通过 USB CDC 进行烧录和查看日志。
- 烧录 `JTAG_SEL_ENABLE` eFuse 后，JTAG 接口的选择将由 strapping 管脚 GPIO34 来决定。ESP32-P4 复位时，如果该 strapping 管脚为低电平，JTAG 接口将使用 GPIO2-GPIO5；如果为高电平，USB_SERIAL_JTAG 将被用作 JTAG 接口。

警告: 请注意，烧录 eFuse 是一项不可逆的操作，请在开始前谨慎考虑以上选项。

配置硬件

1. 找到 JTAG 接口和 ESP32-P4 板上需要相互连接并建立通信的所有管脚或信号。

表 4: ESP32-P4 pins and JTAG signals

ESP32-P4 Pin	JTAG Signal
MTDO / GPIO5	TDO
MTDI / GPIO3	TDI
MTCK / GPIO2	TCK
MTMS / GPIO4	TMS

2. 检查 ESP32-P4 上用于 JTAG 通信的管脚是否被连接到了其它硬件上，这可能会影响 JTAG 的工作。
3. 连接 ESP32-P4 和 JTAG 接口上的管脚或信号。

配置驱动 你可能还需要安装软件驱动，才能使 JTAG 在计算机上正常工作，请参阅你所使用的 JTAG 适配器的有关文档，获取相关详细信息。

在 Linux 中，请务必将 udev 规则文件 复制到 `/etc/udev/rules.d` 目录中，以添加 OpenOCD udev 规则。

连接 将 JTAG 接口连接到计算机，打开 ESP32-P4 和 JTAG 接口板上的电源，然后检查计算机是否可以识别到 JTAG 接口。

如需继续设置调试环境，请前往*运行 OpenOCD* 章节。

运行 OpenOCD

配置完目标并将其连接到电脑后，即可启动 OpenOCD。

打开终端，按照快速入门指南中的*设置好开发环境* 章节进行操作，然后运行如下命令，以启动 OpenOCD（该命令适用于 Windows、Linux 和 macOS）：

```
openocd -f board/esp32p4-builtin.cfg
```

备注： 上述命令中 `-f` 选项后跟的配置文件专用于 ESP32-P4 through built-in USB connection。基于具体使用的硬件，你可能需要选择不同的配置文件，具体内容请参阅*根据目标芯片配置 OpenOCD*。

例如，对于带有用于 JTAG 连接的 FT2232H 或 FT232H 芯片的定制板，或带有 ESP-Prog 的定制板，可使用 `Not Updated!`。

现在应该可以看到如下输出（此日志来自 ESP32-P4 through built-in USB connection）：

```
user-name@computer-name:~/esp/esp-idf$ openocd -f board/esp32p4-builtin.cfg
Open On-Chip Debugger v0.12.0-esp32-20240318 (2024-03-18-18:25)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselecting 'jtag'
Info : esp_usb_jtag: VID set to 0x303a and PID to 0x1001
Info : esp_usb_jtag: capabilities descriptor set to 0x2000
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : esp_usb_jtag: serial (60:55:F9:F9:04:AD)
Info : esp_usb_jtag: Device found. Base speed 24000KHz, div range 1 to 255
Info : clock speed 24000 kHz
Info : JTAG tap: esp32p4.tap0 tap/device found: 0x00012c25 (mfg: 0x612 (Espressif␣
↪Systems), part: 0x0012, ver: 0x0)
Info : JTAG tap: esp32p4.tap1 tap/device found: 0x00012c25 (mfg: 0x612 (Espressif␣
↪Systems), part: 0x0012, ver: 0x0)
Info : [esp32p4.hp.cpu0] datacount=1 progbufsize=2
Info : [esp32p4.hp.cpu0] Core 0 made part of halt group 1.
Info : [esp32p4.hp.cpu0] Examined RISC-V core; found 2 harts
Info : [esp32p4.hp.cpu0]  XLEN=32, misa=0x40901125
Info : [esp32p4.hp.cpu0] Examination succeed
Info : [esp32p4.hp.cpu1] datacount=1 progbufsize=2
Info : [esp32p4.hp.cpu1] Core 1 made part of halt group 1.
Info : [esp32p4.hp.cpu1] Examined RISC-V core; found 2 harts
Info : [esp32p4.hp.cpu1]  XLEN=32, misa=0x40901125
Info : [esp32p4.hp.cpu1] Examination succeed
Info : [esp32p4.hp.cpu0] starting gdb server on 3333
Info : Listening on port 3333 for gdb connections
```

- 如果出现指示权限问题的错误，请打开 `~/esp/openocd-esp32` 目录，参阅 OpenOCD README 文件中关于 “Permissions delegation” 的说明。
- 如果遇到无法找到配置文件的错误，例如 `Can't find board/esp32p4-builtin.cfg`，请检查 `OPENOCD_SCRIPTS` 环境变量是否设置正确，OpenOCD 根据此变量来查找 `-f` 指定的文件，详见*安装 OpenOCD*。此外，还需要检查配置文件是否确实位于该路径下。
- 如果出现 JTAG 错误（例如输出为 `...all ones` 或 `...all zeroes`），请检查硬件连接是否正确，除了 ESP32-P4 的管脚之外是否还有其他信号连接到了 JTAG，并查看是否所有器件都已经上电。

上传待调试的应用程序

按照正常步骤构建并上传 ESP32-P4 应用程序，具体请参阅*第五步：开始使用 ESP-IDF 吧* 章节。

除此以外，还可以使用 OpenOCD 通过 JTAG 接口将应用程序镜像烧写到 flash 中，命令如下:

```
openocd -f board/esp32p4-builtin.cfg -c "program_esp filename.bin 0x10000 verify␣
↪exit"
```

其中 OpenOCD 的烧写命令 `program_esp` 格式如下:

```
program_esp <image_file> <offset> [verify] [reset] [exit] [compress]
[encrypt]
```

- `image_file` - 程序镜像文件存放的路径
- `offset` - 镜像烧写到 flash 中的偏移地址
- `verify` - 烧写完成后校验 flash 中的内容（可选）
- `reset` - 烧写完成后重启目标（可选）
- `exit` - 烧写完成后退出 OpenOCD（可选）
- `compress` - 烧写开始前压缩镜像文件（可选）
- `encrypt` - 烧写到 flash 前加密二进制文件，与 `idf.py encrypted-flash` 功能相同（可选）
- `no_clock_boost` - 禁用在烧写前将目标时钟频率设置为其最大可能值（可选）。默认情况下禁用该选项，即默认启用时钟提升。
- `restore_clock` - 可选。烧写完成后将时钟频率恢复到初始值。默认情况下不启用。

现在可以调试应用程序了，请按照以下章节中的步骤进行操作。

4.13.6 启动调试器

ESP32-P4 的工具链中带有 GNU 调试器（简称 GDB），它和其它工具链软件共同存放于 riscv32-esp-elf-gdb 中。除了直接在命令行终端中调用并操作 GDB 外，也可以在 IDE（例如 Eclipse、Visual Studio Code 等）中进行调用，使用图形用户界面间接操作 GDB，这一方法无需在终端中输入任何命令。

关于调试器的使用方法，详见以下链接。

- 使用 *Eclipse* 调试
- 使用命令行调试
- 使用 VS Code 调试

建议首先检查调试器能否在命令行终端 下正常工作，然后再使用 Eclipse 集成开发环境 进行调试工作。

4.13.7 调试范例

本节适用于不熟悉 GDB 的用户，下文将使用 get-started/blink 下简单的应用程序来演示调试会话的工作流程，同时会介绍以下常用的调试操作:

1. 浏览代码，查看堆栈和线程
2. 设置和清除断点
3. 手动暂停目标
4. 单步执行代码
5. 查看并设置内存
6. 观察和设置程序变量
7. 设置条件断点

此外还会提供在命令行终端进行调试 下使用 GDB 调试的案例。

备注: 调试 *FreeRTOS* 对象 目前仅适用于命令行调试。

在演示之前，请完成 ESP32-P4 目标板设置并加载 get-started/blink 至 ESP32-P4 中。

4.13.8 从源码构建 OpenOCD

以下文档分别介绍了如何在各操作系统平台上从源码构建 OpenOCD。

Windows 环境下从源码编译 OpenOCD

备注： 本文介绍了如何从 OpenOCD 源文件构建二进制文件。如果你想要更快速地构建，也可以从 乐鑫 GitHub 直接下载 OpenOCD 的预构建二进制文件，而无需自己编译（详细信息，请参阅*安装 OpenOCD*）。

备注： 本文涉及的命令行操作均在装有 MINGW32 子系统的 MSYS2 shell 环境中进行了验证。

安装依赖的软件包 安装编译 OpenOCD 所需的软件包：

```
pacman -S --noconfirm --needed autoconf automake git make \
mingw-w64-i686-gcc \
mingw-w64-i686-toolchain \
mingw-w64-i686-libtool \
mingw-w64-i686-pkg-config \
mingw-w64-cross-winpthreads-git \
p7zip
```

下载 OpenOCD 源码 支持 ESP32-P4 的 OpenOCD 源码可以从乐鑫官方 GitHub 获取，网址为 https://github.com/espressif/openocd-esp32。你可以在 Git 中使用以下命令来拉取源代码：

```
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码保存在 `~/esp/openocd-esp32` 目录下。

下载 libusb 构建 OpenOCD 需使用 libusb 库。请执行以下命令来下载特定版本的 libusb，并将其解压至当前目录。

```
wget https://github.com/libusb/libusb/releases/download/v1.0.22/libusb-1.0.22.7z
7z x -olibusb ./libusb-1.0.22.7z
```

现在需要导出以下变量，以便将 libusb 库与 OpenOCD 构建相关联。

```
export CPPFLAGS="$CPPFLAGS -I${PWD}/libusb/include/libusb-1.0"
export LDFLAGS="$LDFLAGS -L${PWD}/libusb/MinGW32/.libs/dll"
```

构建 OpenOCD 配置和构建 OpenOCD，请参考以下命令：

```
cd ~/esp/openocd-esp32
export CPPFLAGS="$CPPFLAGS -D__USE_MINGW_ANSI_STDIO=1 -Wno-error"; export CFLAGS="
↪$CFLAGS -Wno-error"
./bootstrap
./configure --disable-doxygen-pdf --enable-ftdi --enable-jlink --enable-ulink --
↪build=i686-w64-mingw32 --host=i686-w64-mingw32
make
cp ../libusb/MinGW32/dll/libusb-1.0.dll ./src
cp /opt/i686-w64-mingw32/bin/libwinpthread-1.dll ./src
```

构建完成后，OpenOCD 的二进制文件将被保存于 `~/esp/openocd-esp32/src/` 目录下。

你也可以调用 `make install`，将其复制到指定位置。

- 你可以在配置 OpenOCD 时指定这一位置，也可以在调用 `make install` 前设置 `export DESTDIR="/custom/install/dir"`。
- 如果你已经安装过其他开发平台的 OpenOCD，请跳过此步骤，否则原来的 OpenOCD 可能会被覆盖。

备注:

- 如果发生错误，请解决后再次尝试编译，直到 `make` 成功为止。
- 如果 OpenOCD 存在子模块问题，请 `cd` 到 `openocd-esp32` 目录，并输入 `git submodule update --init` 命令。
- 如果 `./configure` 成功运行，JTAG 被使能的信息会被打印在 `OpenOCD configuration summary` 下面。
- 如果你的设备信息未显示在日志中，请根据 `../openocd-esp32/doc/INSTALL.txt` 文中的描述使用 `./configure` 启用它。
- 有关编译 OpenOCD 的详细信息，请参阅 `openocd-esp32/README.Windows`。
- 请记得将 *libusb-1.0.dll* 和 *libwinpthread-1.dll* 从 `~/esp/openocd-esp32/src` 复制到 *OOCD_INSTALLDIR/bin*。

一旦 `make` 过程完成，OpenOCD 的可执行文件会被保存到 `~/esp/openocd-esp32/src/openocd` 目录下。

完整编译过程 OpenOCD 编译过程中所调用的所有命令都已包含在以下代码片段中，你可以将其复制到 shell 脚本中，以便快速执行:

```
pacman -S --noconfirm --needed autoconf automake git make mingw-w64-i686-gcc mingw-
↪w64-i686-toolchain mingw-w64-i686-libtool mingw-w64-i686-pkg-config mingw-w64-
↪cross-winpthreads-git p7zip
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git

wget https://github.com/libusb/libusb/releases/download/v1.0.22/libusb-1.0.22.7z
7z x -olibusb ./libusb-1.0.22.7z
export CPPFLAGS="$CPPFLAGS -I${PWD}/libusb/include/libusb-1.0"; export LDFLAGS="
↪$LDFLAGS -L${PWD}/libusb/MinGW32/.libs/dll"

export CPPFLAGS="$CPPFLAGS -D__USE_MINGW_ANSI_STDIO=1 -Wno-error"; export CFLAGS="
↪$CFLAGS -Wno-error"
cd ~/esp/openocd-esp32
./bootstrap
./configure --disable-doxygen-pdf --enable-ftdi --enable-jlink --enable-ulink --
↪build=i686-w64-mingw32 --host=i686-w64-mingw32
make
cp ../libusb/MinGW32/dll/libusb-1.0.dll ./src
cp /opt/i686-w64-mingw32/bin/libwinpthread-1.dll ./src

# # optional
# export DESTDIR="$PWD"
# make install
# cp ./src/libusb-1.0.dll $DESTDIR/mingw32/bin
# cp ./src/libwinpthread-1.dll $DESTDIR/mingw32/bin
```

下一步 想要进一步配置调试环境，请前往配置 *ESP32-P4* 目标板 章节。

Linux 环境下从源码编译 OpenOCD

除了从 Espressif 官方 直接下载 OpenOCD 可执行文件，你还可以选择从源码编译得到 OpenOCD。如果想要快速设置 OpenOCD 而不是自行编译，请备份好当前文件，前往*安装 OpenOCD* 章节查阅。

下载 OpenOCD 源码 支持 ESP32-P4 的 OpenOCD 源代码可以从乐鑫官方的 GitHub 获得，网址为 https://github.com/espressif/openocd-esp32。请使用以下命令来下载源代码:

```
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码被保存在 `~/esp/openocd-esp32` 目录中。

安装依赖的软件包 安装编译 OpenOCD 所需的软件包。

备注: 依次安装以下软件包，检查安装是否成功，然后继续下一个软件包的安装。在进行下一步操作之前，要先解决当前报告的问题。

```
sudo apt-get install make
sudo apt-get install libtool
sudo apt-get install pkg-config
sudo apt-get install autoconf
sudo apt-get install automake
sudo apt-get install texinfo
sudo apt-get install libusb-1.0
```

备注:

- pkg-config 应为 0.2.3 或以上的版本。
- autoconf 应为 2.6.4 或以上的版本。
- automake 应为 1.9 或以上的版本。
- 当使用 USB-Blaster，ASIX Presto，OpenJTAG 和 FT2232 作为适配器时，需要下载安装 libFTDI 和 FTD2XX 的驱动。
- 当使用 CMSIS-DAP 时，需要安装 HIDAPI。

构建 OpenOCD 配置和构建 OpenOCD 的流程如下:

```
cd ~/esp/openocd-esp32
./bootstrap
./configure
make
```

你可以选择最后再执行 `sudo make install`，如果你已经安装过别的开发平台的 OpenOCD，请跳过这个步骤，因为它可能会覆盖掉原来的 OpenOCD。

备注:

- 如果发生错误，请解决后再次尝试编译，直到 `make` 成功为止。
- 如果 OpenOCD 存在子模块问题，请 `cd` 到 `openocd-esp32` 目录，并输入 `git submodule update --init` 命令。
- 如果 `./configure` 成功运行，JTAG 被使能的信息会被打印在 `OpenOCD configuration summary` 下面。
- 如果你的设备信息未显示在日志中，请根据 `../openocd-esp32/doc/INSTALL.txt` 文中的描述使用 `./configure` 启用它。
- 有关编译 OpenOCD 的详细信息，请参阅 `openocd-esp32/README`。

一旦 make 过程成功结束，OpenOCD 的可执行文件会被保存到 ~/openocd-esp32/bin 目录中。

下一步 想要进一步配置调试环境，请前往*配置 ESP32-P4 目标板* 章节。

MacOS 环境下从源码编译 OpenOCD

除了从 Espressif 官方 直接下载 OpenOCD 可执行文件，你还可以选择从源码编译得到 OpenOCD。如果想要快速设置 OpenOCD 而不是自行编译，请备份好当前文件，前往*安装 OpenOCD* 章节查阅。

下载 OpenOCD 源码 支持 ESP32-P4 的 OpenOCD 源代码可以从乐鑫官方的 GitHub 获得，网址为 https://github.com/espressif/openocd-esp32。请使用以下命令来下载源代码:

```
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码被保存在 ~/esp/openocd-esp32 目录中。

安装依赖的软件包 使用 Homebrew 安装编译 OpenOCD 所需的软件包:

```
brew install automake libtool libusb wget gcc@4.9 pkg-config
```

构建 OpenOCD 配置和构建 OpenOCD 的流程如下:

```
cd ~/esp/openocd-esp32
./bootstrap
./configure
make
```

你可以选择最后再执行 sudo make install，如果你已经安装过别的开发平台的 OpenOCD，请跳过这个步骤，因为它可能会覆盖掉原来的 OpenOCD。

备注:

- 如果发生错误，请解决后再次尝试编译，直到 make 成功为止。
- 发生 Unknown command 'raggedright' 错误可能是因为安装的 texinfo 版本不对，或是由于没有将其添加到 PATH 路径。为了解决该问题，在运行 ./bootstrap 前，请先运行如下命令确保安装合适版本的 texinfo 并将其添加到 PATH 路径:

  ```
  brew install texinfo
  export PATH=/usr/local/opt/texinfo/bin:$PATH
  ```

- 如果 OpenOCD 存在子模块问题，请 cd 到 openocd-esp32 目录，并输入 git submodule update --init 命令。
- 如果 ./configure 成功运行，JTAG 被使能的信息会被打印在 OpenOCD configuration summary 下面。
- 如果你的设备信息未显示在日志中，请根据 ../openocd-esp32/doc/INSTALL.txt 文中的描述使用 ./configure 启用它。
- 有关编译 OpenOCD 的详细信息，请参阅 openocd-esp32/README.OSX。

一旦 make 过程成功结束，OpenOCD 的可执行文件会被保存到 ~/esp/openocd-esp32/src/openocd 目录中。

下一步 想要进一步配置调试环境，请前往配置 *ESP32-P4 目标板* 章节。

本文档在演示中所使用的 OpenOCD 是预编译好的二进制发行版，在安装 *OpenOCD* 章节中有所介绍。

如果要使用本地从源代码编译的 OpenOCD 程序，需要将相应可执行文件的路径修改为 `src/openocd`，并设置 `OPENOCD_SCRIPTS` 环境变量，使得 OpenOCD 能够找到配置文件。Linux 和 macOS 用户可以执行:

```
cd ~/esp/openocd-esp32
export OPENOCD_SCRIPTS=$PWD/tcl
```

Windows 用户可以执行:

```
cd %USERPROFILE%\esp\openocd-esp32
set "OPENOCD_SCRIPTS=%CD%\tcl"
```

针对 Linux 和 macOS 用户，运行本地编译的 OpenOCD 的示例:

```
src/openocd -f board/esp32p4-builtin.cfg
```

Windows 用户的示例如下:

```
src\openocd -f board/esp32p4-builtin.cfg
```

4.13.9 注意事项和补充内容

本节列出了上文中提到的所有注意事项和补充内容的链接。

注意事项和补充内容

本节提供了本指南中各部分提到的一些注意事项和补充内容。

可用的断点和观察点 ESP32-P4 调试器支持 3 个硬件断点和 64 个软件断点。硬件断点是由 ESP32-P4 芯片内部的逻辑电路实现的，能够设置在代码的任何位置：flash 或者 IRAM 的代码区域。除此以外，OpenOCD 实现了两种软件断点：flash 断点（最多 32 个）和 IRAM 断点（最多 32 个）。目前 GDB 无法在 flash 中设置软件断点，因此除非解决此限制，否则这些断点只能由 OpenOCD 模拟为硬件断点（详细信息可以参阅下文）。ESP32-P4 还支持 3 个观察点，所以可以观察 3 个变量的变化或者通过 GDB 命令 `watch myVariable` 来读取变量的值。请注意 menuconfig 中的*CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK* 选项会使用最后一个观察点，如果你想在 OpenOCD 或者 GDB 中再次尝试使用这个观察点，可能不会得到预期的结果。详情请查看 menuconfig 中的帮助文档。

关于断点的补充知识 使用软件 flash 模拟部分硬件断点的意思就是当使用 GDB 命令 `hb myFunction` 给某个函数设置硬件断点时，如果该函数位于 flash 中，并且此时还有可用的硬件断点，那调试器就会使用硬件断点，否则就使用 32 个软件 flash 断点中的一个来模拟。这个规则同样适用于 `b myFunction` 之类的命令，在这种情况下，GDB 会自己决定该使用哪种类型的断点。如果 `myFunction` 位于可写区域 (IRAM)，那就会使用软件 IRAM 断点，否则就会像处理 `hb` 命令一样使用硬件断点或者软件 flash 断点。

flash 映射 vs 软件 flash 断点 为了在 flash 中设置或者清除软件断点，OpenOCD 需要知道它们在 flash 中的地址。为了完成从 ESP32-P4 的地址空间到 flash 地址的转换，OpenOCD 使用 flash 中程序代码区域的映射。这些映射被保存在程序映像的头部，位于二进制数据（代码段和数据段）之前，并且特定于写入 flash 的每一个应用程序的映像。因此，为了支持软件 flash 断点，OpenOCD 需要知道待调试的应用程序映像在 flash 中的位置。默认情况下，OpenOCD 会在 0x8000 处读取分区表并使用第一个找到的应用程序映像的映射，但是也可能会存在无法工作的情况，比如分区表不在标准的 flash 位置，甚至可能有多个映像：一个出厂映像和两个 OTA 映像，你可能想要调试其中的任意一个。为了涵盖所有可能的调试情况，

OpenOCD 支持特殊的命令，用于指定待调试的应用程序映像在 flash 中的具体位置。该命令具有以下格式：

```
esp appimage_offset <offset>
```

偏移量应为十六进制格式，如果要恢复默认行为，可以将偏移地址设置为 `-1` 。

备注： 由于 GDB 在连接 OpenOCD 时仅仅请求一次内存映射，所以可以在 TCL 配置文件中指定该命令，或者通过命令行传递给 OpenOCD。对于后者，命令行示例如下：

```
openocd -f board/esp32p4-builtin.cfg -c "init; halt; esp appimage_offset 0x210000"
```

另外还可以通过 OpenOCD 的 telnet 会话执行该命令，然后再连接 GDB，不过这种方式似乎没有那么便捷。

“next” 命令无法跳过子程序的原因 当使用 `next` 命令单步执行代码时，GDB 会在子程序的前面设置一个断点，这样就可以跳过进入子程序内部的细节。如果 3 个断点都已经设置好，那么 `next` 命令将不起作用。在这种情况下，请删掉其他断点以使其中一个变得可用。当所有断点都已经被使用时，`next` 命令会像 `step` 命令一样工作，调试器就会进入子程序内部。

OpenOCD 支持的编译时的选项 ESP-IDF 有一些针对 OpenOCD 调试功能的选项可以在编译时进行设置：

- *CONFIG_ESP_DEBUG_OCDAWARE* 默认会被使能。如果程序抛出了不可修复或者未处理的异常，并且此时已经连接上了 JTAG 调试器（即 OpenOCD 正在运行），那么 ESP-IDF 将会进入调试器工作模式。
- *CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK* 默认没有使能。在所有任务堆栈的末尾设置观察点，从 1 号开始索引。这是调试任务堆栈溢出的最准确的方式。

更多有关设置编译时的选项的信息，请参阅项目配置菜单。

支持 FreeRTOS OpenOCD 完全支持 ESP-IDF 自带的 FreeRTOS 操作系统，GDB 会将 FreeRTOS 中的任务当做线程。使用 GDB 命令 `i threads` 可以查看所有的线程，使用命令 `thread n` 可以切换到某个具体任务的堆栈，其中 `n` 是线程的编号。检测 FreeRTOS 的功能可以在配置目标时被禁用。更多详细信息，请参阅根据目标芯片配置 *OpenOCD*。

GDB 具有 FreeRTOS 支持的 Python 扩展模块。在系统要求满足的情况下，通过 `idf.py gdb` 命令，ESP-IDF 会将该模块自动加载到 GDB 中。详细信息请参考调试 *FreeRTOS* 对象。

优化 JTAG 的速度 为了实现更高的数据通信速率同时最小化丢包数，建议优化 JTAG 时钟频率的设置，使其达到 JTAG 能稳定运行的最大值。为此，请参考以下建议。

1. 如果 CPU 以 80 MHz 运行，则 JTAG 时钟频率的上限为 20 MHz；如果 CPU 以 160 MHz 或者 240 MHz 运行，则上限为 26 MHz。
2. 根据特定的 JTAG 适配器和连接线缆的长度，你可能需要将 JTAG 的工作频率降低至 20 MHz 或 26 MHz 以下。
3. 在某些特殊情况下，如果你看到 DSR/DIR 错误（并且它并不是由 OpenOCD 试图从一个没有物理存储器映射的地址空间读取数据而导致的），请降低 JTAG 的工作频率。
4. ESP-WROVER-KIT 能够稳定运行在 20 MHz 或 26 MHz 频率下。

调试器的启动命令的含义 在启动时，调试器发出一系列命令来复位芯片并使其在特定的代码行停止运行。这个命令序列（如下所示）支持自定义，用户可以选择在最方便合适的代码行开始调试工作。

- `set remote hardware-watchpoint-limit 3` —限制 GDB 使用芯片支持的硬件观察点数量，ESP32-P4 支持 3 个观察点。更多详细信息，请查阅 GDB 配置远程目标 。
- `mon reset halt` —复位芯片并使 CPU 停止运行。

- `maintenance flush register-cache` —monitor (`mon`) 命令无法通知 GDB 目标状态已经更改，GDB 会假设在 `mon reset halt` 之前所有的任务堆栈仍然有效。实际上，复位后目标状态将发生变化。执行 `maintenance flush register-cache` 是一种强制 GDB 从目标获取最新状态的方法。
- `thb app_main` —在 `app_main` 处插入一个临时的硬件断点，如果有需要，可以将其替换为其他函数名。
- `c` —恢复程序运行，它将会在 `app_main` 的断点处停止运行。

根据目标芯片配置 OpenOCD OpenOCD 有很多种配置文件（`*.cfg`），它们位于 OpenOCD 安装目录的 `share/openocd/scripts` 子目录中（或者在 OpenOCD 源码目录的 `tcl/scripts` 目录中）。本文主要介绍 `board`，`interface` 和 `target` 这三个目录。

- `interface` 包含了例如 ESPProg、J-Link 这些 JTAG 适配器的配置文件。
- `target` 包含了目标芯片或者模组的配置文件。
- `board` 包含有内置了 JTAG 适配器的开发板的配置文件，这些配置文件会根据实际的 JTAG 适配器和芯片/模组来导入某个具体的 `interface` 和 `target` 的配置。

ESP32-P4 可以使用的配置文件如下表所示:

表 5: OpenOCD configuration files for ESP32-P4

Name	Description
`board/esp32p4-builtin.cfg`	Board configuration file for ESP32-P4 through built-in USB, includes target and adapter configuration.
`board/esp32p4-ftdi.cfg`	Board configuration file for ESP32-P4 for via externally connected FTDI-based probe like ESP-Prog, includes target and adapter configuration.
`target/esp32p4.cfg`	ESP32-P4 target configuration file. Can be used together with one of the `interface/` configuration files.
`interface/esp_usb_jtag.cfg`	JTAG adapter configuration file for ESP32-P4.
`interface/ftdi/esp32p4_ftdi.cfg`	JTAG adapter configuration file for ESP-Prog boards.

如果你使用的开发板已经有了一份预定义好的配置文件，你只须将该文件通过 `-f` 参数告诉 OpenOCD。

如果你的开发板不在上述列表中，你需要使用多个 `-f` 参数来告诉 OpenOCD 你选择的 `interface` 和 `target` 配置文件。

自定义配置文件 OpenOCD 的配置文件是用 TCL 语言编写的，包含了定制和编写脚本的各种选项。这在非标准调试的场景中非常有用，更多关于 TCL 脚本的内容请参考 OpenOCD 参考手册。

OpenOCD 中的配置变量 你还可以视情况在导入 `target` 配置文件之前，设定如下变量的值。可以写在自定义配置文件中，或者通过命令行传递。

TCL 语言中为变量赋值的语法是:

```
set VARIABLE_NAME value
```

在命令行中为变量赋值请参考如下示例（请把.cfg 配置文件替换成你自己的开发板配置）：

```
openocd -c 'set VARIABLE_NAME value' -f board/esp-xxxxx-kit.cfg
```

请切记，一定要在导入配置文件之前设置这些变量，否则变量的值将不会生效。为多个变量赋值需要重复多次 `-c` 选项。

表 6: 通用的 ESP 相关的 OpenOCD 变量

变量名	描述
ESP_RTOS	设置成 none 可以关闭 OpenOCD 对 RTOS 的支持，这样的话，你将无法在 GDB 中查看到线程列表。这个功能在调试 FreeRTOS 本身的时候会很有用，可以单步调试调度器的代码。
ESP_FLASH_SIZE	设置成 0 可以关闭对 flash 断点的支持。
ESP_SEMIHOST_BASEDI	设置 semihosting 在主机端的默认目录。
ESP_ONLYCPU	对于多核芯片，将该值设置为 1 可以仅启用单核调试功能

复位 ESP32-P4 通过在 GDB 中输入 `mon reset` 或者 `mon reset halt` 来复位板子。

JTAG 管脚是否能用于其他功能 ESP32-P4 包含一个可用于调试的 USB Serial/JTAG 控制器。默认情况下，ESP32-P4 JTAG 接口连接到内置的外设 USB Serial/JTAG。详细信息可参阅*配置 ESP32-P4 内置 JTAG 接口*。

如果 USB Serial/JTAG 控制器用于调试，GPIO2-GPIO5 可用于其他功能。

如果用户通过烧录 eFuse 将 USB JTAG 接口切换为 GPIO，GPIO2-GPIO5 则可用于 JTAG 调试，但用于 JTAG 调试的 GPIO2-GPIO5 不能再用于其他功能。

如果除了 ESP32-P4 模组和 JTAG 适配器之外的其他硬件也连接到了 JTAG 管脚，那么 JTAG 的操作可能会受到干扰。ESP32-P4 JTAG 使用以下管脚：

表 7: ESP32-P4 pins and JTAG signals

ESP32-P4 Pin	JTAG Signal
MTDO / GPIO5	TDO
MTDI / GPIO3	TDI
MTCK / GPIO2	TCK
MTMS / GPIO4	TMS

如果用户应用程序更改了 JTAG 管脚的配置，JTAG 通信可能会失败。如果 OpenOCD 正确初始化（检测到芯片全部 CPU 内核），但在程序运行期间失去了同步并报出大量 DTR/DIR 错误，原因可能是应用程序将 JTAG 管脚重新配置为其他功能或者用户忘记将 Vtar 连接到 JTAG 适配器。

JTAG 与 flash 加密和安全引导 默认情况下，开启了 flash 加密和（或者）安全引导后，系统在首次启动时，引导程序会烧写 eFuse 的某个比特，从而将 JTAG 永久关闭。

请注意，一旦 JTAG 被永久禁用，就无法重新启用以访问 JTAG。但是我们也提供了暂时禁用 (soft disable) JTAG 的选项。有关如何暂时禁用以及重新启用 JTAG，请参考*HMAC 启用 JTAG 接口*。

Kconfig 配置项*CONFIG_SECURE_BOOT_ALLOW_JTAG* 可以改变这个默认行为，使得用户即使开启了安全引导或者 flash 加密，仍会保留 JTAG 的功能。

然而，因为设置*软件断点* 的需要，OpenOCD 会尝试自动读写 flash 中的内容，这会带来两个问题：

- 软件断点和 flash 加密是不兼容的，目前 OpenOCD 尚不支持对 flash 中的内容进行加密和解密。
- 如果开启了安全引导功能，设置软件断点会改变被签名的程序的摘要，从而使得签名失效。这也意味着，如果设置了软件断点，系统会在下次重启时的签名验证阶段失败，导致无法启动。

关闭 JTAG 的软件断点功能，可以在启动 OpenOCD 时在命令行额外加一项配置参数 `-c 'set ESP_FLASH_SIZE 0'`，请参考*OpenOCD 中的配置变量*。

备注： 同样地，当启用该选项，并且在调试过程中设置了软件断点，引导程序将无法校验通过应用程序的签名。

报告 OpenOCD/GDB 的问题 如果你遇到 OpenOCD 或者 GDB 程序本身的问题，并且在网上没有找到可用的解决方案，请前往 https://github.com/espressif/openocd-esp32/issues 新建一个议题。

1. 请在问题报告中提供你使用的配置的详细信息：
   a. JTAG 适配器类型。
   b. 用于编译和加载正在调试的应用程序的 ESP-IDF 版本号。
   c. 用于调试的操作系统的详细信息。
   d. 操作系统是在本地计算机运行还是在虚拟机上运行？
2. 创建一个能够演示问题的简单示例工程，描述复现该问题的步骤。且这个调试示例不能受到 Wi-Fi 协议栈引入的非确定性行为的影响，这样再次遇到同样问题时，更容易复现。
3. 在启动命令中添加额外的参数来输出调试日志。
   OpenOCD 端：

   ```
   openocd -l openocd_log.txt -d3 -f board/esp32p4-builtin.cfg
   ```

   这种方式会将日志输出到文件，但是它会阻止调试信息打印在终端上。当有大量信息需要输出的时候（比如调试等级提高到 `-d3`）这是个不错的选择。如果你仍然希望在屏幕上看到调试日志，请改用以下命令：

   ```
   openocd -d3 -f board/esp32p4-builtin.cfg 2>&1 | tee openocd.log
   ```

   Debugger 端：

   ```
   riscv32-esp-elf-gdb -ex "set remotelogfile gdb_log.txt" <all other options>
   ```

   也可以将命令 `remotelogfile gdb_log.txt` 添加到 `gdbinit` 文件中。
4. 请将 `openocd_log.txt` 和 `gdb_log.txt` 文件附在你的问题报告中。

4.13.10 相关文档

使用调试器

本节介绍以下几种配置和运行调试器的方法：

- 使用 *Eclipse* 调试
- 使用命令行调试
- 使用 *idf.py* 进行调试

关于如何使用 VS Code 进行调试，请参阅文档 使用 VS Code 调试。

使用 Eclipse 调试

备注： 建议首先通过*idf.py* 或命令行 检查调试器是否正常工作，然后再转到使用*Eclipse* 平台。

作为一款集成开发环境 (IDE)，Eclipse 提供了一套强大的工具，用于开发和调试软件应用程序。对于 ESP-IDF 应用程序，IDF Eclipse 插件 提供了两种调试方式：

1. ESP-IDF GDB OpenOCD 调试
2. GDB 硬件调试

默认情况下，Eclipse 通过 GDB 硬件调试插件支持 OpenOCD 调试。该调试方式需要从命令行启动 OpenOCD 服务器，并在 Eclipse 中配置 GDB 客户端，整个过程耗时且容易出错。

为了使调试过程更加容易，IDF Eclipse 插件 提供了定制的 ESP-IDF GDB OpenOCD 调试功能，支持在 Eclipse 内部配置好 OpenOCD 服务器和 GDB 客户端。该插件已经设置好所有必需的配置参数，点击一个按钮即可开始调试。

因此，建议通过 IDF Eclipse 插件 进行 ESP-IDF GDB OpenOCD 调试。

GDB 硬件调试

备注： 只有在无法使用 ESP-IDF GDB OpenOCD 调试 的情况下，才建议使用 GDB 硬件调试。

首先，打开 Eclipse，选择 `Help` > `Install New Software` 来安装 `GDB Hardware Debugging` 插件。

安装完成后，按照以下步骤配置调试会话。请注意，一些配置参数是通用的，有些则针对特定项目。我们会通过配置"blink" 示例项目的调试环境来进行展示，请先按照 Eclipse Plugin 介绍的方法将该示例项目添加到 Eclipse 的工作空间。示例项目 get-started/blink 的源代码可以在 ESP-IDF 仓库的 examples 目录下找到。

1. 在 Eclipse 中，进入 `Run` > `Debug Configuration`，会出现一个新的窗口。在窗口的左侧窗格中，双击 `GDB Hardware Debugging`（或者选择 `GDB Hardware Debugging` 然后按下 `New` 按钮）来新建一个配置。
2. 在右边显示的表单中，`Name:` 一栏中输入配置的名称，例如："Blink checking"。
3. 在下面的 `Main` 选项卡中，点击 `Project:` 边上的 `Browse` 按钮，然后选择当前的 `blink` 项目。
4. 在下一行的 `C/C++ Application:` 中，点击 `Browse` 按钮，选择 `blink.elf` 文件。如果 `blink.elf` 文件不存在，那么很有可能该项目还没有编译，请参考 Eclipse Plugin 指南中的介绍。
5. 最后，在 `Build (if required) before launching` 下面点击 `Disable auto build`。

   上述步骤 1 - 5 的示例输入如下图所示。

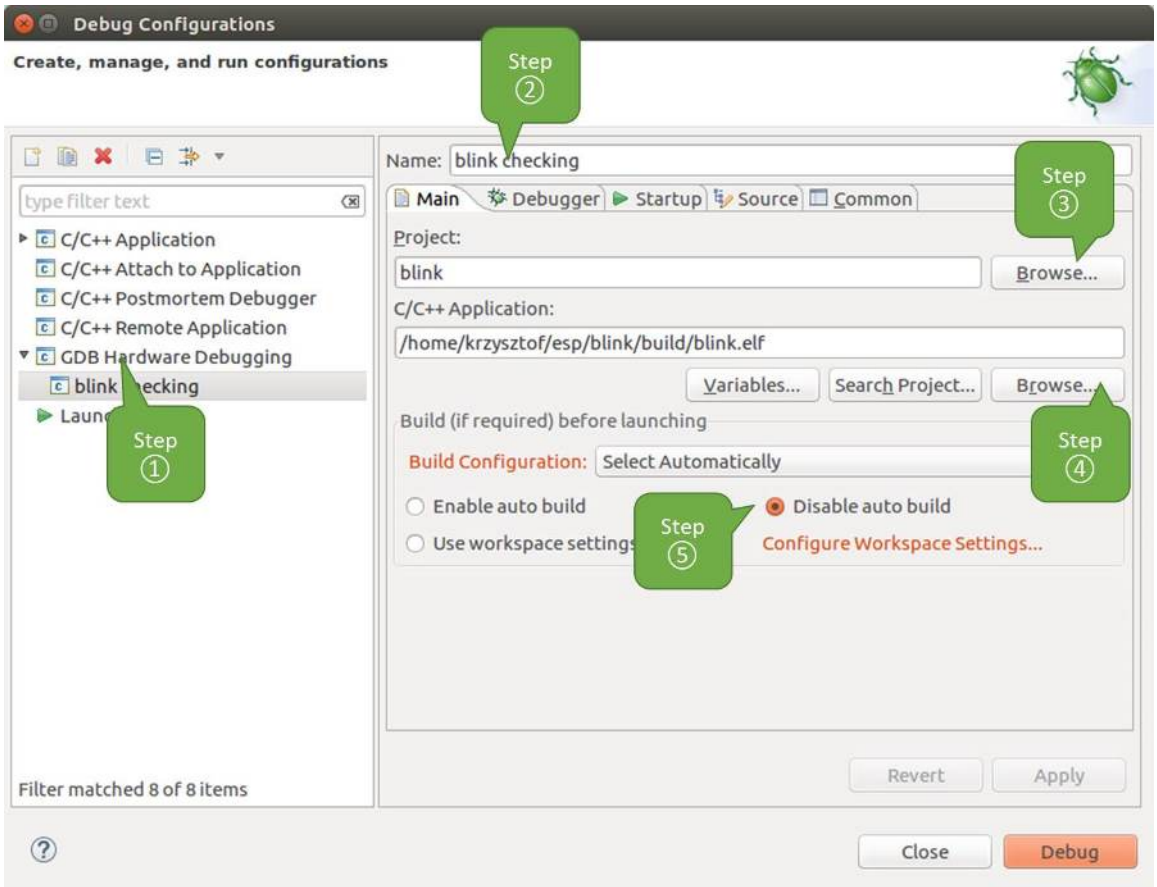


图 11: GDB 硬件调试的配置 - Main 选项卡

6. 点击 `Debugger` 选项卡，在 `GDB Command` 栏中输入 `riscv32-esp-elf-gdb` 来调用调试器。
7. 更改 `Remote host` 的默认配置，在 `Port number` 下面输入 `3333`。

   上述步骤 6 - 7 的示例输入如下图所示。
8. 最后一个需要更改默认配置的选项卡是 `Startup` 选项卡。在 `Initialization Commands` 下，取消选中 `Reset and Delay (seconds)` 和 `Halt`，然后在下面一栏中输入以下命令：

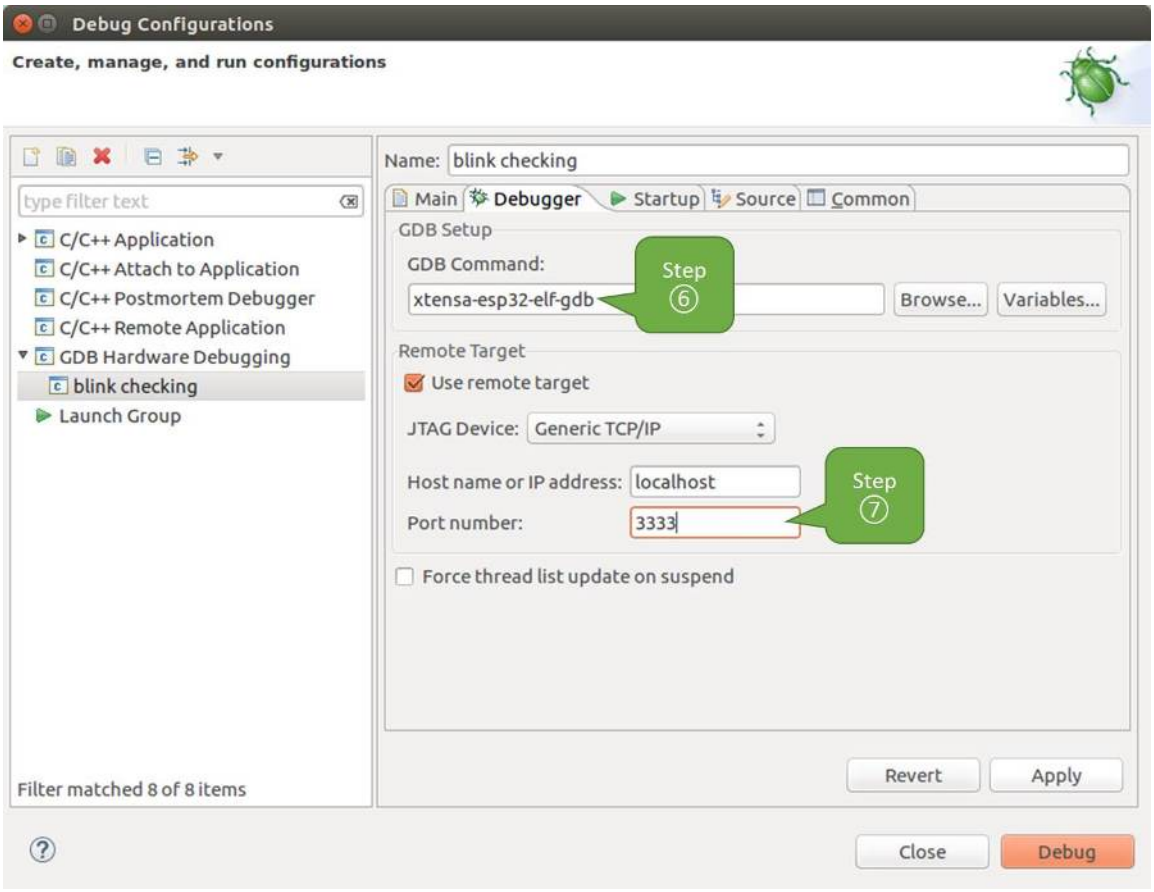


图 12: GDB 硬件调试的配置 - Debugger 选项卡

```
mon reset halt
maintenance flush register-cache
set remote hardware-watchpoint-limit 2
```

备注： 如果想在启动新的调试会话之前自动更新闪存中的镜像，请在 `Initialization Commands` 文本框的开头添加以下命令行:

```
mon reset halt
mon program_esp ${workspace_loc:blink/build/blink.bin} 0x10000 verify
```

有关 `program_esp` 命令的说明请参考*上传待调试的应用程序* 章节。

9. 在 `Load Image and Symbols` 下，取消选中 `Load image` 选项。
10. 在同一个选项卡中继续往下浏览，建立一个初始断点用来在调试器复位后暂停 CPU。插件会根据 `Set break point at:` 一栏中输入的函数名，在该函数的开头设置断点。选中这一选项，并在相应的字段中输入 `app_main`。
11. 选中 `Resume` 选项，这会使得程序在每次调用步骤 8 中的 `mon reset halt` 后恢复，然后在 `app_main` 的断点处停止。
    上述步骤 8 - 11 的示例输入如下图所示。

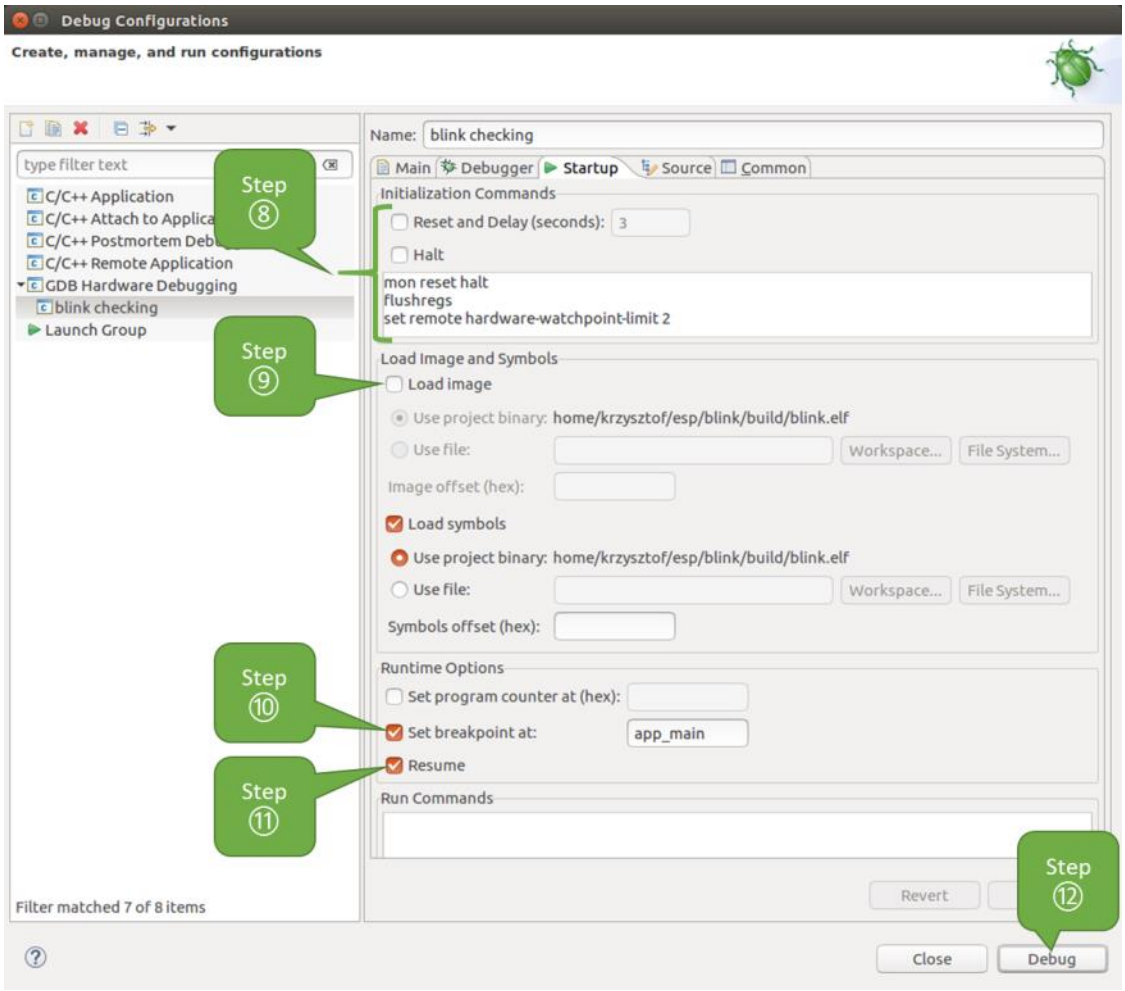


图 13: GDB 硬件调试的配置 - Startup 选项卡

上面的启动序列看起来有些复杂，如果你对其中的初始化命令不太熟悉，请查阅*调试器的启动命令的含义* 章节获取更多说明。

12. 如果前面已经完成*配置 ESP32-P4 目标板* 中介绍的步骤，目标正在运行并准备好与调试器进行对话，那么点击 `Debug` 按钮直接进行调试。如果尚未完成前面步骤，请点击 `Apply` 按钮保存配置，返回*配置 ESP32-P4 目标板* 章节进行配置，最后再回到这里开始调试。

一旦所有 1-12 的配置步骤都已经完成，Eclipse 就会打开 Debug 视图，如下图所示。

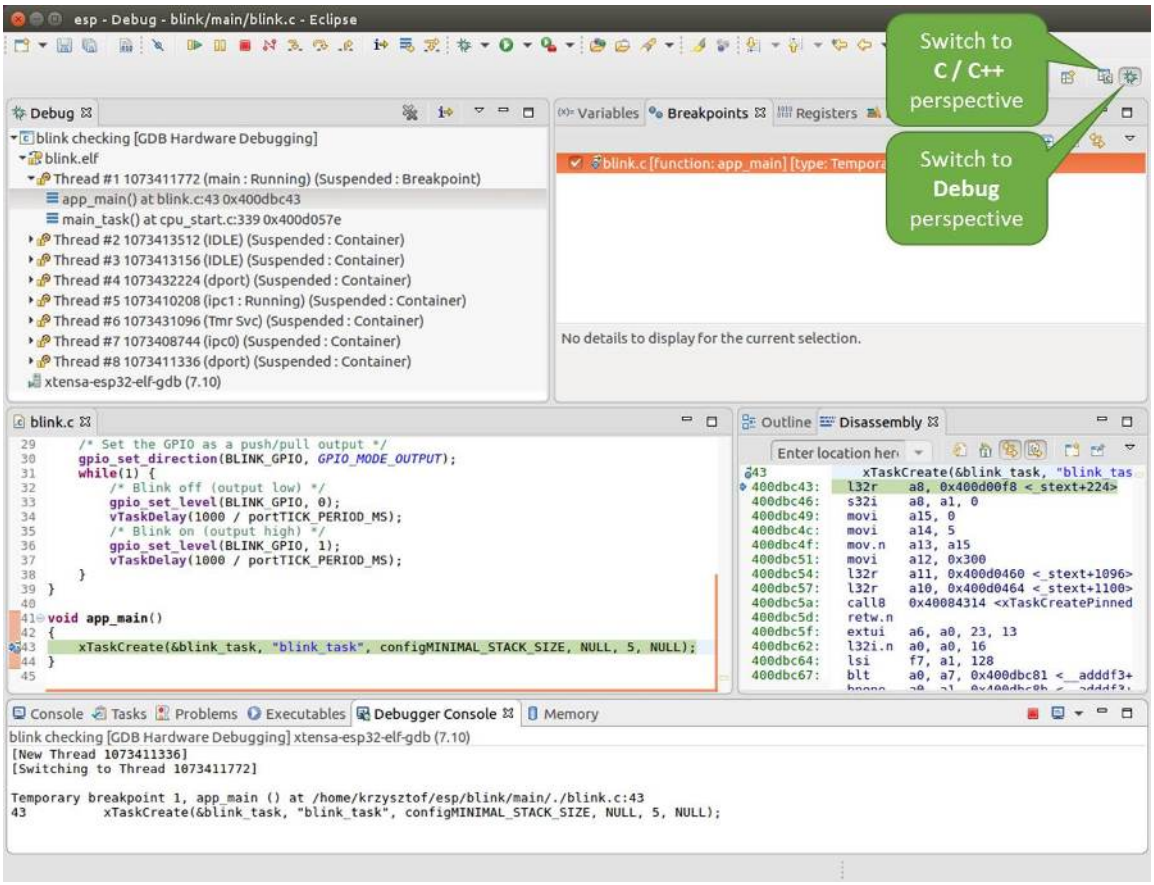


图 14: Eclipse 中的调试视图

如果不太了解 GDB 的常用方法，请查阅使用 *Eclipse* 的调试示例 文章中的调试示例章节调试范例。

使用命令行调试

1. 为了能够启动调试会话，需要先启动并运行目标，如果还没有完成，请按照配置 *ESP32-P4* 目标板 中的介绍进行操作。
2. 打开一个新的终端会话并前往待调试的项目目录，比如：

   ```
   cd ~/esp/blink
   ```

3. 当启动调试器时，通常需要提供几个配置参数和命令，为了避免每次都在命令行中逐行输入这些命令，你可以新建一个配置文件，并将其命名为 gdbinit:

   ```
   target remote :3333
   set remote hardware-watchpoint-limit 2
   mon reset halt
   maintenance flush register-cache
   thb app_main
   c
   ```

   将此文件保存在当前目录中。
   有关 gdbinit 文件内部的更多详细信息，请参阅调试器的启动命令的含义 章节。

4. 准备好启动 GDB，请在终端中输入以下内容：

   ```
   riscv32-esp-elf-gdb -x gdbinit build/blink.elf
   ```

5. 如果前面的步骤已经正确完成，你会看到如下所示的输出日志，在日志的最后会出现 (gdb) 提示符：

```
user-name@computer-name:~/esp/blink$ riscv32-esp-elf-gdb -x gdbinit build/
↪blink.elf
GNU gdb (crosstool-NG crosstool-ng-1.22.0-61-gab8375a) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-build_pc-linux-gnu --target=riscv32-
↪esp-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from build/blink.elf...done.
0x400d10d8 in esp_vApplicationIdleHook () at /home/user-name/esp/esp-idf/
↪components/esp32p4/./freertos_hooks.c:52
52          asm("waiti 0");
JTAG tap: esp32p4.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica),␣
↪part: 0x2003, ver: 0x1)
JTAG tap: esp32p4.slave tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica),␣
↪part: 0x2003, ver: 0x1)
esp32p4: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
esp32p4: Core was reset (pwrstat=0x5F, after clear 0x0F).
esp32p4 halted. PRO_CPU: PC=0x5000004B (active)    APP_CPU: PC=0x00000000
esp32p4: target state: halted
esp32p4: Core was reset (pwrstat=0x1F, after clear 0x0F).
Target halted. PRO_CPU: PC=0x40000400 (active)    APP_CPU: PC=0x40000400
esp32p4: target state: halted
Hardware assisted breakpoint 1 at 0x400db717: file /home/user-name/esp/blink/
↪main/./blink.c, line 43.
0x0:    0x00000000
Target halted. PRO_CPU: PC=0x400DB717 (active)    APP_CPU: PC=0x400D10D8
[New Thread 1073428656]
[New Thread 1073413708]
[New Thread 1073431316]
[New Thread 1073410672]
[New Thread 1073408876]
[New Thread 1073432196]
[New Thread 1073411552]
[Switching to Thread 1073411996]

Temporary breakpoint 1, app_main () at /home/user-name/esp/blink/main/./blink.
↪c:43
43      xTaskCreate(&blink_task, "blink_task", 512, NULL, 5, NULL);
(gdb)
```

注意上面日志的倒数第三行显示了调试器已经在 app_main() 函数的断点处停止，该断点在 gdbinit 文件中设定。由于处理器已经暂停运行，LED 不会再闪烁。如果你的 LED 也停止了闪烁，则可以开始调试。

如果不太了解 GDB 的常用方法，请查阅使用命令行的调试示例 文章中的调试示例章节调试范例。

使用 idf.py 进行调试 你还可以使用 idf.py 更方便地执行上述提到的调试命令，可以使用以下命令：

1. idf.py openocd

在终端中运行 OpenOCD，其配置信息来源于环境变量或者命令行。默认会使用 `OPENOCD_SCRIPTS` 环境变量中指定的脚本路径，它是由 ESP-IDF 项目仓库中的导出脚本（`export.sh` or `export.bat`）添加到系统环境变量中的。当然，你可以在命令行中通过 `--openocd-scripts` 参数来覆盖这个变量的值。
至于当前开发板的 JTAG 配置，请使用环境变量 `OPENOCD_COMMANDS` 或命令行参数 `--openocd-commands`。如果这两者都没有被定义，那么 OpenOCD 会使用 `-f board/esp32p4-builtin.cfg` 参数来启动。

2. `idf.py gdb`
   根据当前项目的 elf 文件自动生成 GDB 启动脚本，然后会按照使用命令行调试 中所描述的步骤启动 GDB。
3. `idf.py gdbtui`
   和步骤 2 相同，但是会在启动 GDB 的时候传递 `tui` 参数，这样可以方便在调试过程中查看源代码。
4. `idf.py gdbgui`
   启动 gdbgui，在浏览器中打开调试器的前端界面。请在运行安装脚本时添加"--enable-gdbgui" 参数，即运行 `install.sh --enable-gdbgui`，从而确保支持 `gdbgui` 选项。
   上述这些命令也可以合并到一起使用，`idf.py` 会自动将后台进程（比如 openocd）最先运行，交互式进程（比如 GDB，monitor）最后运行。
   常用的组合命令如下所示:

```
idf.py openocd gdbgui monitor
```

上述命令会将 OpenOCD 运行至后台，然后启动 gdbgui 打开一个浏览器窗口，显示调试器的前端界面，最后在活动终端打开串口监视器。

调试示例

本节将介绍如何在*Eclipse* 和命令行 中使用 GDB 进行调试的示例。

使用 Eclipse 的调试示例 请检查目标板是否已经准备好，并加载了 get-started/blink 示例代码，然后按照使用 *Eclipse* 调试 中介绍的步骤配置和启动调试器，最后选择让应用程序在 `app_main()` 建立的断点处停止。

本小节的示例

1. 浏览代码，查看堆栈和线程
2. 设置和清除断点
3. 手动暂停目标
4. 单步执行代码
5. 查看并设置内存
6. 观察和设置程序变量
7. 设置条件断点

浏览代码，查看堆栈和线程 当目标暂停时，调试器会在 "Debug" 窗口中显示线程的列表，程序暂停的代码行在下面的另一个窗口中被高亮显示，如下图所示。此时板子上的 LED 停止了闪烁。

暂停的程序所在线程也会被展开，显示函数调用的堆栈，它表示直到目标暂停所在代码行（下图高亮处）为止的相关函数的调用关系。1 号线程下函数调用堆栈的第一行包含了最后一个调用的函数 `app_main()`，根据下一行显示，它又是在函数 `main_task()` 中被调用的。堆栈的每一行还包含调用函数的文件名和行号。通过单击每个堆栈的条目，在下面的窗口中，你将看到此文件的内容。

通过展开线程，你可以浏览整个应用程序。展开 5 号线程，它包含了更长的函数调用堆栈，你可以看到函数调用旁边的数字，比如 `0x4000000c`，它们代表未以源码形式提供的二进制代码所在的内存地址。

无论项目是以源代码还是仅以二进制形式提供，在右边一个窗口中，都可以看到反汇编后的机器代码。

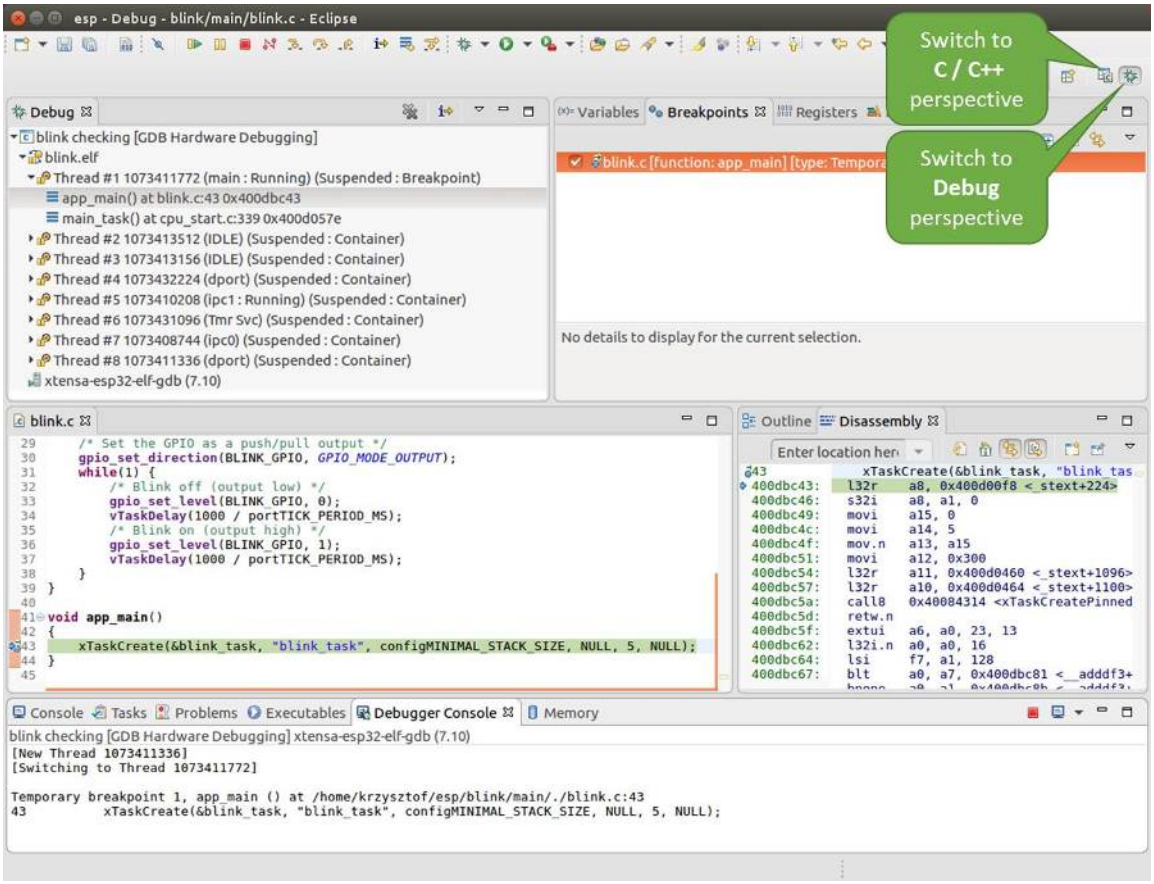


图 15: Eclipse 中的 Debug 视图

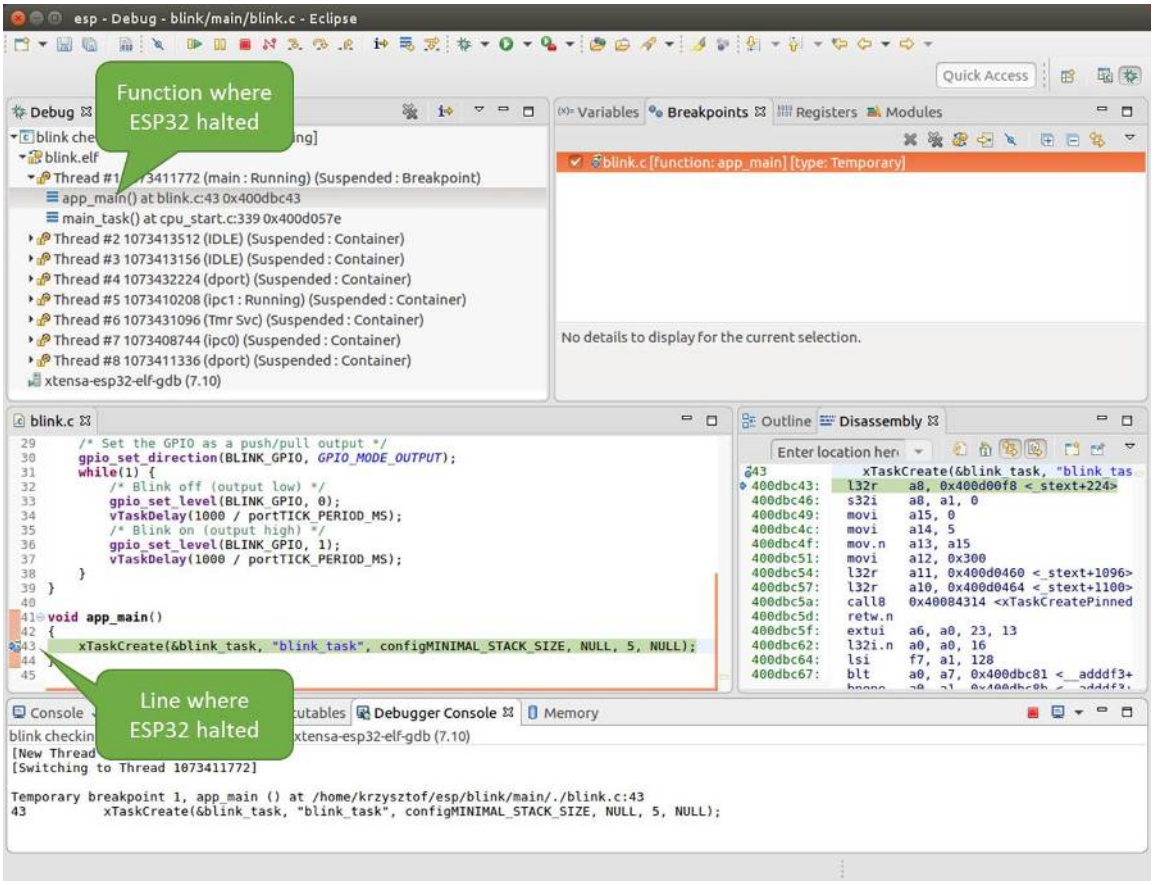

esp - Debug - blink/main/blink.c - Eclipse
Function where ESP32 halted
Line where ESP32 halted
Debug
Variables
Breakpoints
Registers
Modules
app_main() at blink.c:43 0x400dbc43
main_task() at cpu_start.c:339 0x400d057e
Thread #2 1073413512 (IDLE) (Suspended : Container)
Thread #3 1073413156 (IDLE) (Suspended : Container)
Thread #4 1073432224 (dport) (Suspended : Container)
Thread #5 1073410208 (ipc1 : Running) (Suspended : Container)
Thread #6 1073431096 (Tmr Svc) (Suspended : Container)
Thread #7 1073408744 (ipc0) (Suspended : Container)
Thread #8 1073411336 (dport) (Suspended : Container)
xtensa-esp32-elf-gdb (7.10)
blink.c [function: app_main] [type: Temporary]
No details to display for the current selection.
blink.c
Outline
Disassembly
Console
Debugger Console
Memory
[New Thread
[Switching to Thread 1073411772]
Temporary breakpoint 1, app_main () at /home/krzysztof/esp/blink/main/./blink.c:43
43 xTaskCreate(&blink_task, "blink_task", configMINIMAL_STACK_SIZE, NULL, 5, NULL);

图 16: 调试时目标停止

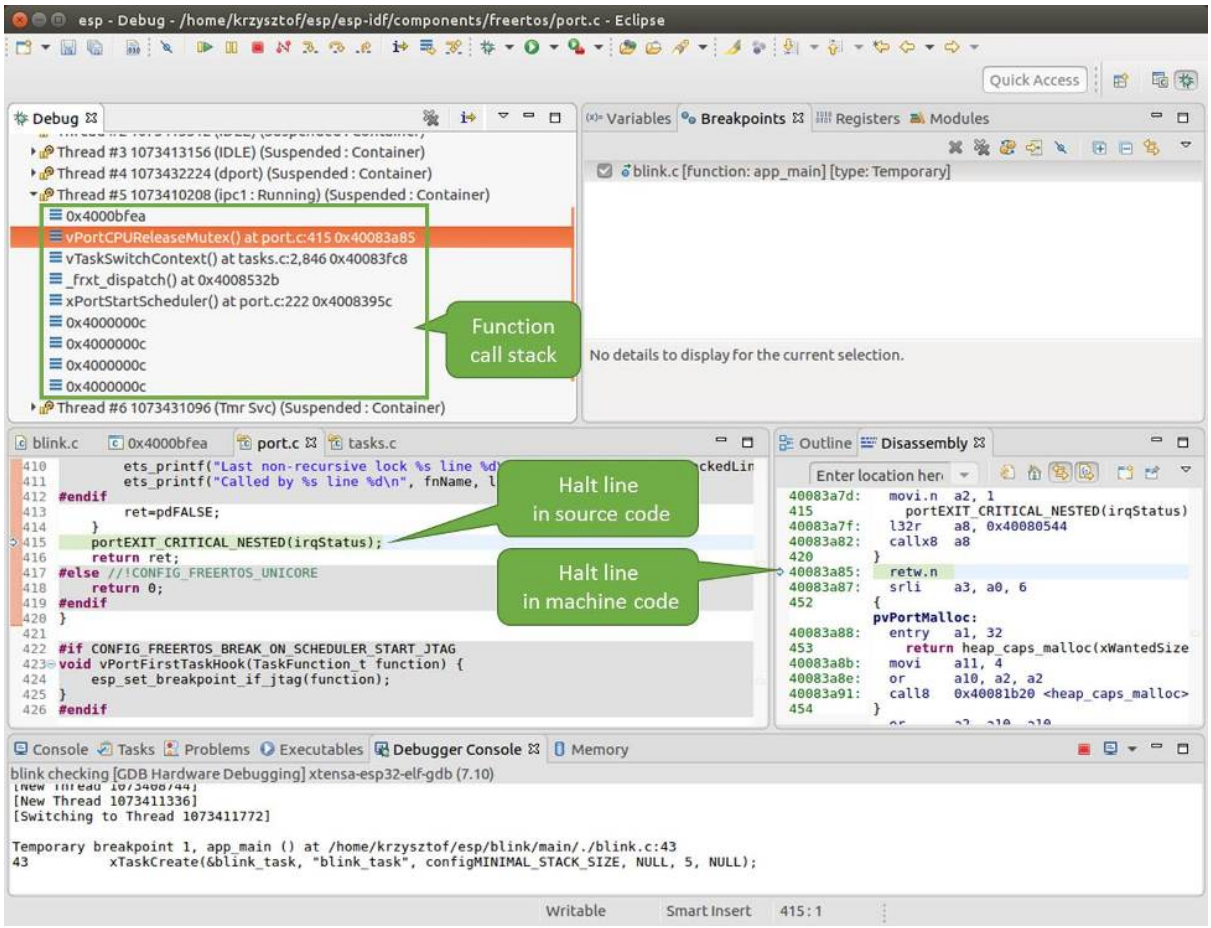


图 17: 浏览函数调用堆栈

回到 1 号线程中的 `app_main()` 函数所在的 `blink.c` 源码文件，下面的示例将会以该文件为例介绍调试的常用功能。调试器可以轻松浏览整个应用程序的代码，这给单步调试代码和设置断点带来了很大的便利，下面将一一展开讨论。

设置和清除断点 在调试时，我们希望能够在关键的代码行停止应用程序，然后检查特定的变量、内存、寄存器和外设的状态。为此我们需要使用断点，以便在特定某行代码处快速访问和停止应用程序。

我们在控制 LED 状态发生变化的两处代码行分别设置一个断点。基于以上代码列表，这两处分别为第 33 和 36 代码行。按住键盘上的“Control”键，双击 blink.c 文件中的行号 33，并在弹出的对话框中点击“OK”按钮进行确定。如果你不想看到此对话框，双击行号即可。执行同样操作，在第 36 行设置另外一个断点。

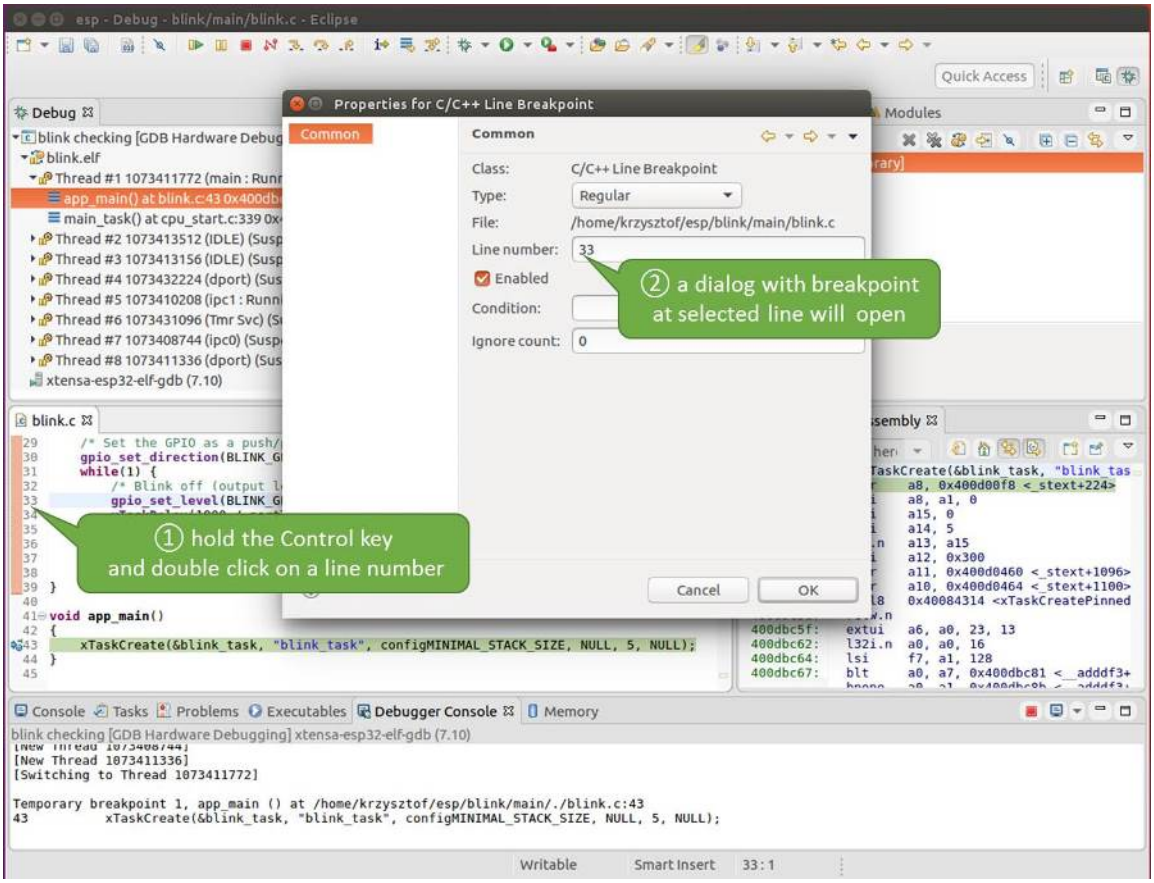


图 18: 设置断点

断点的数量和位置信息会显示在右上角的“断点”窗口中。单击“Show Breakpoints Supported by Selected Target”图标可以刷新此列表。除了刚才设置的两个断点外，列表中可能还包含在调试器启动时设置在 `app_main()` 函数处的临时断点。由于最多只允许设置两个断点（详细信息请参阅可用的断点和观察点），你需要将其删除，否则调试会失败。

单击“Resume”（如果“Resume”按钮是灰色的，请先单击 8 号线程的 `blink_task()` 函数）后处理器将开始继续运行，并在断点处停止。再一次单击“Resume”按钮，使程序再次运行，然后停在第二个断点处，依次类推。

每次单击“Resume”按钮恢复程序运行后，都会看到 LED 切换状态。

更多关于断点的信息，请参阅可用的断点和观察点 和关于断点的补充知识。

手动暂停目标 在调试时，你可以恢复程序运行并输入代码等待某个事件发生或者保持无限循环而不设置任何断点。后者，如果想要返回调试模式，可以通过单击“Suspend”按钮来手动中断程序的运行。

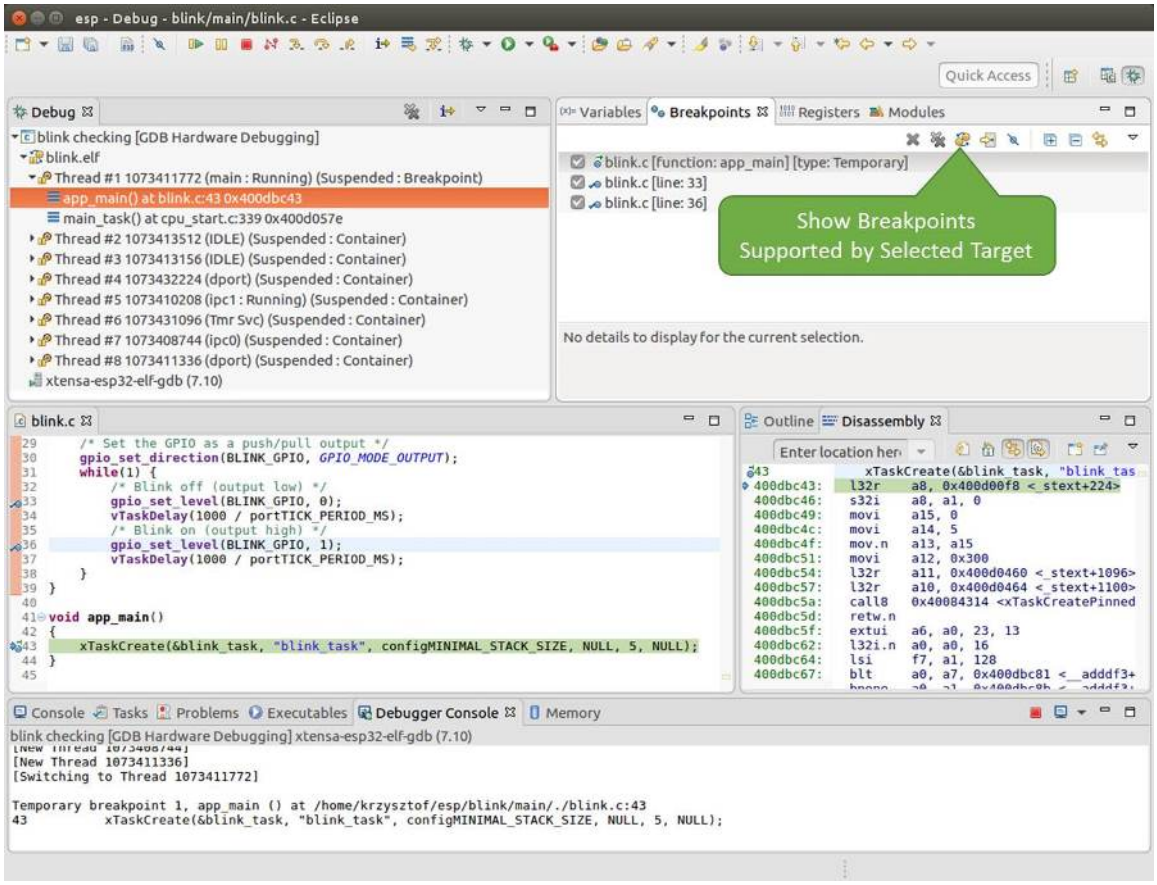


图 19: 设置了三个断点 / 最多允许两个断点

在此之前，请删除所有的断点，然后单击“Resume”按钮。接着单击“Suspend”按钮，应用程序会停止在某个随机的位置，此时 LED 也将停止闪烁。调试器将展开线程并高亮显示停止的代码行。

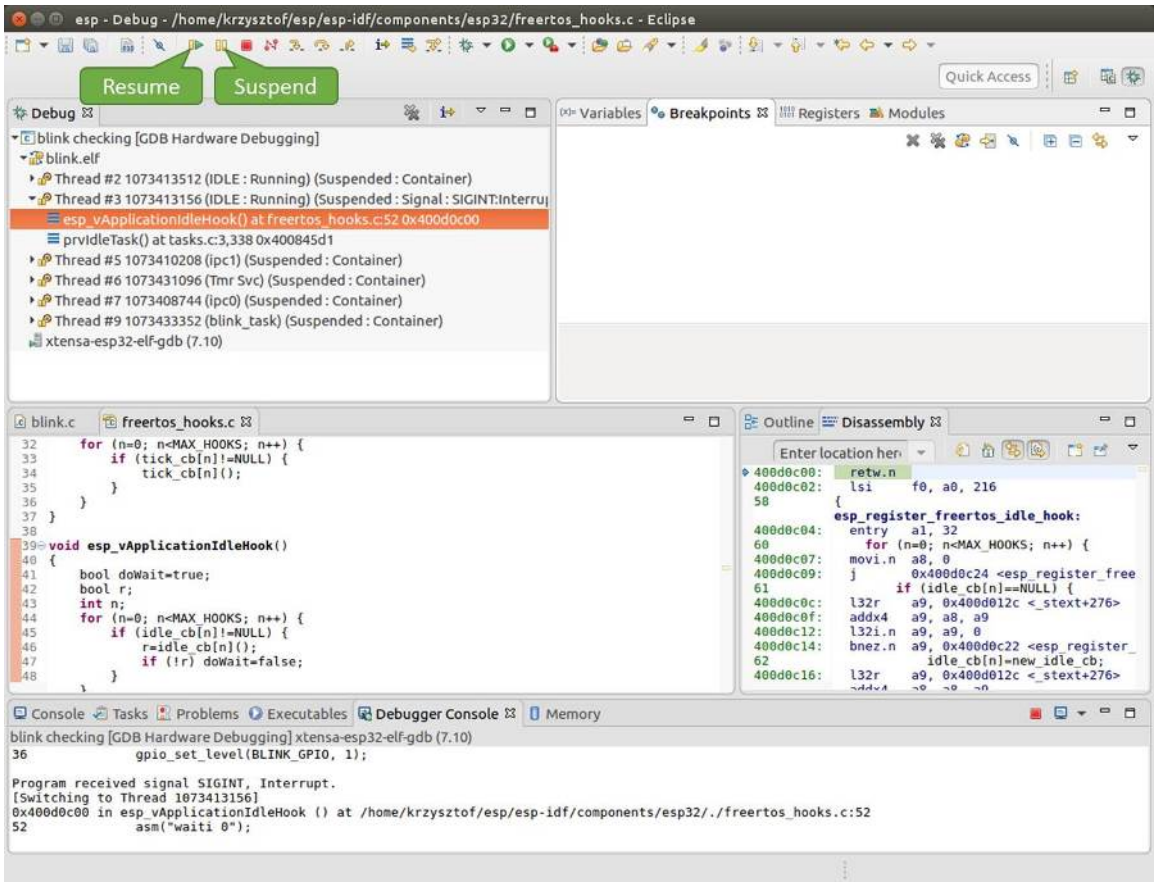


图 20: 手动暂停目标

在上图所示的情况中，应用程序已经在 `freertos_hooks.c` 文件的第 52 行暂停运行，现在你可以通过单击“Resume”按钮再次将其恢复运行或者进行下面要介绍的调试工作。

单步执行代码 我们还可以使用“Step Into (F5)”和“Step Over (F6)”命令单步执行代码，这两者之间的区别是执行“Step Into (F5)”命令会进入调用的子程序，而执行“Step Over (F6)”命令则会直接将子程序看成单个源码行，单步就能将其运行结束。

在继续演示此功能之前，请参照上文所述确保目前只在 `blink.c` 文件的第 36 行设置了一个断点。

按下 F8 键让程序继续运行然后在断点处停止运行，多次按下“Step Over (F6)”按钮，观察调试器是如何单步执行一行代码的。

如果你改用”Step Into (F5)”，那么调试器将会进入调用的子程序内部。

在上述例子中，调试器进入 `gpio_set_level(BLINK_GPIO, 0)` 代码内部，同时代码窗口快速切换到 `gpio.c` 驱动文件。

请参阅 *“next”命令无法跳过子程序的原因* 文档以了解 `next` 命令的潜在局限。

查看并设置内存 要显示或者设置内存的内容，请使用“调试”视图中位于底部的“Memory”选项卡。

在“Memory”选项卡下，我们将在内存地址 `0x3FF44004` 处读取和写入内容。该地址也是 `GPIO_OUT_REG` 寄存器的地址，可以用来控制（设置或者清除）某个 GPIO 的电平。

关于该寄存器的更多详细信息，请参阅 *ESP32-P4 技术参考手册* > *IO MUX 和 GPIO Matrix (GPIO, IO_MUX)* [PDF] 章节。

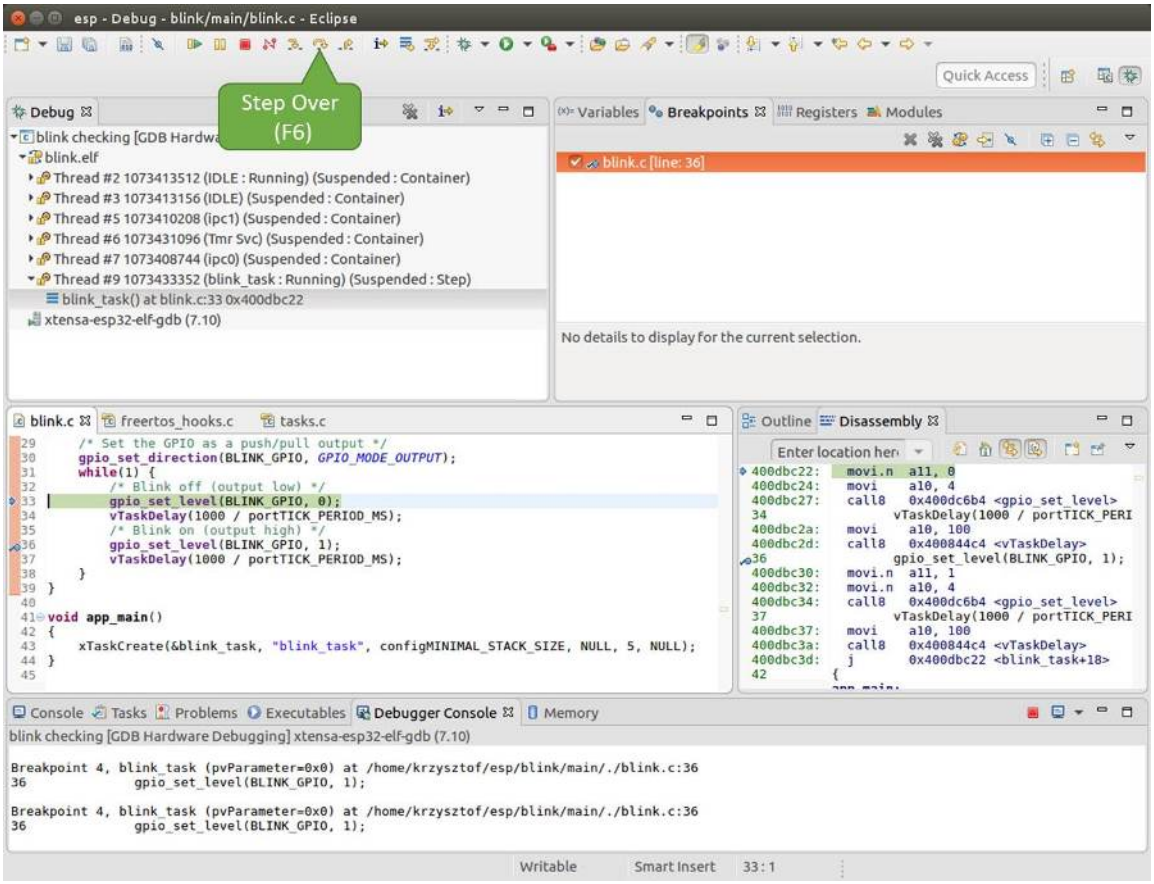


图 21: 使用“Step Over (F6)”单步执行代码

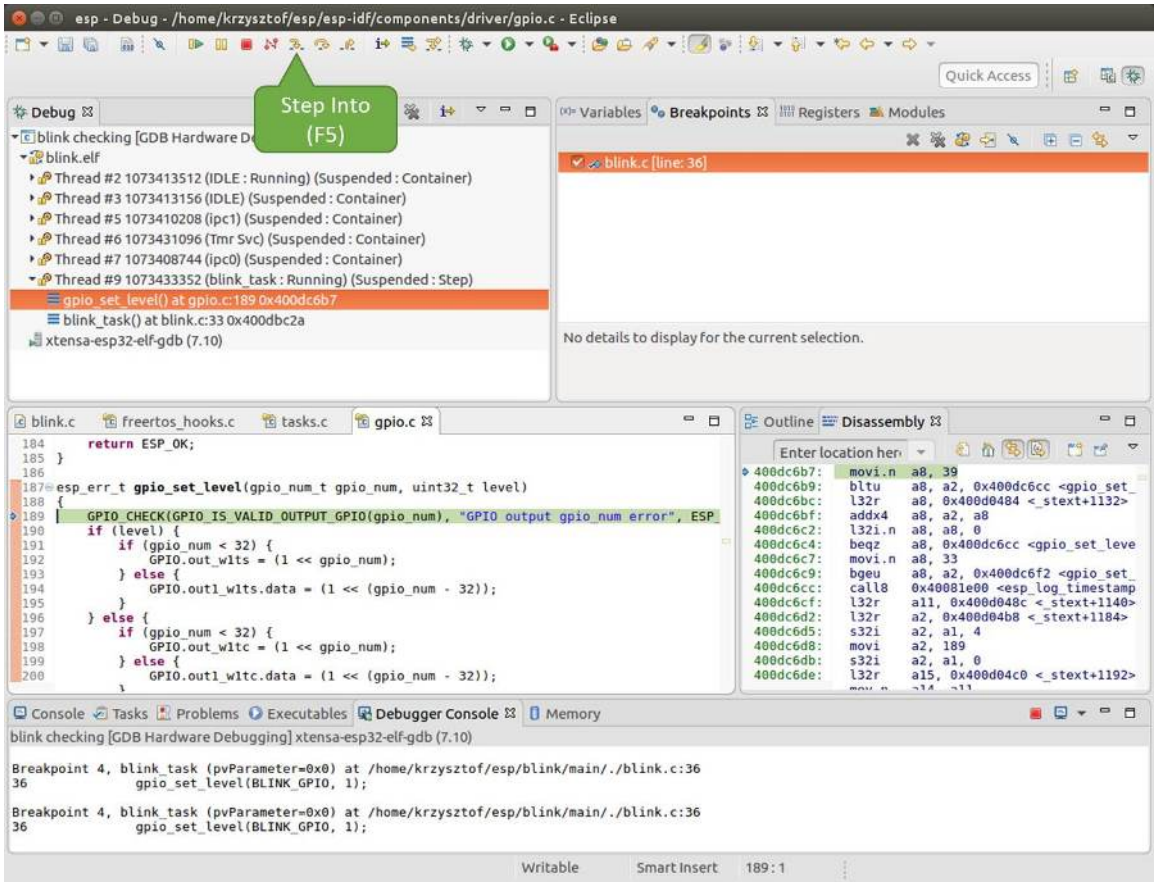


图 22: 使用“Step Into (F5)”单步执行代码

同样在 `blink.c` 项目文件中，在两个 `gpio_set_level` 语句的后面各设置一个断点，单击“Memory”选项卡，然后单击“Add Memory Monitor”按钮，在弹出的对话框中输入 `0x3FF44004`。

按下 F8 按键恢复程序运行，并观察“Monitor”选项卡。

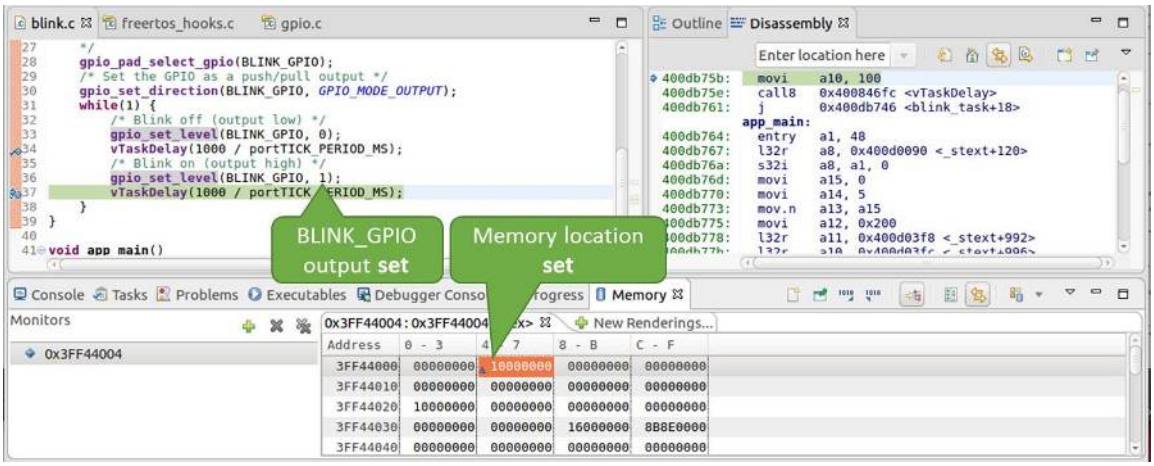


图 23: 观察内存地址 0x3FF44004 处的某个比特被置高

每按一下 F8，你就会看到在内存 `0x3FF44004` 地址处的一个比特位被翻转（并且 LED 会改变状态）。

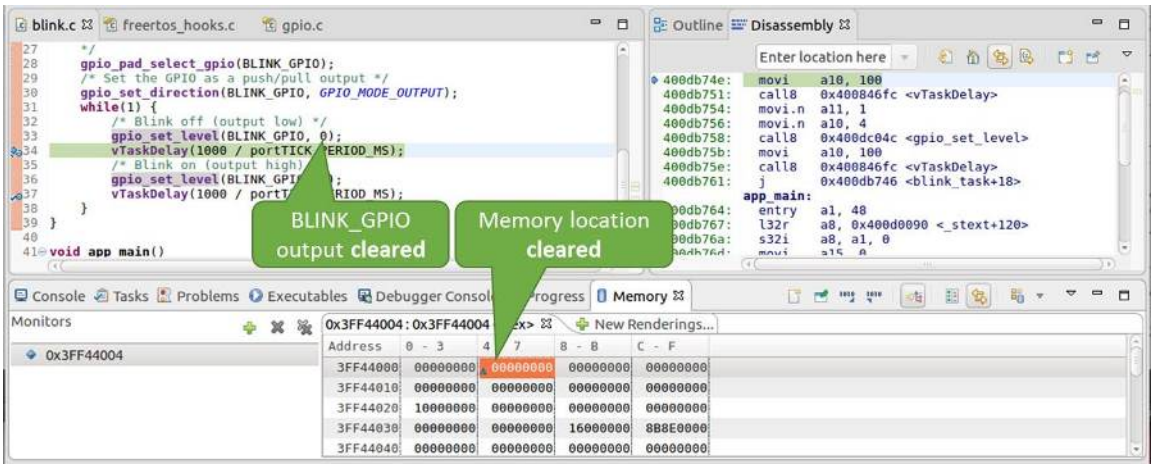


图 24: 观察内存地址 0x3FF44004 处的某个比特被置低

要修改内存的数值，请在“Monitor”选项卡中找到待修改的内存地址，如前面观察的结果一样，输入特定比特翻转后的值。当按下回车键后，将立即看到 LED 的状态发生了改变。

观察和设置程序变量 常见的调试任务是在程序运行期间检查程序中某个变量的值，为了演示这个功能，更新 `blink.c` 文件，在 `blink_task` 函数的上面添加一个全局变量的声明 `int i`，然后在 `while(1)` 里添加 `i++`，这样每次 LED 改变状态的时候，变量 `i` 都会增加 1。

退出调试器，这样就不会与新代码混淆，然后重新构建并烧写代码到 ESP32-P4 中，接着重启调试器。注意，这里不需要我们重启 OpenOCD。

一旦程序停止运行，在代码 `i++` 处添加一个断点。

下一步，在“Breakpoints”所在的窗口中，选择“Expressions”选项卡。如果该选项卡不存在，请在顶部菜单栏的 Window > Show View > Expressions 中添加这一选项卡。然后在该选项卡中单击“Add new expression”，并输入 `i`。

按下 F8 继续运行程序，每次程序停止时，都会看到变量 `i` 的值在递增。

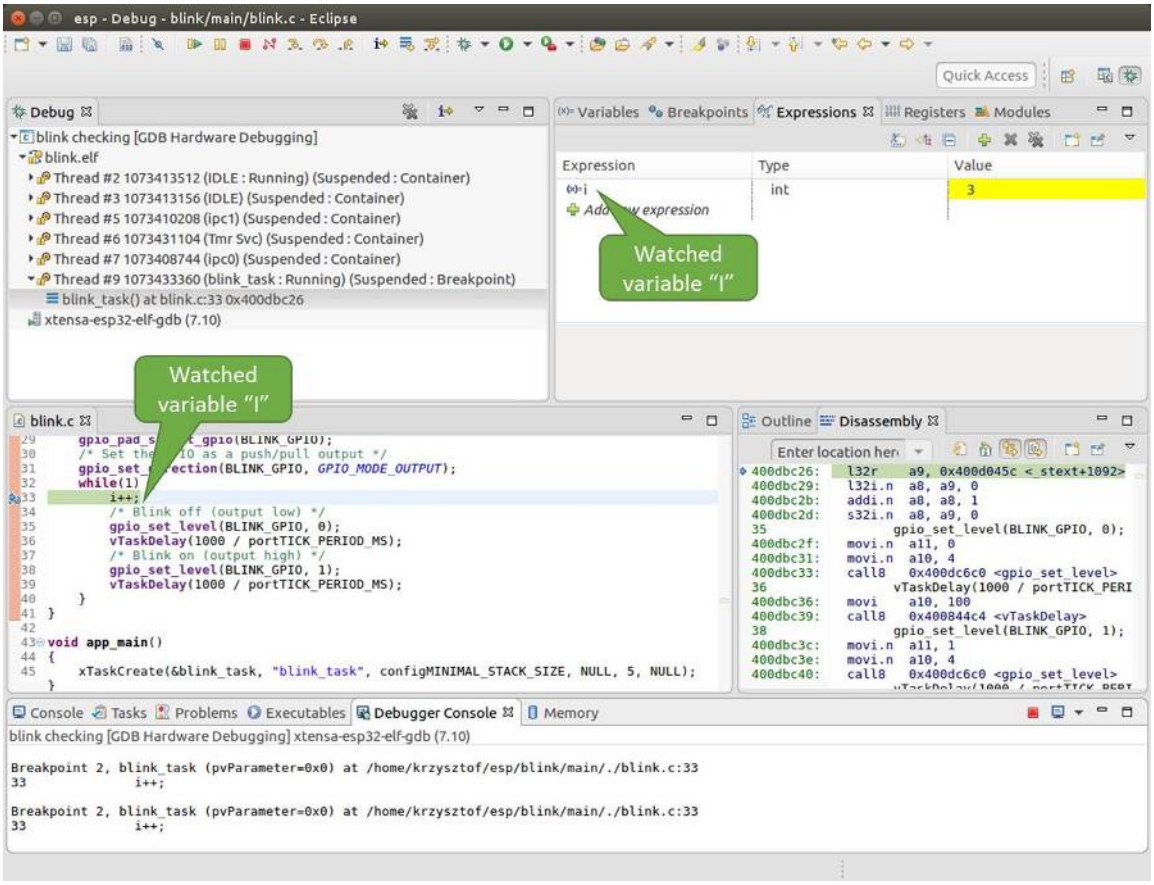

esp - Debug - blink/main/blink.c - Eclipse
Quick Access
Debug
blink checking [GDB Hardware Debugging]
blink.elf
Thread #2 1073413512 (IDLE : Running) (Suspended : Container)
Thread #3 1073413156 (IDLE) (Suspended : Container)
Thread #5 1073410208 (ipc1) (Suspended : Container)
Thread #6 1073431104 (Tmr Svc) (Suspended : Container)
Thread #7 1073408744 (ipc0) (Suspended : Container)
Thread #9 1073433360 (blink_task : Running) (Suspended : Breakpoint)
blink_task() at blink.c:33 0x400dbc26
xtensa-esp32-elf-gdb (7.10)
Variables
Breakpoints
Expressions
Registers
Modules
Expression
Type
Value
i
int
3
Watched variable "i"
Watched variable "i"
blink.c
Outline
Disassembly
Console
Tasks
Problems
Executables
Debugger Console
Memory
blink checking [GDB Hardware Debugging] xtensa-esp32-elf-gdb (7.10)
Breakpoint 2, blink_task (pvParameter=0x0) at /home/krzysztof/esp/blink/main/./blink.c:33
33 i++;
Breakpoint 2, blink_task (pvParameter=0x0) at /home/krzysztof/esp/blink/main/./blink.c:33
33 i++;

图 25: 观察程序变量 "i"

如想更改 `i` 的值，可以在“Value”一栏中输入新的数值。按下“Resume (F8)”后，程序将从新输入的数字开始递增 `i`。

设置条件断点 接下来的内容更为有趣，你可能想在一定条件满足的情况下设置断点，然后让程序停止运行。右击断点打开上下文菜单，选择“Breakpoint Properties”，将“Type:”改选为“Hardware”然后在“Condition:”一栏中输入条件表达式，例如 `i == 2`。

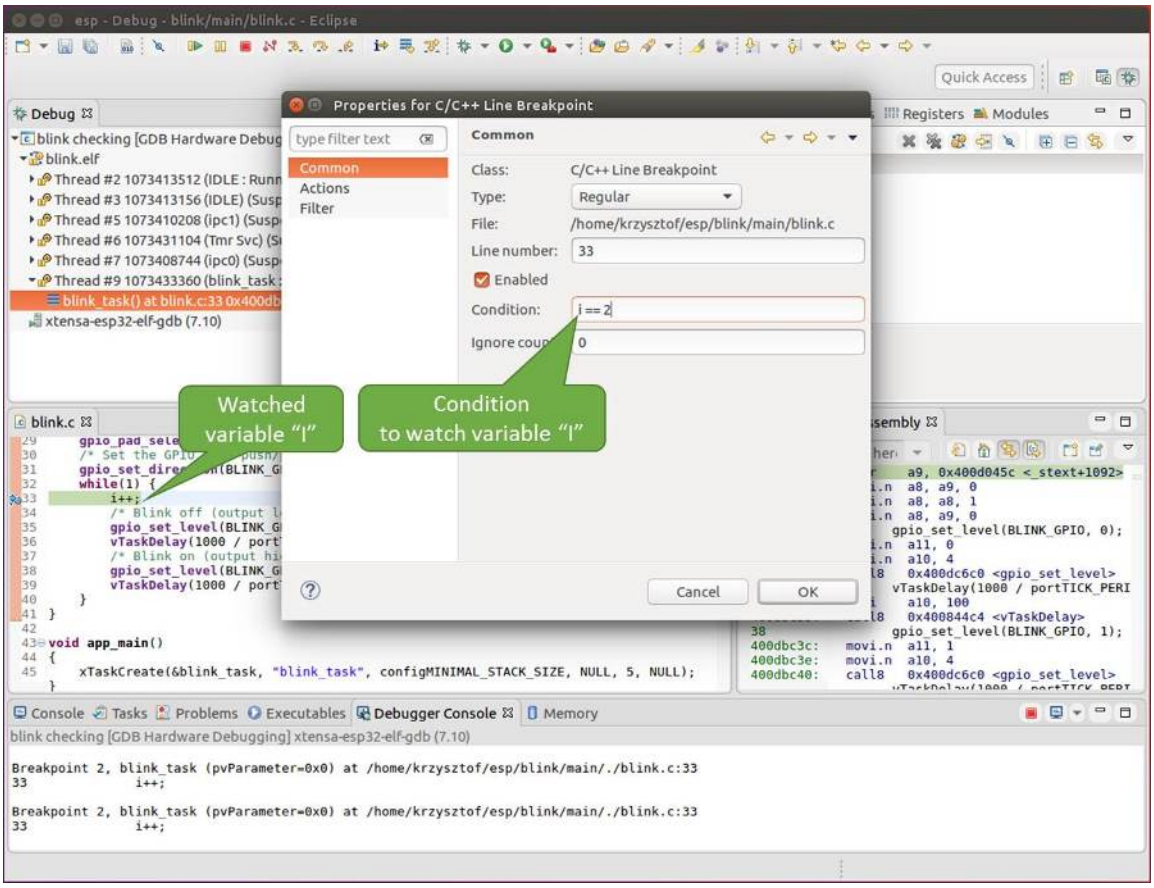


图 26: 设置条件断点

如果当前 `i` 的值小于 2（如果有需要也可以更改这个阈值）并且程序被恢复运行，那么 LED 就会循环闪烁，直到 `i == 2` 条件成立，最后程序停止在该处。

使用命令行的调试示例 请检查你的目标板是否已经准备好，并加载了 get-started/blink 示例代码，然后按照*使用命令行调试* 中介绍的步骤配置和启动调试器，最后选择让应用程序在 `app_main()` 建立的断点处停止运行

```
Temporary breakpoint 1, app_main () at /home/user-name/esp/blink/main/./blink.c:43
43      xTaskCreate(&blink_task, "blink_task", configMINIMAL_STACK_SIZE, NULL, 5,␣
↪NULL);
(gdb)
```

本小节的示例

1. *浏览代码，查看堆栈和线程*
2. *设置和清除断点*
3. *暂停和恢复应用程序的运行*
4. *单步执行代码*
5. *查看并设置内存*

6. 观察和设置程序变量
7. 设置条件断点
8. 调试 *FreeRTOS* 对象

浏览代码，查看堆栈和线程 当看到 (gdb) 提示符的时候，应用程序已停止运行，LED 也停止闪烁。

要找到代码暂停的位置，输入 `l` 或者 `list` 命令，调试器会打印出暂停点（`blink.c` 代码文件的第 43 行）附近的几行代码

```
(gdb) l
38      }
39  }
40
41  void app_main()
42  {
43      xTaskCreate(&blink_task, "blink_task", configMINIMAL_STACK_SIZE, NULL, 5,␣
↪NULL);
44  }
(gdb)
```

也可以通过输入 `l 30, 40` 等命令来查看特定行号范围内的代码。

使用 `bt` 或者 `backtrace` 来查看哪些函数最终导致了此代码被调用:

```
(gdb) bt
#0  app_main () at /home/user-name/esp/blink/main/./blink.c:43
#1  0x400d057e in main_task (args=0x0) at /home/user-name/esp/esp-idf/components/
↪esp32p4/./cpu_start.c:339
(gdb)
```

输出的第 0 行表示应用程序暂停之前调用的最后一个函数，即我们之前列出的 `app_main ()`。`app_main ()` 又被位于 `cpu_start.c` 文件第 339 行的 `main_task` 函数调用。

想查看 `cpu_start.c` 文件中 `main_task` 函数的上下文，需要输入 `frame N`，其中 N = 1，因为根据前面的输出，`main_task` 位于 #1 下:

```
(gdb) frame 1
#1  0x400d057e in main_task (args=0x0) at /home/user-name/esp/esp-idf/components/
↪esp32p4/./cpu_start.c:339
339     app_main();
(gdb)
```

输入 `l` 将显示一段名为 `app_main()` 的代码（在第 339 行）:

```
(gdb) l
334         ;
335     }
336 #endif
337     //Enable allocation in region where the startup stacks were located.
338     heap_caps_enable_nonos_stack_heaps();
339     app_main();
340     vTaskDelete(NULL);
341 }
342
(gdb)
```

通过打印前面的一些行，你会看到我们一直在寻找的 `main_task` 函数:

```
(gdb) l 326, 341
326 static void main_task(void* args)
327 {
328     // Now that the application is about to start, disable boot watchdogs
```

(下页继续)

(续上页)

```
329     REG_CLR_BIT(TIMG_WDTCONFIG0_REG(0), TIMG_WDT_FLASHBOOT_MOD_EN_S);
330     REG_CLR_BIT(RTC_CNTL_WDTCONFIG0_REG, RTC_CNTL_WDT_FLASHBOOT_MOD_EN);
331 #if !CONFIG_FREERTOS_UNICORE
332     // Wait for FreeRTOS initialization to finish on APP CPU, before replacing␣
↪its startup stack
333     while (port_xSchedulerRunning[1] == 0) {
334         ;
335     }
336 #endif
337     //Enable allocation in region where the startup stacks were located.
338     heap_caps_enable_nonos_stack_heaps();
339     app_main();
340     vTaskDelete(NULL);
341 }
(gdb)
```

如果要查看其他代码，可以输入 `i threads` 命令，则会输出目标板上运行的线程列表:

```
(gdb) i threads
  Id   Target Id         Frame
  8    Thread 1073411336 (dport) 0x400d0848 in dport_access_init_core (arg=
↪<optimized out>)
    at /home/user-name/esp/esp-idf/components/esp32p4/./dport_access.c:170
  7    Thread 1073408744 (ipc0) xQueueGenericReceive (xQueue=0x3ffae694,␣
↪pvBuffer=0x0, xTicksToWait=1644638200,
    xJustPeeking=0) at /home/user-name/esp/esp-idf/components/freertos/./queue.
↪c:1452
  6    Thread 1073431096 (Tmr Svc) prvTimerTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos/./timers.c:445
  5    Thread 1073410208 (ipc1 : Running) 0x4000bfea in ?? ()
  4    Thread 1073432224 (dport) dport_access_init_core (arg=0x0)
    at /home/user-name/esp/esp-idf/components/esp32p4/./dport_access.c:150
  3    Thread 1073413156 (IDLE) prvIdleTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos/./tasks.c:3282
  2    Thread 1073413512 (IDLE) prvIdleTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos/./tasks.c:3282
* 1    Thread 1073411772 (main : Running) app_main () at /home/user-name/esp/blink/
↪main/./blink.c:43
(gdb)
```

线程列表显示了每个线程最后一个被调用的函数以及所在的 C 源文件名（如果存在的话）。

你可以通过输入 `thread N` 进入特定的线程，其中 `N` 是线程 ID。我们进入 5 号线程来看一下它是如何工作的:

```
(gdb) thread 5
[Switching to thread 5 (Thread 1073410208)]
#0  0x4000bfea in ?? ()
(gdb)
```

然后查看回溯:

```
(gdb) bt
#0  0x4000bfea in ?? ()
#1  0x40083a85 in vPortCPUReleaseMutex (mux=<optimized out>) at /home/user-name/
↪esp/esp-idf/components/freertos/./port.c:415
#2  0x40083fc8 in vTaskSwitchContext () at /home/user-name/esp/esp-idf/components/
↪freertos/./tasks.c:2846
#3  0x4008532b in _frxt_dispatch ()
#4  0x4008395c in xPortStartScheduler () at /home/user-name/esp/esp-idf/components/
↪freertos/./port.c:222
```

(下页继续)

(续上页)

```
#5  0x4000000c in ?? ()
#6  0x4000000c in ?? ()
#7  0x4000000c in ?? ()
#8  0x4000000c in ?? ()
(gdb)
```

如上所示，回溯可能会包含多个条目，方便查看直至目标停止运行的函数调用顺序。如果找不到某个函数的源码文件，将会使用问号 `??` 替代，这表示该函数是以二进制格式提供的。像 `0x4000bfea` 这样的值是被调用函数所在的内存地址。

使用诸如 `bt`，`i threads`，`thread N` 和 `list` 命令可以浏览整个应用程序的代码。这给单步调试代码和设置断点带来很大的便利，下面将一一展开来讨论。

设置和清除断点 在调试时，我们希望能够在关键的代码行停止应用程序，然后检查特定的变量、内存、寄存器和外设的状态。为此我们需要使用断点，以便在特定某行代码处快速访问和停止应用程序。

我们在控制 LED 状态发生变化的两处代码行分别设置一个断点。基于以上代码列表，这两处分别为第 33 和 36 代码行。使用命令 `break M` 设置断点，其中 M 是具体的代码行:

```
(gdb) break 33
Breakpoint 2 at 0x400db6f6: file /home/user-name/esp/blink/main/./blink.c, line 33.
(gdb) break 36
Breakpoint 3 at 0x400db704: file /home/user-name/esp/blink/main/./blink.c, line 36.
```

输入命令 `c`，处理器将运行并在断点处停止。再次输入 `c` 将使其再次运行，并在第二个断点处停止，依此类推:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB6F6 (active)    APP_CPU: PC=0x400D10D8

Breakpoint 2, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↪blink.c:33
33          gpio_set_level(BLINK_GPIO, 0);
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB6F8 (active)    APP_CPU: PC=0x400D10D8
Target halted. PRO_CPU: PC=0x400DB704 (active)    APP_CPU: PC=0x400D10D8

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↪blink.c:36
36          gpio_set_level(BLINK_GPIO, 1);
(gdb)
```

只有在输入命令 `c` 恢复程序运行后才能看到 LED 改变状态。

查看已设置断点的数量和位置，请使用命令 `info break`:

```
(gdb) info break
Num     Type           Disp Enb Address    What
2       breakpoint     keep y   0x400db6f6 in blink_task at /home/user-name/esp/
↪blink/main/./blink.c:33
    breakpoint already hit 1 time
3       breakpoint     keep y   0x400db704 in blink_task at /home/user-name/esp/
↪blink/main/./blink.c:36
    breakpoint already hit 1 time
(gdb)
```

请注意，断点序号（在 `Num` 栏列出）从 2 开始，这是因为在调试器启动时执行 `thb app_main` 命令已经在 `app_main()` 函数处建立了第一个断点。由于它是一个临时断点，已经被自动删除，所以没有被列出。

要删除一个断点，请输入 delete N 命令（或者简写成 d N），其中 N 代表断点序号:

```
(gdb) delete 1
No breakpoint number 1.
(gdb) delete 2
(gdb)
```

更多关于断点的信息，请参阅可用的断点和观察点 和关于断点的补充知识。

暂停和恢复应用程序的运行 在调试时，可以恢复程序运行并输入代码等待某个事件发生或者保持无限循环而不设置任何断点。对于后者，想要返回调试模式，可以通过输入 Ctrl+C 手动中断程序的运行。

在此之前，请删除所有的断点，然后输入 c 恢复程序运行。接着输入 Ctrl+C，应用程序会停止在某个随机的位置，此时 LED 也将停止闪烁。调试器会打印如下信息:

```
(gdb) c
Continuing.
^CTarget halted. PRO_CPU: PC=0x400D0C00             APP_CPU: PC=0x400D0C00 (active)
[New Thread 1073433352]

Program received signal SIGINT, Interrupt.
[Switching to Thread 1073413512]
0x400d0c00 in esp_vApplicationIdleHook () at /home/user-name/esp/esp-idf/
↪components/esp32p4/./freertos_hooks.c:52
52          asm("waiti 0");
(gdb)
```

在上图所示的情况下，应用程序已经在 freertos_hooks.c 文件的第 52 行暂停运行，现在你可以通过输入 c 再次将其恢复运行或者进行如下所述的一些调试工作。

单步执行代码 我们还可以使用 step 和 next 命令（可以简写成 s 和 n）单步执行代码，这两者之间的区别是执行 “step” 命令会进入调用的子程序内部，而执行 “next” 命令则会直接将子程序看成单个源码行，单步就能将其运行结束。

在继续演示此功能之前，请使用前面介绍的 break 和 delete 命令，确保目前只在 blink.c 文件的第 36 行设置了一个断点:

```
(gdb) info break
Num     Type           Disp Enb Address    What
3       breakpoint     keep y   0x400db704 in blink_task at /home/user-name/esp/
↪blink/main/./blink.c:36
    breakpoint already hit 1 time
(gdb)
```

输入 c 恢复程序运行然后等它在断点处停止运行:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB754 (active)    APP_CPU: PC=0x400D1128

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↪blink.c:36
36          gpio_set_level(BLINK_GPIO, 1);
(gdb)
```

然后输入 n 多次，观察调试器是如何单步执行一行代码的:

```
(gdb) n
Target halted. PRO_CPU: PC=0x400DB756 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB758 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DC04C (active)    APP_CPU: PC=0x400D1128
```

(下页继续)

(续上页)

```
Target halted. PRO_CPU: PC=0x400DB75B (active)    APP_CPU: PC=0x400D1128
37          vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) n
Target halted. PRO_CPU: PC=0x400DB75E (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400846FC (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB761 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB746 (active)    APP_CPU: PC=0x400D1128
33          gpio_set_level(BLINK_GPIO, 0);
(gdb)
```

如果你输入 s，那么调试器将进入子程序:

```
(gdb) s
Target halted. PRO_CPU: PC=0x400DB748 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB74B (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DC04C (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DC04F (active)    APP_CPU: PC=0x400D1128
gpio_set_level (gpio_num=GPIO_NUM_4, level=0) at /home/user-name/esp/esp-idf/
↪components/esp_driver_gpio/src/gpio.c:183
183     GPIO_CHECK(GPIO_IS_VALID_OUTPUT_GPIO(gpio_num), "GPIO output gpio_num error
↪", ESP_ERR_INVALID_ARG);
(gdb)
```

上述例子中，调试器进入 gpio_set_level(BLINK_GPIO, 0) 代码内部，同时代码窗口快速切换到 gpio.c 驱动文件。

请参阅 *"next" 命令无法跳过子程序的原因* 文档以了解 next 命令的潜在局限。

查看并设置内存 使用命令 x 可以显示内存的内容，配合其余参数还可以调整所显示内存位置的格式和数量。运行 help x 可以查看更多相关细节。与 x 命令配合使用的命令是 set，它允许你将值写入内存。

为了演示 x 和 set 的使用，我们将在内存地址 0x3FF44004 处读取和写入内容。该地址也是 GPIO_OUT_REG 寄存器的地址，可以用来控制（设置或者清除）某个 GPIO 的电平。

关于该寄存器的更多详细信息，请参阅 *ESP32-P4 技术参考手册 > IO MUX 和 GPIO Matrix (GPIO, IO_MUX)* [PDF] 章节。

同样在 blink.c 项目文件中，在两个 gpio_set_level 语句的后面各设置一个断点。输入两次 c 命令后停止在断点处，然后输入 x /1wx 0x3FF44004 来显示 GPIO_OUT_REG 寄存器的值:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB75E (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB74E (active)    APP_CPU: PC=0x400D1128

Breakpoint 2, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↪blink.c:34
34          vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000000
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB751 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB75B (active)    APP_CPU: PC=0x400D1128

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↪blink.c:37
37          vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000010
(gdb)
```

如果闪烁的 LED 连接到了 GPIO4，那么每次 LED 改变状态时你会看到第 4 比特被翻转:

```
0x3ff44004: 0x00000000
...
0x3ff44004: 0x00000010
```

现在，当 LED 熄灭时，与之对应地会显示 `0x3ff44004: 0x00000000`，尝试使用 `set` 命令向相同的内存地址写入 `0x00000010` 来将该比特置高:

```
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000000
(gdb) set {unsigned int}0x3FF44004=0x000010
```

在输入 `set {unsigned int}0x3FF44004=0x000010` 命令后，你会立即看到 LED 亮起。

观察和设置程序变量 常见的调试任务是在程序运行期间检查程序中某个变量的值，为了能够演示这个功能，更新 `blink.c` 文件，在 `blink_task` 函数的上面添加一个全局变量的声明 `int i`，然后在 `while(1)` 里添加 `i++`，这样每次 LED 改变状态的时候，变量 `i` 都会增加 1。

退出调试器，这样就不会与新代码混淆，然后重新构建并烧写代码到 ESP32-P4 中，接着重启调试器。注意，这里不需要我们重启 OpenOCD。

一旦程序停止运行，输入命令 `watch i`:

```
(gdb) watch i
Hardware watchpoint 2: i
(gdb)
```

这会在所有变量 `i` 发生改变的代码处插入所谓的 “观察点”。现在输入 `continue` 命令来恢复应用程序的运行并观察它停止:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB751 (active)    APP_CPU: PC=0x400D0811
[New Thread 1073432196]

Program received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 1073432196]
0x400db751 in blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↪blink.c:33
33          i++;
(gdb)
```

多次恢复程序运行后，变量 `i` 的值会增加，现在你可以输入 `print i`（简写 `p i`）来查看当前 `i` 的值:

```
(gdb) p i
$1 = 3
(gdb)
```

要修改 `i` 的值，请使用 `set` 命令，如下所示（可以将其打印输出来查看是否确已修改）:

```
(gdb) set var i = 0
(gdb) p i
$3 = 0
(gdb)
```

最多可以使用两个观察点，详细信息请参阅可用的断点和观察点。

设置条件断点 接下来的内容更为有趣，你可能想在一定条件满足的情况下设置断点。请先删除已有的断点，然后尝试如下命令:

```
(gdb) break blink.c:34 if (i == 2)
Breakpoint 3 at 0x400db753: file /home/user-name/esp/blink/main/./blink.c, line 34.
(gdb)
```

以上命令在 `blink.c` 文件的 34 处设置了一个条件断点，当 `i == 2` 条件满足时，程序会停止运行。

如果当前 `i` 的值小于 2 并且程序被恢复运行，那么 LED 就会循环闪烁，直到 `i == 2` 条件成立，最后程序停止在该处:

```
(gdb) set var i = 0
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB755 (active)    APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB753 (active)    APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB755 (active)    APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB753 (active)    APP_CPU: PC=0x400D112C

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↪blink.c:34
34          gpio_set_level(BLINK_GPIO, 0);
(gdb)
```

调试 FreeRTOS 对象 该部分内容或许可以帮助你调试 FreeRTOS 任务交互。

需要调试 FreeRTOS 任务交互的用户可使用 GDB 命令 `freertos`。该命令并非 GDB 原生命令，而是来自于 Python 扩展模块 freertos-gdb，其包含一系列子命令:

```
(gdb) freertos
"freertos" 后面必须紧随子命令的名称
freertos 子命令如下：

freertos queue --  打印当前队列信息
freertos semaphore --  打印当前信号量信息
freertos task --  打印当前任务及其状态
freertos timer --  打印当前定时器信息
```

点击 https://pypi.org/project/freertos-gdb 链接了解此扩展模块的详细信息。

备注: ESP-IDF 在安装 Pyhton 包时会自动安装 freertos-gdb Python 模块，详情请参考第三步：设置工具。

如果使用 `idf.py gdb` 命令运行 GDB，FreeRTOS 扩展会自动加载。也可以使用 GDB 内部命令 `python import freertos_gdb` 使能该模块。

请保证使用 Python 3.6 及以上版本，该版本具有 Python 共享库。

获得命令的帮助信息 目前所介绍的都是些非常基础的命令，目的在于让你快速上手 JTAG 调试。如果想获得特定命令的语法和功能相关的信息，请在 `(gdb)` 提示符下输入 `help` 和命令名:

```
(gdb) help next
Step program, proceeding through subroutine calls.
Usage: next [N]
Unlike "step", if the current source line calls a subroutine,
this command does not enter the subroutine, but instead steps over
the call, in effect treating it as a single source line.
(gdb)
```

只需输入 `help` 命令，即可获得高级命令列表，帮助你了解更多详细信息。此外，还可以参考一些 GDB 命令速查表，比如 https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf。虽然不是所有命令都适用于嵌入式环境，但还是会有所裨益。

结束调试会话 输入命令 q 可以退出调试器:

```
(gdb) q
A debugging session is active.

    Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) y
Detaching from program: /home/user-name/esp/blink/build/blink.elf, Remote target
Ending remote debugging.
user-name@computer-name:~/esp/blink$
```

- 使用调试器
- 调试示例
- 注意事项和补充内容
- 应用层跟踪库
- ESP-Prog 调试板介绍

4.14 链接器脚本生成机制

4.14.1 概述

ESP32-P4 中有多个用于存放代码和数据的内存区域 。代码和只读数据默认存放在 flash 中，可写数据存放在 RAM 中。不过有时，用户必须更改默认存放区域。

例如:

- 将关键代码存放到 RAM 中以提高性能;
- 将可执行代码存放到 IRAM 中，以便在缓存被禁用时运行这些代码;
- 将代码存放到 RTC 内存中，以便 ULP 协处理器使用。

链接器脚本生成机制可以让用户指定代码和数据在 ESP-IDF 组件中的存放区域。组件包含如何存放符号、目标或完整库的信息。在构建应用程序时，组件中的这些信息会被收集、解析并处理；生成的存放规则用于链接应用程序。

4.14.2 快速上手

本段将指导如何使用 ESP-IDF 的即用方案，快速将代码和数据放入 RAM 和 RTC 存储器中。

假设用户有:

```
components
└── my_component
    ├── CMakeLists.txt
    ├── Kconfig
    ├── src/
    │   ├── my_src1.c
    │   ├── my_src2.c
    │   └── my_src3.c
    └── my_linker_fragment_file.lf
```

- 名为 my_component 的组件，在构建过程中存储为 libmy_component.a 库文件
- 库文件包含的三个源文件: my_src1.c、my_src2.c 和 my_src3.c，编译后分别为 my_src1.o、my_src2.o 和 my_src3.o
- 在 my_src1.o 中定义 my_function1 功能；在 my_src2.o 中定义 my_function2 功能

- 在 my_component 下 Kconfig 中存在布尔类型配置 PERFORMANCE_MODE (y/n) 和整数类型配置 PERFORMANCE_LEVEL （范围是 0-3）

创建和指定链接器片段文件

首先，用户需要创建链接器片段文件。链接器片段文件是一个扩展名为 .lf 的文本文件，想要存放的位置信息会写入该文件内。文件创建成功后，需要将其呈现在构建系统中。ESP-IDF 支持的构建系统指南如下：

在组件目录的 CMakeLists.txt 文件中，指定 idf_component_register 调用引数 LDFRAGMENTS 的值。LDFRAGMENTS 可以为绝对路径，也可为组件目录的相对路径，指向已创建的链接器片段文件。

```
# 相对于组件的 CMakeLists.txt 的文件路径
idf_component_register(...
                       LDFRAGMENTS "path/to/linker_fragment_file.lf" "path/to/
↪another_linker_fragment_file.lf"
                       ...
                       )
```

指定存放区域

可以按照下列粒度指定存放区域：

- 目标文件（.obj 或 .o 文件）
- 符号（函数/变量）
- 库（.a 文件）

存放目标文件 假设整个 my_src1.o 目标文件对性能至关重要，所以最好把该文件放在 RAM 中。另外，my_src2.o 目标文件包含从深度睡眠唤醒所需的符号，因此需要将其存放到 RTC 存储器中。

在链接器片段文件中可以写入以下内容：

```
[mapping:my_component]
archive: libmy_component.a
entries:
    my_src1 (noflash)      # 将所有 my_src1 代码和只读数据存放在 IRAM 和 DRAM 中
    my_src2 (rtc)          # 将所有 my_src2 代码、数据和只读数据存放到 RTC 快速 RAM␣
↪和 RTC 慢速 RAM 中
```

那么 my_src3.o 放在哪里呢？由于未指定存放区域，my_src3.o 会存放到默认区域。更多关于默认存放区域的信息，请查看这里。

存放符号 继续上文的例子，假设 object1.o 目标文件定义的功能中，只有 my_function1 影响到性能；object2.o 目标文件中只有 my_function2 需要在芯片从深度睡眠中唤醒后运行。要实现该目的，可在链接器片段文件中写入以下内容：

```
[mapping:my_component]
archive: libmy_component.a
entries:
    my_src1:my_function1 (noflash)
    my_src2:my_function2 (rtc)
```

my_src1.o 和 my_src2.o 中的其他函数以及整个 object3.o 目标文件会存放到默认区域。要指定数据的存放区域，仅需将上文的函数名替换为变量名即可，如：

```
my_src1:my_variable (noflash)
```

注意： 按照符号粒度存放代码和数据有一定的局限。为确保存放区域合适，您也可以将相关代码和数据集中在源文件中，参考使用目标文件的存放规则。

存放整个库 在这个例子中，假设整个组件库都需存放到 RAM 中，可以写入以下内容存放整个库：

```
[mapping:my_component]
archive: libmy_component.a
entries:
    * (noflash)
```

类似的，写入以下内容可以将整个组件存放到 RTC 存储器中：

```
[mapping:my_component]
archive: libmy_component.a
entries:
    * (rtc)
```

根据具体配置存放 假设只有在某个条件为真时，比如 `CONFIG_PERFORMANCE_MODE == y` 时，整个组件库才有特定存放区域，可以写入以下内容实现：

```
[mapping:my_component]
archive: libmy_component.a
entries:
    if PERFORMANCE_MODE = y:
        * (noflash)
    else:
        * (default)
```

来看一种更复杂的情况。假设 `CONFIG_PERFORMANCE_LEVEL == 1` 时，只有 `object1.o` 存放到 RAM 中；`CONFIG_PERFORMANCE_LEVEL == 2` 时，`object1.o` 和 `object2.o` 会存放到 RAM 中；`CONFIG_PERFORMANCE_LEVEL == 3` 时，库中的所有目标文件都会存放到 RAM 中。以上三个条件为假时，整个库会存放到 RTC 存储器中。虽然这种使用场景很罕见，不过，还是可以通过以下方式实现：

```
[mapping:my_component]
archive: libmy_component.a
entries:
    if PERFORMANCE_LEVEL = 1:
        my_src1 (noflash)
    elif PERFORMANCE_LEVEL = 2:
        my_src1 (noflash)
        my_src2 (noflash)
    elif PERFORMANCE_LEVEL = 3:
        my_src1 (noflash)
        my_src2 (noflash)
        my_src3 (noflash)
    else:
        * (rtc)
```

也可以嵌套条件检查。以下内容与上述片段等效：

```
[mapping:my_component]
archive: libmy_component.a
entries:
    if PERFORMANCE_LEVEL <= 3 && PERFORMANCE_LEVEL > 0:
        if PERFORMANCE_LEVEL >= 1:
            object1 (noflash)
            if PERFORMANCE_LEVEL >= 2:
                object2 (noflash)
```

(下页继续)

(续上页)

```
                if PERFORMANCE_LEVEL >= 3:
                    object2 (noflash)
    else:
        * (rtc)
```

默认存放区域

到目前为止，“默认存放区域”在未指定 rtc 和 noflash 存放规则时才会作为备选方案使用。需要注意的是，noflash 或者 rtc 标记不仅仅是关键字，实际上还是被称作片段的实体，确切地说是协议。

与 rtc 和 noflash 类似，还有一个 默认协议，定义了默认存放规则。顾名思义，该协议规定了代码和数据通常存放的区域，即代码和恒量存放在 flash 中，变量存放在 RAM 中。更多关于默认协议的信息，请见这里。

备注： 使用链接器脚本生成机制的 IDF 组件示例，请参阅 freertos/CMakeLists.txt。为了提高性能，freertos 使用链接器脚本生成机制，将其目标文件存放到 RAM 中。

快速入门指南到此结束，下文将详述这个机制的内核，有助于创建自定义存放区域或修改默认方式。

4.14.3 链接器脚本生成机制内核

链接是将 C/C++ 源文件转换成可执行文件的最后一步。链接由工具链的链接器完成，接受指定代码和数据存放区域等信息的链接脚本。链接器脚本生成机制的转换过程类似，区别在于传输给链接器的链接脚本根据 (1) 收集的链接器片段文件 和 (2) 链接器脚本模板 动态生成。

备注： 执行链接器脚本生成机制的工具存放在 tools/ldgen 之下。

链接器片段文件

如快速入门指南所述，片段文件是拓展名为 .lf 的简单文本文件，内含想要存放区域的信息。不过，这是对片段文件所包含内容的简化版描述。实际上，片段文件内包含的是“片段”。片段是实体，包含多条信息，这些信息放在一起组成了存放规则，说明目标文件各个段在二进制输出文件中的存放位置。片段一共有三种，分别是段、协议 和映射。

语法 三种片段类型使用同一种语法：

```
[type:name]
key: value
key:
    value
    value
    value
    ...
```

- 类型：片段类型，可以为 段、协议或 映射。
- 名称：片段名称，指定片段类型的片段名称应唯一。
- 键值：片段内容。每个片段类型可支持不同的键值和不同的键值语法。
  - 在段 和协议 中，仅支持 entries 键。
  - 在映射 中，支持 archive 和 entries 键。

备注： 多个片段的类型和名称相同时会引发异常。

备注： 片段名称和键值只能使用字母、数字和下划线。

条件检查

条件检查使得链接器脚本生成机制可以感知配置。含有配置值的表达式是否为真，决定了使用哪些特定键值。检查使用的是 kconfiglib 脚本的 `eval_string`，遵循该脚本要求的语法和局限性，支持：

- **比较**
  - 小于 <
  - 小于等于 <=
  - 大于 >
  - 大于等于 >=
  - 等于 =
  - 不等于 !=
- **逻辑**
  - 或 ||
  - 和 &&
  - 取反 !
- **分组**
  - 圆括号 ()

条件检查和其他语言中的 `if...elseif/elif...else` 块作用一样。键值和完整片段都可以进行条件检查。以下两个示例效果相同：

```
# 键值取决于配置
[type:name]
key_1:
    if CONDITION = y:
        value_1
    else:
        value_2
key_2:
    if CONDITION = y:
        value_a
    else:
        value_b
```

```
# 完整片段的定义取决于配置
if CONDITION = y:
    [type:name]
    key_1:
        value_1
    key_2:
        value_a
else:
    [type:name]
    key_1:
        value_2
    key_2:
        value_b
```

注释

链接器片段文件中的注释以 # 开头。和在其他语言中一样，注释提供了有用的描述和资料，在处理过程中会被忽略。

类型 段

段定义了 GCC 编译器输出的一系列目标文件段，可以是默认段（如 `.text`、`.data`），也可以是用户通过 `__attribute__` 关键字定义的段。

'+' 表示段列表开始，且当前段为列表中的第一个段。这种表达方式更加推荐。

```
[sections:name]
entries:
    .section+
    .section
    ...
```

示例：

```
# 不推荐的方式
[sections:text]
entries:
    .text
    .text.*
    .literal
    .literal.*

# 推荐的方式，效果与上面等同
[sections:text]
entries:
    .text+              # 即 .text 和 .text.*
    .literal+           # 即 .literal 和 .literal.*
```

协议

协议定义了每个段对应的 目标。

```
[scheme:name]
entries:
    sections -> target
    sections -> target
    ...
```

示例：

```
[scheme:noflash]
entries:
    text -> iram0_text          # text 段下的所有条目均归入 iram0_text
    rodata -> dram0_data        # rodata 段下的所有条目均归入 dram0_data
```

默认协议

注意，有一个 默认的协议很特殊，特殊在于包罗存放规则都是根据这个协议中的条目生成的。这意味着，如果该协议有一条条目是 `text -> flash_text`，则将为目标 `flash_text` 生成如下的存放规则:

```
*(.literal .literal.* .text .text.*)
```

这些生成的包罗规则将用于未指定映射规则的情况。

默认协议在 esp_system/app.lf 文件中定义。快速上手指南中提到的内置 `noflash` 协议和 `rtc` 协议也在该文件中定义。

映射

映射定义了可映射实体（即目标文件、函数名、变量名和库）对应的协议。

```
[mapping]
archive: archive                # 构建后输出的库文件名称（即 libxxx.a）
entries:
    object:symbol (scheme)      # 符号
    object (scheme)             # 目标
    * (scheme)                  # 库
```

有三种存放粒度：

- 符号：指定了目标文件名称和符号名称。符号名称可以是函数名或变量名。
- 目标：只指定目标文件名称。
- 库：指定 *，即某个库下面所有目标文件的简化表达法。

为了更好地理解条目的含义，请看一个按目标存放的例子。

```
object (scheme)
```

根据条目定义，将这个协议展开：

```
object (sections -> target,
        sections -> target,
        ...)
```

再根据条目定义，将这个段展开：

```
object (.section,
        .section,
        ... -> target, # 根据目标文件将这里所列出的所有段放在该目标位置

        .section,
        .section,
        ... -> target, # 同样的方法指定其他段

        ...)           # 直至所有段均已展开
```

示例：

```
[mapping:map]
archive: libfreertos.a
entries:
    * (noflash)
```

除了实体和协议，条目中也支持指定如下标志：（注：<> = 参数名称，[] = 可选参数）

1. ALIGN(<alignment>[, pre, post])
   根据 alignment 中指定的数字对齐存放区域，根据是否指定 pre 和 post，或两者都指定，在输入段描述（生成于映射条目）的前面和/或后面生成：
2. SORT([<sort_by_first>, <sort_by_second>])
   在输入段描述中输出 SORT_BY_NAME, SORT_BY_ALIGNMENT, SORT_BY_INIT_PRIORITY 或 SORT。
   sort_by_first 和 sort_by_second 的值可以是：name、alignment、init_priority。
   如果既没指定 sort_by_first 也没指定 sort_by_second，则输入段会按照名称排序，如果两者都指定了，那么嵌套排序会遵循 https://sourceware.org/binutils/docs/ld/Input-Section-Wildcards.html 中的规则。
3. KEEP()
   用 KEEP 命令包围输入段描述，从而防止链接器丢弃存放区域。更多细节请参考 https://sourceware.org/binutils/docs/ld/Input-Section-Keep.html

4.SURROUND(<name>)

在存放区域的前面和后面生成符号，生成的符号遵循 _<name>_start 和 _<name>_end 的命名方式，例如，如果 name == sym1

在添加标志时，协议中需要指定具体的 section -> target。对于多个 section -> target，使用逗号作为分隔符，例如：

```
# 注意
# A. entity-scheme 后使用分号
# B. section2 -> target2 前使用逗号
# C. 在 scheme1 条目中定义 section1 -> target1 和 section2 -> target2
entity1 (scheme1);
```

（下页继续）

(续上页)

```
    section1 -> target1 KEEP() ALIGN(4, pre, post),
    section2 -> target2 SURROUND(sym) ALIGN(4, post) SORT()
```

合并后，如下的映射：

```
[mapping:name]
archive: lib1.a
entries:
    obj1 (noflash);
        rodata -> dram0_data KEEP() SORT() ALIGN(8) SURROUND(my_sym)
```

会在链接器脚本上生成如下输出：

```
. = ALIGN(8)
_my_sym_start = ABSOLUTE(.)
KEEP(lib1.a:obj1.*( SORT(.rodata) SORT(.rodata.*) ))
_my_sym_end = ABSOLUTE(.)
```

注意，正如在 flag 描述中提到的，ALIGN 和 SURROUND 的使用对顺序敏感，因此如果将两者顺序调换后用到相同的映射片段，则会生成：

```
_my_sym_start = ABSOLUTE(.)
. = ALIGN(8)
KEEP(lib1.a:obj1.*( SORT(.rodata) SORT(.rodata.*) ))
_my_sym_end = ABSOLUTE(.)
```

按符号存放 按符号存放可通过编译器标志 `-ffunction-sections` 和 `-ffdata-sections` 实现。ESP-IDF 默认用这些标志编译。用户若选择移除标志，便不能按符号存放。另外，即便有标志，也会其他限制，具体取决于编译器输出的段。

比如，使用 `-ffunction-sections`，针对每个功能会输出单独的段。段的名称可以预测，即 `.text.{func_name}` 和 `.literal.{func_name}`。但是功能内的字符串并非如此，因为字符串会进入字符串池，或者使用生成的段名称。

使用 `-fdata-sections`，对全局数据来说编译器可输出 `.data.{var_name}`、`.rodata.{var_name}` 或 `.bss.{var_name}`；因此类型 `I` 映射词条可以适用。但是，功能中声明的静态数据并非如此，生成的段名称是将变量名称和其他信息混合。

链接器脚本模板

链接器脚本模板是指定存放规则的存放位置的框架，与其他链接器脚本没有本质区别，但带有特定的标记语法，可以指示存放生成的存放规则的位置。

如需引用一个目标标记下的所有存放规则，请使用以下语法：

```
arrays[target]      /* SURROUND 关键字下的对象 */
mapping[target]     /* 所有其他数据 */
```

示例：

以下示例是某个链接器脚本模板的摘录，定义了输出段 `.iram0.text`，该输出段包含一个引用目标 `iram0_text` 的标记。

```
.iram0.text :
{
    /* 标记 IRAM 空间不足 */
    _iram_text_start = ABSOLUTE(.);

    /* 引用 iram0_text */
```

(下页继续)

(续上页)

```
    arrays[iram0_text]
    mapping[iram0_text]

    _iram_text_end = ABSOLUTE(.);
} > iram0_0_seg
```

假设链接器脚本生成器收集到了以下片段定义：

```
[sections:text]
    .text+
    .literal+

[sections:iram]
    .iram1+

[scheme:default]
entries:
    text -> flash_text
    iram -> iram0_text

[scheme:noflash]
entries:
    text -> iram0_text

[mapping:freertos]
archive: libfreertos.a
entries:
    * (noflash)
```

然后生成的链接器脚本的相应摘录如下：

```
.iram0.text :
{
    /* 标记 IRAM 空间不足 */
    _iram_text_start = ABSOLUTE(.);

    /* 处理片段生成的存放规则，存放在模板标记的位置处 */
    *(.iram1 .iram1.*)
    *libfreertos.a:(.literal .text .literal.* .text.*)

    _iram_text_end = ABSOLUTE(.);
} > iram0_0_seg
```

`*libfreertos.a:(.literal .text .literal.* .text.*)`

这是根据 `freertos` 映射的 `* (noflash)` 条目生成的规则。`libfreertos.a` 库下所有目标文件的所有 `text` 段会收集到 `iram0_text` 目标下（按照 `noflash` 协议），并放在模板中被 `iram0_text` 标记的地方。

`*(.iram1 .iram1.*)`

这是根据默认协议条目 `iram -> iram0_text` 生成的规则。默认协议指定了 `iram -> iram0_text` 条目，因此生成的规则同样也放在被 `iram0_text` 标记的地方。由于该规则是根据默认协议生成的，因此在同一目标下收集的所有规则下排在第一位。

目前使用的链接器脚本模板是 esp_system/ld/esp32p4/sections.ld.in，生成的脚本存放在构建目录下。

4.15 lwIP

ESP-IDF 使用开源的 lwIP 轻量级 TCP/IP 协议栈，该版 lwIP (esp-lwip) 相对上游项目做了修改和增补。

4.15.1 支持的 API

ESP-IDF 支持以下 lwIP TCP/IP 协议栈功能：

- *BSD 套接字 API*
- *Netconn API* 已启用，但暂无对 ESP-IDF 应用程序的官方支持

适配的 API

> **警告：** 在使用除*BSD 套接字 API* 外的任意 lwIP API 时，请确保所用 API 为线程安全。请启用*CONFIG_LWIP_CHECK_THREAD_SAFETY* 配置选项并运行应用程序，检查所用 API 是否线程安全。此时，lwIP 断言 TCP/IP 核心功能可以正确访问。如果未能从正确的*lwIP FreeRTOS 任务* 访问，或没有正确锁定，则执行中止。建议使用*ESP-NETIF* 组件与 lwIP 交互。

ESP-IDF 间接支持以下常见的 lwIP 应用程序 API：

- 动态主机设置协议 (DHCP) 服务器和客户端，由*ESP-NETIF* 功能间接支持。
- 域名系统 (DNS)；获取 DHCP 地址时，可以自动分配 DNS 服务器，也可以通过*ESP-NETIF* API 手动配置。

备注： lwIP 中的 DNS 服务器配置为全局配置，而非针对特定接口的配置。如需同时使用不同 DNS 服务器的多个网络接口，在从一个接口获取 DHCP 租约时，请注意避免意外覆盖另一个接口的 DNS 设置。

- 简单网络时间协议 (SNTP)，由*ESP-NETIF* 功能间接支持，或通过 lwip/include/apps/esp_sntp.h 中的函数直接支持。该函数还为 lwip/lwip/src/include/lwip/apps/sntp.h 函数提供了线程安全的 API，请参阅*SNTP 时间同步*。
- ICMP Ping，由 lwIP ping API 的变体支持，请参阅*ICMP 回显*。
- ICMPv6 Ping，由 lwIP 的 ICMPv6 Echo API 支持，用于测试 IPv6 网络连接情况。有关详细信息，请参阅 protocols/sockets/icmpv6_ping。
- NetBIOS 查找，由标准的 lwIP API 支持，protocols/http_server/restful_server 示例中提供了使用 NetBIOS 在局域网中查找主机的选项。
- mDNS 与 lwIP 的默认 mDNS 使用不同实现方式，请参阅*mDNS 服务*。但启用*CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES* 设置项后，lwIP 可以使用 `gethostbyname()` 等标准 API 和 `hostname.local` 约定查找 mDNS 主机。
- lwIP 中的 PPP 实现可用于在 ESP-IDF 中创建 PPPoS（串行 PPP）接口。请参阅*ESP-NETIF* 组件文档，使用 esp_netif/include/esp_netif_defaults.h 中定义的 `ESP_NETIF_DEFAULT_PPP()` 宏创建并配置 PPP 网络接口。esp_netif/include/esp_netif_ppp.h 中提供了其他的运行时设置。PPPoS 接口通常用于与 NBIoT/GSM/LTE 调制解调器交互。esp_modem 仓库还支持更多应用层友好的 API，该仓库内部使用了上述 PPP lwIP 模块。

4.15.2 BSD 套接字 API

BSD 套接字 API 是一种常见的跨平台 TCP/IP 套接字 API，最初源于 UNIX 操作系统的伯克利标准发行版，现已标准化为 POSIX 规范的一部分。BSD 套接字有时也称 POSIX 套接字，或伯克利套接字。

在 ESP-IDF 中，lwIP 支持 BSD 套接字 API 的所有常见用法。

参考

BSD 套接字的相关参考资料十分丰富，包括但不限于：

- 单一 UNIX 规范 - BSD 套接字
- 伯克利套接字 - 维基百科

示例

以下为 ESP-IDF 中使用 BSD 套接字 API 的部分示例：

- protocols/sockets/tcp_server
- protocols/sockets/tcp_client
- protocols/sockets/udp_server
- protocols/sockets/udp_client
- protocols/sockets/udp_multicast
- protocols/http_request：此简化示例使用 TCP 套接字发送 HTTP 请求，但更推荐使用*ESP HTTP 客户端* 发送 HTTP 请求

支持的函数

在 ESP-IDF 中，lwIP 支持以下 BSD 套接字 API 函数，详情请参阅 lwip/lwip/src/include/lwip/sockets.h。

- `socket()`
- `bind()`
- `accept()`
- `shutdown()`
- `getpeername()`
- `getsockopt()` 和 `setsockopt()`：请参阅*套接字选项*
- `close()`：通过*虚拟文件系统组件* 调用
- `read()`、`readv()`、`write()`、`writev()`：通过*虚拟文件系统组件* 调用
- `recv()`、`recvmsg()`、`recvfrom()`
- `send()`、`sendmsg()`、`sendto()`
- `select()`：通过*虚拟文件系统组件* 调用
- `poll()`：ESP-IDF 通过在内部调用 `select()` 实现 `poll()`，因此，建议直接调用 `select()`
- `fcntl()`：请参阅*fcntl()*

非标准函数：

- `ioctl()`：请参阅*ioctl()*

备注： 部分 lwIP 应用程序示例代码使用了带前缀的 BSD API，如 `lwip_socket()`，而非标准 `socket()`。ESP-IDF 支持使用以上两种形式，但更建议使用标准名称。

套接字错误处理

要使套接字应用程序保持稳定，BSD 套接字错误处理代码至关重要。套接字错误处理通常涉及以下几个方面：

- 错误检测
- 获取错误原因代码
- 根据错误原因代码处理错误

在 lwIP 中，处理套接字错误分以下两种情况：

- 套接字 API 返回错误，请参阅*套接字 API 错误*。
- `select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout)` 包含异常描述符，表示套接字出现错误，详情请参阅*select() 错误*。

套接字 API 错误　错误检测

- 根据返回值判断套接字 API 是否出错。

获取错误原因代码

- 套接字 API 出错时，其返回值不包含失败原因，可以通过应用程序访问 `errno` 获取错误原因代码。不同返回值具有不同含义，详情请参阅套接字错误原因代码。

示例：

```
int err;
int sockfd;

if (sockfd = socket(AF_INET,SOCK_STREAM,0) < 0) {
    // 从 errno 获取错误代码
    err = errno;
    return err;
}
```

`select()` 错误　错误检测

- `select()` 包含异常描述符时的套接字错误。

获取错误原因代码

- 如果 `select()` 报告套接字错误，访问 `errno` 无法获取错误原因代码，此时，应调用 `getsockopt()`。因为当 `select()` 包含异常描述符时，错误代码不会直接赋值给 `errno`。

备注： `getsockopt()` 函数具有以下原型：`int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen)`。原型可以获取任意类型、任意状态套接字选项的当前值，并将结果存储在 `optval` 中。例如，要在套接字上获取错误代码，可以通过 `getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &err, &optlen)` 实现。

示例：

```
int err;

if (select(sockfd + 1, NULL, NULL, &exfds, &tval) <= 0) {
    err = errno;
    return err;
} else {
    if (FD_ISSET(sockfd, &exfds)) {
        // 使用 getsockopt() 获取 select() 异常集
        int optlen = sizeof(int);
        getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &err, &optlen);
        return err;
    }
}
```

套接字错误原因代码　以下是常见错误代码列表。有关标准 POSIX/C 错误代码的详细列表，请参阅 newlib errno.h 和特定平台扩展 newlib/platform_include/errno.h。

错误代码	描述
ECONNREFUSED	拒绝连接
EADDRINUSE	地址已在使用中
ECONNABORTED	软件导致连接中断
ENETUNREACH	网络不可达
ENETDOWN	未配置网络接口
ETIMEDOUT	连接超时
EHOSTDOWN	主机已关闭
EHOSTUNREACH	主机不可达
EINPROGRESS	连接已在进行中
EALREADY	套接字已连接
EDESTADDRREQ	需要目标地址
EPROTONOSUPPORT	未知协议

套接字选项

`getsockopt()` 支持获取套接字选项，`setsockopt()` 支持设置套接字选项。

在 ESP-IDF 中，lwIP 并不支持所有标准套接字选项。以下套接字选项受 lwIP 支持：

常见选项 与级别参数 `SOL_SOCKET` 一起使用。

- `SO_REUSEADDR`：如果*CONFIG_LWIP_SO_REUSE* 已启用，则该选项可用，可以设置*CONFIG_LWIP_SO_REUSE_RXTOALL* 自定义其行为
- `SO_KEEPALIVE`
- `SO_BROADCAST`
- `SO_ACCEPTCONN`
- `SO_RCVBUF`：如果*CONFIG_LWIP_SO_RCVBUF* 已启用，则该选项可用
- `SO_SNDTIMEO` / `SO_RCVTIMEO`
- `SO_ERROR`：此选项仅支持与 `select()` 一起使用，请参阅套接字错误处理
- `SO_TYPE`
- `SO_NO_CHECK`：仅适用于 UDP 套接字

IP 选项 与级别参数 `IPPROTO_IP` 一起使用。

- `IP_TOS`
- `IP_TTL`
- `IP_PKTINFO`：如果*CONFIG_LWIP_NETBUF_RECVINFO* 已启用，则该选项可用

对于组播 UDP 套接字：

- `IP_MULTICAST_IF`
- `IP_MULTICAST_LOOP`
- `IP_MULTICAST_TTL`
- `IP_ADD_MEMBERSHIP`
- `IP_DROP_MEMBERSHIP`

TCP 选项 只适用于 TCP 套接字，与级别参数 `IPPROTO_TCP` 一起使用。

- `TCP_NODELAY`

与 TCP 保活探测相关的选项：

- `TCP_KEEPALIVE`：整数值，以毫秒为单位，设置 TCP 保活探测周期
- `TCP_KEEPIDLE`：整数值，以秒为单位，与 `TCP_KEEPALIVE` 相同
- `TCP_KEEPINTVL`：整数值，以秒为单位，设置保活探测间隔
- `TCP_KEEPCNT`：整数值，设置超时前进行的保活探测次数

IPv6 选项 只适用于 IPv6 套接字，与级别参数 `IPPROTO_IPV6` 一起使用。

- `IPV6_CHECKSUM`
- `IPV6_V6ONLY`

对于组播 IPv6 UDP 套接字：

- `IPV6_JOIN_GROUP` / `IPV6_ADD_MEMBERSHIP`
- `IPV6_LEAVE_GROUP` / `IPV6_DROP_MEMBERSHIP`
- `IPV6_MULTICAST_IF`
- `IPV6_MULTICAST_HOPS`
- `IPV6_MULTICAST_LOOP`

`fcntl()`

`fcntl()` 函数是设置与文件描述符相关选项的标准 API。在 ESP-IDF 中，使用虚拟文件系统组件 层实现该函数。

当文件描述符为套接字时，仅支持以下 `fcntl()` 值：

- `O_NONBLOCK` 用于置位或清除非阻塞 I/O 模式。`O_NDELAY` 也受支持，与前者功能相同。
- `O_RDONLY`、`O_WRONLY`、`O_RDWR` 标志用于不同的读或写模式，只能用 `F_GETFL` 读取，且无法用 `F_SETFL` 设置。根据连接状况，即两端开启或任一端关闭，TCP 套接字会返回不同模式，而 UDP 套接字始终返回 `O_RDWR`。

`ioctl()`

`ioctl()` 函数以半标准的方式访问 TCP/IP 协议栈的部分内部功能。ESP-IDF 通过虚拟文件系统组件 层实现此函数。

当文件描述符为套接字时，仅支持以下 `ioctl()` 值：

- `FIONREAD` 返回套接字网络 buffer 中接收的待处理字节数。
- `FIONBIO` 和 `fcntl(fd, F_SETFL, O_NONBLOCK, ...)` 相同，也可置位或清除套接字非阻塞 I/O 状态。

4.15.3 Netconn API

lwIP 支持两种较低级别的 API 和 BSD 套接字 API，即 Netconn API 和 Raw API。

lwIP Raw API 适用于单线程设备，无法在 ESP-IDF 中使用。

Netconn API 用于在 lwIP 内部使用 BSD 套接字 API，支持直接从 ESP-IDF 的应用程序调用。相较于 BSD 套接字 API，该 API 占用资源更少。无需提前将数据复制到内部 lwIP buffer，即可使用 Netconn API 发送和接收数据。

重要： 乐鑫尚未在 ESP-IDF 中测试 Netconn API，因此 **此功能已启用，但尚无官方支持**。对于某些功能，可能只有在从 BSD 套接字 API 中使用时才能正常运作。

有关 Netconn API 的更多信息，请参阅 lwip/lwip/src/include/lwip/api.h 和 lwIP 应用程序 ** 非官方 ** 开发手册的一部分。

4.15.4 lwIP FreeRTOS 任务

lwIP 创建了专用的 TCP/IP FreeRTOS 任务，处理来自其他任务的套接字 API 请求。

以下配置项可用于修改任务，并调整向 TCP/IP 任务发送数据和从 TCP/IP 任务接收数据的队列（邮箱）：

- *CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*

- *CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*
- *CONFIG_LWIP_TCPIP_TASK_AFFINITY*

4.15.5 IPv6 支持

系统支持 IPv4 和 IPv6 的双栈功能，并默认启用这两种协议。如无需要，可将其禁用，请参阅最小内存使用。

在 ESP-IDF 中，IPv6 支持仅限 **无状态自动配置**，不支持 **有状态配置**，上游的 lwIP 也不支持 **有状态配置**。

IPv6 地址配置通过以下协议或服务定义：

- 支持 **SLAAC** IPv6 无状态地址配置 (RFC-2462)
- 支持 **DHCPv6** IPv6 动态主机配置协议 (RFC-8415)

以上两种地址配置默认处于禁用状态，设备仅使用链路本地地址或静态定义的地址。

无状态自动配置流程

要通过路由器通告协议启用地址自动配置，请启用此配置选项：

- *CONFIG_LWIP_IPV6_AUTOCONFIG*

该配置选项启用了所有网络接口的 IPv6 自动配置。而在上游 lwIP 中，需要设置 `netif->ip6_autoconfig_enabled=1`，针对每个 `netif` 明确启用自动配置。

DHCPv6

lwIP 中的 DHCPv6 非常简单，仅支持无状态配置，可通过以下配置选项启用：

- *CONFIG_LWIP_IPV6_DHCP6*

由于 DHCPv6 仅在无状态配置下工作，因此还需要通过*CONFIG_LWIP_IPV6_AUTOCONFIG* 启用无状态自动配置流程。

此外，还需要使用以下语句，在应用程序代码中明确启用 DHCPv6：

```
dhcp6_enable_stateless(netif);
```

IPv6 自动配置中的 DNS 服务器

要自动配置 DNS 服务器，尤其是在仅使用 IPv6 的网络中配置，可使用以下两种选项：

- 递归域名系统 (DNS)，属于邻居发现协议 (NDP) 的一部分，可使用无状态自动配置流程。
  DNS 服务器的数量必须设置为*CONFIG_LWIP_IPV6_RDNSS_MAX_DNS_SERVERS*，该选项默认禁用，即置位为 0。
- DHCPv6 无状态配置，使用*DHCPv6* 配置 DNS 服务器。注意，此配置假设 IPv6 路由通告标志 (RFC-5175) 进行了如下设置
  - 管理地址配置标志 (Managed Address Configuration Flag) = 0
  - 其他配置标志 (Other Configuration Flag) = 1

4.15.6 ESP-lwIP 自定义修改

补充内容

以下代码均为新增代码，尚未包含至上游 lwIP 版本：

线程安全的套接字 调用 `close()` 可以从不同于创建套接字的线程中关闭该套接字。该调用持续阻塞，直至其他任务中使用该套接字的函数调用返回。

然而，任务处于主动等待 `select()` 或 `poll()` API 的状态时，无法删除该任务。销毁任务前，这些 API 必须先退出，否则可能会破坏内部数据结构，并导致后续 lwIP 崩溃。这些 API 在栈上分配了全局引用的回调指针，因此，在未完全卸载栈的情况下删除任务时，lwIP 仍可以持有指向已删除栈的指针。

按需定时器 lwIP 中的 IGMP 和 MLD6 功能都会初始化一个定时器，以便在特定时间触发超时事件。

即便没有活动的超时事件，lwIP 也会默认始终启用这些定时器，增加自动 Light-sleep 模式下的 CPU 使用率和功耗。`ESP-lwIP` 则默认将各定时器设置为 *按需* 使用，即只在有待处理事件时启用。

如果要返回默认 lwIP 设置，即始终启用定时器，请禁用*CONFIG_LWIP_TIMERS_ONDEMAND*。

lwIP 定时器 API 不使用 Wi-Fi 时，可以通过 API 关闭 lwIP 定时器，减少功耗。

以下 API 函数均受支持，详情请参阅 lwip/lwip/src/include/lwip/timeouts.h。

- `sys_timeouts_init()`
- `sys_timeouts_deinit()`

附加套接字选项

- 目前已实现部分标准 IPV4 和 IPV6 组播套接字选项，详情请参阅套接字选项。
- 使用 `IPV6_V6ONLY` 套接字选项，可以设置仅使用 IPV6 的 UDP 和 TCP 套接字，而 lwIP 一般只支持 TCP 套接字。

IP 层特性

- IPV4 源地址基础路由实现不同
- 支持 IPV4 映射 IPV6 地址

NAPT 和端口转发 支持 IPv4 网络地址端口转换（NAPT）和端口转发。然而，仅限于单个接口启用 NAPT。

- 要在两个接口之间使用 NAPT 转发数据包，必须在连接到目标网络的接口上启用 NAPT。例如，为了通过 Wi-Fi 接口为以太网流量启用互联网访问，必须在以太网接口上启用 NAPT。
- NAPT 的使用示例可参考 network/vlan_support。

自定义 lwIP 钩子 原始 lwIP 支持通过 `LWIP_HOOK_FILENAME` 实现自定义的编译时修改。ESP-IDF 端口层已使用该文件，但仍支持通过由宏 `ESP_IDF_LWIP_HOOK_FILENAME` 定义的头文件，在 ESP-IDF 中包含并实现自定义添加。以下示例展示了向构建过程添加自定义钩子文件的过程，其中钩子文件名为 `my_hook.h`，位于项目的 `main` 文件夹中：

```
idf_component_get_property(lwip lwip COMPONENT_LIB)
target_compile_options(${lwip} PRIVATE "-I${PROJECT_DIR}/main")
target_compile_definitions(${lwip} PRIVATE "-DESP_IDF_LWIP_HOOK_FILENAME=\"my_hook.
↪h\"")
```

使用 ESP-IDF 构建系统自定义 lwIP 选项 组件配置菜单可以配置常见的 lwIP 选项，但是一些自定义选项需要通过命令行添加。CMake 函数 `target_compile_definitions()` 可以用于定义宏，示例如下：

```
idf_component_get_property(lwip lwip COMPONENT_LIB)
target_compile_definitions(${lwip} PRIVATE "-DETHARP_SUPPORT_VLAN=1")
```

使用这种方法可能无法定义函数式宏。虽然 GCC 支持此类定义，但是未必所有编译器都会接受。为了解决这一限制，可以使用 `add_definitions()` 函数为整个项目定义宏，例如 `add_definitions("-DFALLBACK_DNS_SERVER_ADDRESS(addr)=\"IP_ADDR4((addr),8,8,8,8)\"")`。

另一种方法是在头文件中定义函数式宏，该头文件将预先包含在 lwIP 钩子文件中，请参考*自定义 lwIP 钩子*。

限制

如*适配的 API* 所述，ESP-IDF 中的 lwIP 扩展功能仍然受到全局 DNS 限制的影响。为了在应用程序代码中解决这一限制，可以使用 `FALLBACK_DNS_SERVER_ADDRESS()` 宏定义所有接口能够访问的全局 DNS 备用服务器，或者单独维护每个接口的 DNS 服务器，并在默认接口更改时重新配置。

在 UDP 套接字上重复调用 `send()` 或 `sendto()` 最终可能会导致错误。此时 `errno` 报错为 `ENOMEM`，错误原因是底层网络接口驱动程序中的 buffer 大小有限。当所有驱动程序的传输 buffer 已满时，UDP 传输事务失败。如果应用程序需要发送大量 UDP 数据报，且不希望发送方丢弃数据报，建议检查错误代码，采用短延迟的重传机制。

4.15.7 性能优化

影响 TCP/IP 性能因素较多，可以从多方面进行优化。经调整，ESP-IDF 的默认设置已在 TCP/IP 的吞吐量、响应时间和内存使用间达到平衡。

最大吞吐量

乐鑫使用 iperf 测试应用程序 https://iperf.fr/ 测试了 ESP-IDF 的 TCP/IP 吞吐量。关于实际测试和优化配置的更多信息，请参考*提高网络速度*。

重要： 建议逐步应用更改，并在每次更改后，通过特定应用程序的工作负载检查性能。

- 如果系统中有许多任务抢占 CPU 时间，可以考虑调整 lwIP 任务的 CPU 亲和性 (*CONFIG_LWIP_TCPIP_TASK_AFFINITY*)，并以固定优先级 (18, `ESP_TASK_TCPIP_PRIO`) 运行。为优化 CPU 使用，可以考虑将竞争任务分配给不同核心，或将其优先级调整至较低值。有关内置任务优先级的更多详情，请参阅*内置任务优先级*。
- 如果使用仅带有套接字参数的 `select()` 函数，禁用*CONFIG_VFS_SUPPORT_SELECT* 可以更快地调用 `select()`。
- 如果有足够的空闲 IRAM，可以选择*CONFIG_LWIP_IRAM_OPTIMIZATION* 和*CONFIG_LWIP_EXTRA_IRAM_OPTIMIZATION*，提高 TX/RX 吞吐量。

最低延迟

除增加 buffer 大小外，大多数增加吞吐量的设置会减少 lwIP 函数占用 CPU 的时间，进而降低延迟，缩短响应时间。

- 对于 TCP 套接字，lwIP 支持设置标准的 `TCP_NODELAY` 标记以禁用 Nagle 算法。

最小内存使用

由于 RAM 按需从堆中分配，多数 lwIP 的 RAM 使用也按需分配。因此，更改 lwIP 设置减少 RAM 使用时，或许不会改变空闲时的 RAM 使用量，但可以改变高峰期的 RAM 使用量。

- 减少*CONFIG_LWIP_MAX_SOCKETS* 可以减少系统中的最大套接字数量。更改此设置，会让处于 `WAIT_CLOSE` 状态的 TCP 套接字在需要打开新套接字时更快地关闭和复用，进一步降低峰值 RAM 使用量。

- 减少*CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*、*CONFIG_LWIP_TCP_RECVMBOX_SIZE*和*CONFIG_LWIP_UDP_RECVMBOX_SIZE* 可以减少 RAM 使用量，但会影响吞吐量，具体取决于使用情况。
- 减少*CONFIG_LWIP_TCP_ACCEPTMBOX_SIZE* 可以通过限制同时接受的连接数来减少 RAM 使用量。
- 减少*CONFIG_LWIP_TCP_MSL* 和*CONFIG_LWIP_TCP_FIN_WAIT_TIMEOUT* 可以减少系统中的最大分段寿命，同时会使处于 `TIME_WAIT` 和 `FIN_WAIT_2` 状态的 TCP 套接字能更快地关闭和复用。
- 禁用*CONFIG_LWIP_IPV6* 可以在系统启动时节省大约 39 KB 的固件大小和 2 KB 的 RAM，并在运行 TCP/IP 栈时节省 7 KB 的 RAM。如果无需支持 IPV6，可以禁用 IPv6，减少 flash 和 RAM 占用。
- 禁用*CONFIG_LWIP_IPV4* 可以在系统启动时节省大约 26 KB 的固件大小和 600 B 的 RAM，并在运行 TCP/IP 栈时节省 6 KB 的 RAM。如果本地网络仅支持 IPv6 配置，可以禁用 IPv4，减少 flash 和 RAM 占用。

最大 buffer 使用 lwIP 消耗的最大堆内存即 lwIP 驱动程序 **理论上可能消耗的最大内存**，通常取决于以下因素：

- 创建 UDP 连接所需的内存：`lwip_udp_conn`
- 创建 TCP 连接所需的内存：`lwip_tcp_conn`
- 应用程序拥有的 UDP 连接数量：`lwip_udp_con_num`
- 应用程序拥有的 TCP 连接数量：`lwip_tcp_con_num`
- TCP 的 TX 窗口大小：`lwip_tcp_tx_win_size`
- TCP 的 RX 窗口大小：`lwip_tcp_rx_win_size`

因此，lwIP 消耗的最大堆内存可以用以下公式计算： lwip_dynamic_peek_memory = (lwip_udp_con_num * lwip_udp_conn) + (lwip_tcp_con_num * (lwip_tcp_tx_win_size + lwip_tcp_rx_win_size + lwip_tcp_conn))

某些基于 TCP 的应用程序只需要一个 TCP 连接。然而，当出现错误（如发送失败）时，应用程序可能会关闭此 TCP 连接，并创建一个新的连接。根据 TCP 状态机和 RFC793，关闭 TCP 连接可能需要很长时间，这可能导致系统中同时存在多个 TCP 连接。

4.16 存储器类型

ESP32-P4 芯片具有不同类型的存储器和灵活的存储器映射特性，本小节将介绍 ESP-IDF 默认如何使用这些功能。

ESP-IDF 区分了指令总线（IRAM、IROM、RTC FAST memory）和数据总线 (DRAM、DROM)。指令存储器是可执行的，只能通过 4 字节对齐字读取或写入。数据存储器不可执行，可以通过单独的字节操作访问。有关总线的更多信息，请参阅 *ESP32-P4 技术参考手册* > *系统和存储器* [PDF]。

4.16.1 DRAM（数据 RAM）

非常量静态数据（.data 段）和零初始化数据（.bss 段）由链接器放入内部 SRAM 作为数据存储。此区域中的剩余空间可在程序运行时用作堆。

通过应用 `EXT_RAM_BSS_ATTR` 宏，零初始化数据也可以放入外部 RAM。使用这个宏需要启用*CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY*。详情请见*允许.bss 段放入片外存储器*。

备注： 静态分配的 DRAM 的最大值也会因编译应用程序的*IRAM（指令 RAM）* 大小而减小。运行时可用的堆内存会因应用程序的总静态 IRAM 和 DRAM 使用而减少。

常量数据也可能被放入 DRAM，例如当它被用于 non-flash-safe ISR 时（具体请参考*如何将代码放入 IRAM*）。

"noinit" DRAM

可以将 __NOINIT_ATTR 宏用作属性，从而将数据放入 .noinit 部分。放入该部分的值在启动时不会被初始化，在软件重启后也会保持值不变。

通过使用 EXT_RAM_NOINIT_ATTR 宏，noinit 数据也可以放入外部 RAM 中。为此，需要启用*CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY*，可参考*允许.noinit 段放入片外存储器*。如果没有启用*CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY*，EXT_RAM_NOINIT_ATTR 会和 __NOINIT_ATTR 一样，将数据放入内部 RAM 的 .noinit 部分。

示例:

```
__NOINIT_ATTR uint32_t noinit_data;
```

4.16.2 IRAM（指令 RAM）

备注: 内部 SRAM 中不用于指令 RAM 的部分都会作为*DRAM（数据 RAM）*供静态数据和动态分配（堆）使用。

何时需要将代码放入 IRAM

以下情况时应将部分应用程序放入 IRAM:

- 如果在注册中断处理程序时使用了 ESP_INTR_FLAG_IRAM，则中断处理程序必须要放入 IRAM。更多信息可参考*IRAM 安全中断处理程序*。
- 可将一些时序关键代码放入 IRAM，以减少从 flash 中加载代码造成的相关损失。ESP32-P4 通过 MMU 缓存从 flash 中读取代码和数据。在某些情况下，将函数放入 IRAM 可以减少由缓存未命中造成的延迟，从而显著提高函数的性能。

如何将代码放入 IRAM

借助链接器脚本，一些代码会被自动放入 IRAM 区域中。

如果需要将某些特定的应用程序代码放入 IRAM，可以使用*链接器脚本生成机制* 功能并在组件中添加链接器脚本片段文件，在该片段文件中，可以给整个目标源文件或其中的个别函数打上 noflash 标签。更多信息可参考*链接器脚本生成机制*。

或者，也可以通过使用 IRAM_ATTR 宏在源代码中指定需要放入 IRAM 的代码:

```
#include "esp_attr.h"

void IRAM_ATTR gpio_isr_handler(void* arg)
{
    // ...
}
```

放入 IRAM 后可能会导致 IRAM 安全中断处理程序出现问题:

- IRAM_ATTR 函数中的字符串或常量可能没有自动放入 RAM 中，这时可以使用 DRAM_ATTR 属性进行标记，或者也可以使用链接器脚本方法将它们自动放入 RAM 中。

```
void IRAM_ATTR gpio_isr_handler(void* arg)
{
   const static DRAM_ATTR uint8_t INDEX_DATA[] = { 45, 33, 12, 0 };
   const static char *MSG = DRAM_STR("I am a string stored in RAM");
}
```

注意，具体哪些数据需要被标记为 DRAM_ATTR 可能很难确定。如果没有被标记为 DRAM_ATTR，某些变量或表达式有时会被编译器别为常量（即使它们没有被标记为 const）并将其放入 flash 中。

- GCC 的优化会自动生成跳转表或 switch/case 查找表，并将这些表放在 flash 中。IDF 默认在编译所有文件时使用 `-fno-jump-tables -fno-tree-switch-conversion` 标志来避免这种情况。

可以为不需要放置在 IRAM 中的单个源文件重新启用跳转表优化。关于如何在编译单个源文件时添加 `-fno-jump-tables -fno-tree-switch-conversion` 选项，请参考组件编译控制。

4.16.3 IROM（代码从 flash 中运行）

如果一个函数没有被显式地声明放在 IRAM 或者 RTC 存储器中，则它会放在 flash 中。由于 IRAM 空间有限，应用程序的大部分二进制代码都需要放入 IROM 中。

在*启动* 过程中，从 IRAM 中运行的引导加载程序配置 MMU flash 缓存，将应用程序的指令代码区域映射到指令空间。通过 MMU 访问的 flash 使用一些内部 SRAM 进行缓存，访问缓存的 flash 数据与访问其他类型的内部存储器一样快。

4.16.4 DROM（数据存储在 flash 中）

默认情况下，链接器将常量数据放入一个映射到 MMU flash 缓存的区域中。这与*IROM（代码从 flash 中运行）* 部分相同，但此处用于只读数据而不是可执行代码。

唯一没有默认放入 DROM 的常量数据是被编译器嵌入到应用程序代码中的字面常量。这些被放置在周围函数的可执行指令中。

DRAM_ATTR 属性可以用来强制将常量从 DROM 放入*DRAM（数据 RAM）* 部分（见上文）。

4.16.5 RTC FAST memory（RTC 快速存储器）

除非禁用*CONFIG_ESP_SYSTEM_ALLOW_RTC_FAST_MEM_AS_HEAP* 选项，否则剩余的 RTC FAST memory 会被添加到堆中。该部分内存可以和*DRAM（数据 RAM）* 互换使用，但是访问速度稍慢一点。

4.16.6 紧密耦合内存 (TCM)

TCM 是靠近 CPU 放置的内存，支持在 CPU 频率下直接访问，无需通过 cache。虽然在一般情况下，TCM 的效率或速度相较 cache 偏低，但是访问 TCM 所需的时间是可以预测且始终一致的。具有稳定的访问速度对于时间关键型例程来说十分重要，因此 TCM 对于此类例程而言非常有用。

4.16.7 具备 DMA 功能

大多数的 DMA 控制器（比如 SPI、sdmmc 等）都要求发送/接收缓冲区放在 DRAM 中，并且按字对齐。我们建议将 DMA 缓冲区放在静态变量而不是堆栈中。使用 DMA_ATTR 宏可以声明该全局/本地的静态变量具备 DMA 功能，例如:

```
DMA_ATTR uint8_t buffer[]="I want to send something";

void app_main()
{
    // 初始化代码
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8 * sizeof(buffer),
    };
    spi_device_transmit(spi, &temp);
    // 其它程序
}
```

或者:

```
void app_main()
{
    DMA_ATTR static uint8_t buffer[] = "I want to send something";
    // 初始化代码
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8 * sizeof(buffer),
    };
    spi_device_transmit(spi, &temp);
    // 其它程序
}
```

也可以通过使用*MALLOC_CAP_DMA* 标志来动态分配具备 DMA 能力的内存缓冲区。

4.16.8 在堆栈中放置 DMA 缓冲区

可以在堆栈中放置 DMA 缓冲区，但建议尽量避免。如果实在有需要的话，请注意以下几点:

- 如果堆栈在 PSRAM 中，则不建议将 DRAM 缓冲区放在堆栈上。如果任务堆栈在 PSRAM 中，则必须执行*片外 RAM* 中描述的几个步骤。
- 在函数中使用 `WORD_ALIGNED_ATTR` 宏来修饰变量，将其放在适当的位置上，比如:

```
void app_main()
{
    uint8_t stuff;
    WORD_ALIGNED_ATTR uint8_t buffer[] = "I want to send something";    //否则␣
↪buffer 会被存储在 stuff 变量后面
    // 初始化代码
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8 * sizeof(buffer),
    };
    spi_device_transmit(spi, &temp);
    // 其它程序
}
```

4.17 OpenThread

OpenThread 是在 802.15.4 MAC 层上运行的 IP 协议栈，支持 mesh 网络，具有低功耗特性。

4.17.1 OpenThread 协议栈运行模式

在乐鑫的芯片上，OpenThread 可按以下模式运行:

独立节点模式

在此模式下，完整的 OpenThread 协议栈及其应用层在同一芯片上运行，适用于支持 15.4 无线通信协议的芯片，如 ESP32-H2, ESP32-C6。

无线协处理器 (RCP) 模式

在此模式下，芯片通过连接到运行 OpenThread IP 协议栈的另一个主机，代表主机发送和接收 15.4 数据包。该模式适用于支持 15.4 无线通信协议的芯片，如 ESP32-H2, ESP32-C6。对于芯片和主机之间的通信方式，目前 ESP-IDF 支持 SPI 或 UART。考虑到传输延迟，建议使用 SPI。

OpenThread 主机模式

在此模式下，不支持 15.4 无线通信协议的芯片可以连接到 RCP，并在主机模式下运行 OpenThread。这种模式支持在 Wi-Fi 芯片上（如 ESP32、ESP32-S2、ESP32-S3 和 ESP32-C3 等）运行 OpenThread。下图展示了设备在不同模式下的工作方式：

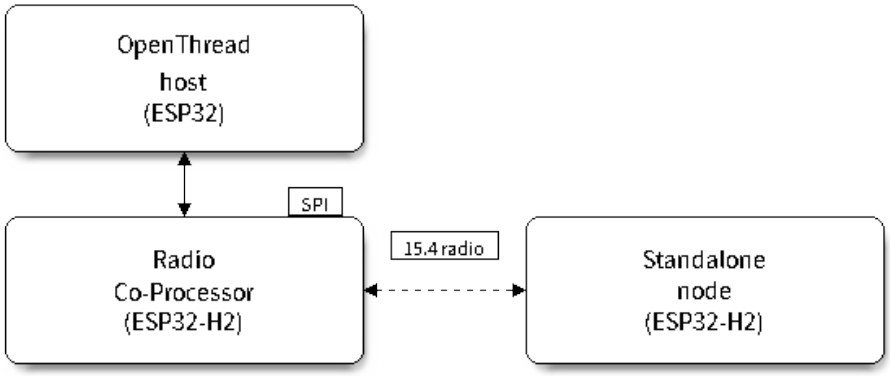


图 27: OpenThread 设备模式

4.17.2 编写 OpenThread 应用程序

要学习编写 OpenThread 应用程序，可以从 OpenThread 应用示例 openthread/ot_cli 开始。该示例中展示了简单的 OpenThread 网络组网流程，以及在 Openthread 网络上，如何实现基于套接字的服务器和客户端之间的简单通信。

初始化 OpenThread 协议栈所需的准备

- s1.1：主任务调用 `esp_vfs_eventfd_register()`，初始化 eventfd 虚拟文件系统。eventfd 文件系统用于实现 OpenThread 协议栈中的任务通知。
- s1.2：主任务调用 `nvs_flash_init()` 初始化 NVS，即 Thread 网络数据的存储位置。
- s1.3：**可选**。主任务调用 `esp_netif_init()`，为 Thread 创建网络接口。
- s1.4：主任务调用 `esp_event_loop_create()` 创建系统事件任务，并初始化应用程序事件的回调函数。

OpenThread 协议栈初始化

- s2.1：调用 `esp_openthread_init()` 初始化 OpenThread 协议栈。

OpenThread 网络接口初始化

以下为 **可选**步骤，仅在应用程序需为 Thread 创建网络接口时使用。

- s3.1：使用 `ESP_NETIF_DEFAULT_OPENTHREAD` 调用 `esp_netif_new()`，创建网络接口。

- s3.2：调用*esp_openthread_netif_glue_init()*，创建 OpenThread 网络接口处理程序。
- s3.3：调用*esp_netif_attach()* 将处理程序附加到网络接口。

OpenThread 主循环

- s4.3：调用*esp_openthread_launch_mainloop()* 启动 OpenThread 主循环。注意，OpenThread 主循环属于忙等循环，仅在 OpenThread 协议栈终止后返回。

调用 OpenThread API

OpenThread API 非线程安全。当从其他任务中调用 OpenThread API 时，请确保以*esp_openthread_lock_acquire()* 获取锁，并在之后以*esp_openthread_lock_release()* 释放锁。

卸载 Openthread 协议栈

要在应用程序中卸载 OpenThread 协议栈，请遵循以下步骤：

- 如果创建了 OpenThread 网络接口，请调用*esp_netif_destroy()* 和*esp_openthread_netif_glue_deinit()* 卸载 OpenThread 协议栈。
- 调用*esp_openthread_deinit()* 卸载 OpenThread 协议栈。

4.17.3 OpenThread 边界路由器

OpenThread 边界路由器连接了 Thread 网络和其他 IP 网络，提供 IPv6 连通性、服务注册和委托功能。

要在 ESP 芯片上启用 OpenThread 边界路由器，需要将 RCP 连接到具备 Wi-Fi 功能的芯片上，如 ESP32。

在初始化过程中，调用*esp_openthread_border_router_init()* 会启用所有边界路由功能。

要了解更多有关边界路由器的详细信息，请参阅 openthread/ot_br 示例和其中的 README 文件。

警告： This document is not updated for ESP32P4 yet, so some of the content may not be correct.

This warning was automatically inserted due to the source file being in the *add_warnings_pages* list.

4.18 分区表

4.18.1 概述

每片 ESP32-P4 的 flash 可以包含多个应用程序，以及多种不同类型的数据（例如校准数据、文件系统数据、参数存储数据等）。因此，我们在 flash 的默认偏移地址 0x8000 处烧写一张分区表。

分区表的长度为 0xC00 字节，最多可以保存 95 条分区表条目。MD5 校验和附加在分区表之后，用于在运行时验证分区表的完整性。分区表占据了整个 flash 扇区，大小为 0x1000 (4 KB)。因此，它后面的任何分区至少需要位于 (默认偏移地址) + 0x1000 处。

分区表中的每个条目都包括以下几个部分：Name（标签）、Type（app、data 等）、SubType 以及在 flash 中的偏移量（分区的加载地址）。

在使用分区表时，最简单的方法就是打开项目配置菜单（`idf.py menuconfig`），并在*CONFIG_PARTITION_TABLE_TYPE* 下选择一个预定义的分区表：

- ”Single factory app, no OTA”
- ”Factory app, two OTA definitions”

在以上两种选项中，出厂应用程序均将被烧录至 flash 的 0x10000 偏移地址处。这时，运行 *idf.py partition-table* ，即可以打印当前使用分区表的信息摘要。

4.18.2 内置分区表

以下是”Single factory app, no OTA” 选项的分区表信息摘要:

```
# ESP-IDF Partition Table
# Name,    Type, SubType, Offset,  Size,    Flags
nvs,       data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,   app,  factory, 0x10000, 1M,
```

- flash 的 0x10000 (64 KB) 偏移地址处存放一个标记为”factory” 的二进制应用程序，且启动加载器将默认加载这个应用程序。
- 分区表中还定义了两个数据区域，分别用于存储 NVS 库专用分区和 PHY 初始化数据。

以下是”Factory app, two OTA definitions” 选项的分区表信息摘要:

```
# ESP-IDF Partition Table
# Name,    Type, SubType, Offset,  Size, Flags
nvs,       data, nvs,     0x9000,  0x4000,
otadata,   data, ota,     0xd000,  0x2000,
phy_init, data, phy,     0xf000,  0x1000,
factory,   app,  factory, 0x10000,  1M,
ota_0,     app,  ota_0,   0x110000, 1M,
ota_1,     app,  ota_1,   0x210000, 1M,
```

- 分区表中定义了三个应用程序分区，这三个分区的类型都被设置为 “app”，但具体 app 类型不同。其中，位于 0x10000 偏移地址处的为出厂应用程序 (factory)，其余两个为 OTA 应用程序（ota_0，ota_1）。
- 新增了一个名为 “otadata” 的数据分区，用于保存 OTA 升级时需要的数据。启动加载器会查询该分区的数据，以判断该从哪个 OTA 应用程序分区加载程序。如果 “otadata” 分区为空，则会执行出厂程序。

4.18.3 创建自定义分区表

如果在 `menuconfig` 中选择了 “Custom partition table CSV”，则还需要输入该分区表的 CSV 文件在项目中的路径。CSV 文件可以根据需要，描述任意数量的分区信息。

CSV 文件的格式与上面摘要中打印的格式相同，但是在 CSV 文件中并非所有字段都是必需的。例如下面是一个自定义的 OTA 分区表的 CSV 文件:

```
# Name,    Type, SubType,  Offset,   Size,  Flags
nvs,       data, nvs,      0x9000,  0x4000
otadata,   data, ota,      0xd000,  0x2000
phy_init, data, phy,      0xf000,  0x1000
factory,   app,  factory,  0x10000,  1M
ota_0,     app,  ota_0,    ,         1M
ota_1,     app,  ota_1,    ,         1M
nvs_key,   data, nvs_keys, ,        0x1000
```

- 字段之间的空格会被忽略，任何以 # 开头的行（注释）也会被忽略。
- CSV 文件中的每个非注释行均为一个分区定义。
- 每个分区的 `Offset` 字段可以为空，`gen_esp32part.py` 工具会从分区表位置的后面开始自动计算并填充该分区的偏移地址，同时确保每个分区的偏移地址正确对齐。

Name 字段

Name 字段可以是任何有意义的名称，但不能超过 16 个字节，其中包括一个空字节（之后的内容将被截断）。该字段对 ESP32-P4 并不是特别重要。

Type 字段

Type 字段可以指定为 app (0x00) 或者 data (0x01)，也可以直接使用数字 0-254（或者十六进制 0x00-0xFE）。注意，0x00-0x3F 不得使用（预留给 esp-idf 的核心功能）。

如果你的应用程序需要以 ESP-IDF 尚未支持的格式存储数据，请在 0x40-0xFE 内添加一个自定义分区类型。

参考*esp_partition_type_t* 关于 app 和 data 分区的枚举定义。

如果用 C++ 编写，那么指定一个应用程序定义的分区类型，需要在*esp_partition_type_t* 中使用整数，从而与*分区 API* 一起使用。例如:

```
static const esp_partition_type_t APP_PARTITION_TYPE_A = (esp_partition_type_
↪t)0x40;
```

注意，启动加载器将忽略 app (0x00) 和 data (0x01) 以外的其他分区类型。

SubType 字段

SubType 字段长度为 8 bit，内容与具体分区 Type 有关。目前，esp-idf 仅仅规定了 “app” 和 “data” 两种分区类型的子类型含义。

参考*esp_partition_subtype_t*，以了解 ESP-IDF 定义的全部子类型列表，包括:

- 当 Type 定义为 app 时，SubType 字段可以指定为 factory (0x00)、ota_0 (0x10) ···ota_15 (0x1F) 或者 test (0x20)。
  - factory (0x00) 是默认的 app 分区。启动加载器将默认加载该应用程序。但如果存在类型为 data/ota 分区，则启动加载器将加载 data/ota 分区中的数据，进而判断启动哪个 OTA 镜像文件。
    - OTA 升级永远都不会更新 factory 分区中的内容。
    - 如果你希望在 OTA 项目中预留更多 flash，可以删除 factory 分区，转而使用 ota_0 分区。
  - ota_0 (0x10) ···ota_15 (0x1F) 为 OTA 应用程序分区，启动加载器将根据 OTA 数据分区中的数据来决定加载哪个 OTA 应用程序分区中的程序。在使用 OTA 功能时，应用程序应至少拥有 2 个 OTA 应用程序分区（ota_0 和 ota_1）。更多详细信息，请参考*OTA 文档* 。
  - test (0x20) 为预留的子类型，用于工厂测试流程。如果没有其他有效 app 分区，test 将作为备选启动分区使用。也可以配置启动加载器在每次启动时读取 GPIO，如果 GPIO 被拉低则启动该分区。详细信息请查阅*从测试固件启动*。
- 当 Type 定义为 data 时，SubType 字段可以指定为 ota (0x00)、phy (0x01)、nvs (0x02)、nvs_keys (0x04) 或者其他组件特定的子类型（请参考*子类型枚举*).
  - ota (0) 即*OTA 数据分区* ，用于存储当前所选的 OTA 应用程序的信息。这个分区的大小需要设定为 0x2000。更多详细信息，请参考*OTA 文档* 。
  - phy (1) 分区用于存放 PHY 初始化数据，从而保证可以为每个设备单独配置 PHY，而非必须采用固件中的统一 PHY 初始化数据。
    - 默认配置下，phy 分区并不启用，而是直接将 phy 初始化数据编译至应用程序中，从而节省分区表空间（直接将此分区删掉）。
    - 如果需要从此分区加载 phy 初始化数据，请打开项目配置菜单（idf.py menuconfig），并且使能 NOT UPDATED YET 选项。此时，还需要手动将 phy 初始化数据烧至设备 flash（esp-idf 编译系统并不会自动完成该操作）。
  - nvs (2) 是专门给*非易失性存储 (NVS) API* 使用的分区。
    - 用于存储每台设备的 PHY 校准数据（注意，并不是 PHY 初始化数据）。
    - NVS API 还可以用于其他应用程序数据。
    - 强烈建议为 NVS 分区分配至少 0x3000 字节空间。
    - 如果使用 NVS API 存储大量数据，请增加 NVS 分区的大小（默认是 0x6000 字节）。
  - nvs_keys (4) 是 NVS 秘钥分区。详细信息，请参考*非易失性存储 (NVS) API* 文档。

        * 用于存储加密密钥（如果启用了 *NVS 加密*功能）。
        * 此分区应至少设定为 4096 字节。
    - ESP-IDF 还支持其他用于数据存储的预定义子类型，包括：
        * `coredump` (0x03) 用于在使用自定义分区表 CSV 文件时存储核心转储，详情请参阅*核心转储*。
        * `efuse` (0x05) 使用*虚拟 eFuse* 模拟 eFuse 位。
        * `undefined` (0x06) 隐式用于未指定子类型（即子类型为空）的数据分区，但也可显式将其标记为未定义。
        * `fat` (0x81) 用于*FAT 文件系统*。
        * `spiffs` (0x82) 用于*SPIFFS 文件系统*。
        * `littlefs` (0x83) 用于 LittleFS 文件系统，详情可参阅 storage/littlefs 示例。
- 如果分区类型是由应用程序定义的任意值 (0x40-0xFE)，那么 `subtype` 字段可以是由应用程序选择的任何值 (0x00-0xFE)。
  请注意，如果用 C++ 编写，应用程序定义的子类型值需要转换为`esp_partition_type_t`，从而与*分区 API* 一起使用。

额外分区 SubType 字段

组件可以通过设置 `EXTRA_PARTITION_SUBTYPES` 属性来定义额外的分区子类型。`EXTRA_PARTITION_SUBTYPES` 是一个 CMake 列表，其中的每个条目由字符串组成，以逗号为分隔，格式为 `<type>, <subtype>, <value>`。构建系统通过该属性会自动添加额外的子类型，并在`esp_partition_subtype_t` 中插入名为 `ESP_PARTITION_SUBTYPE_<type>_<subtype>` 的字段。项目可以使用这个子类型来定义分区表 CSV 文件中的分区，并使用`esp_partition_subtype_t` 中的新字段。

偏移地址 (Offset) 和大小 (Size) 字段

- 偏移地址表示 SPI flash 中的分区地址，扇区大小为 0x1000 (4 KB)。因此，偏移地址必须是 4 KB 的倍数。
- 若 CSV 文件中的分区偏移地址为空，则该分区会接在前一个分区之后；若为首个分区，则将接在分区表之后。
- `app` 分区的偏移地址必须与 0x10000 (64 KB) 对齐。如果偏移字段留空，则 `gen_esp32part.py` 工具会自动计算得到一个满足对齐要求的偏移地址。如果 `app` 分区的偏移地址没有与 0x10000 (64 KB) 对齐，则该工具会报错。
- `app` 分区的大小必须与 flash 扇区大小对齐。为 `app` 分区指定未对齐的大小将返回错误。
- `app` 分区的大小和偏移地址可以采用十进制数或是以 0x 为前缀的十六进制数，且支持 K 或 M 的倍数单位（K 和 M 分别代表 1024 和 1024*1024 字节）。

如果你希望允许分区表中的分区采用任意起始偏移量 (*CONFIG_PARTITION_TABLE_OFFSET*)，请将分区表（CSV 文件）中所有分区的偏移字段都留空。注意，此时，如果你更改了分区表中任意分区的偏移地址，则其他分区的偏移地址也会跟着改变。这种情况下，如果你之前还曾设定某个分区采用固定偏移地址，则可能造成分区表冲突，从而导致报错。

Flags 字段

目前支持 `encrypted` 和 `readonly` 标记：

- 如果 Flags 字段设置为 `encrypted`，且已启用*flash 加密* 功能，则该分区将会被加密。

备注： 无论是否设置 Flags 字段，`app` 分区都将保持加密。

- 如果 Flags 字段设置为 `readonly`，则该分区为只读分区。`readonly` 标记仅支持除 `ota` 和 `coredump` 子类型外的 `data` 分区。使用该标记，防止意外写入如出厂数据分区等包含关键设备特定配置数据的分区。

> **备注：** 在任何写入模式下 (`w`、`w+`、`a`、`a+`、`r+`)，尝试通过 C 文件 I/O API 打开文件 (`fopen`) 的操作都将失败并返回 `NULL`。除 `O_RDONLY` 外，`open` 与任何标志一同使用都将失败并返回 `-1`，全局变量 `errno` 也将设置为 `EROFS`。上述情况同样适用于通过其他 POSIX 系统调用函数执行写入或擦除的操作。在只读分区上，以读写模式打开 NVS 的句柄将失败并返回*ESP_ERR_NOT_ALLOWED* 错误代码，使用 `esp_partition` 或 `spi_flash` 等较低级别的 API 进行写入操作也将返回*ESP_ERR_NOT_ALLOWED* 错误代码。

可以使用冒号连接不同的标记，来同时指定多个标记，如 `encrypted:readonly`。

4.18.4 生成二进制分区表

烧写到 ESP32-P4 中的分区表采用二进制格式，而不是 CSV 文件本身。此时，partition_table/gen_esp32part.py 工具可以实现 CSV 和二进制文件之间的转换。

如果你在项目配置菜单（`idf.py menuconfig`）中设置了分区表 CSV 文件的名称，然后构建项目或执行 `idf.py partition-table`。这时，转换将在编译过程中自动完成。

手动将 CSV 文件转换为二进制文件:

```
python gen_esp32part.py input_partitions.csv binary_partitions.bin
```

手动将二进制文件转换为 CSV 文件:

```
python gen_esp32part.py binary_partitions.bin input_partitions.csv
```

在标准输出 (stdout) 上，打印二进制分区表的内容（运行 `idf.py partition-table` 时展示的信息摘要也是这样生成的）：

```
python gen_esp32part.py binary_partitions.bin
```

4.18.5 分区大小检查

ESP-IDF 构建系统将自动检查生成的二进制文件大小与可用的分区大小是否匹配，如果二进制文件太大，则会构建失败并报错。

目前会对以下二进制文件进行检查：

- 引导加载程序的二进制文件的大小要适合分区表前的区域大小（分区表前的区域都分配给了引导加载程序），具体请参考*引导加载程序大小*。
- 应用程序二进制文件应至少适合一个 “app” 类型的分区。如果不适合任何应用程序分区，则会构建失败。如果只适合某些应用程序分区，则会打印相关警告。

> **备注：** 即使分区大小检查返回错误并导致构建失败，仍然会生成可以烧录的二进制文件（它们对于可用空间来说过大，因此无法正常工作）。

MD5 校验和

二进制格式的分区表中含有一个 MD5 校验和。这个 MD5 校验和是根据分区表内容计算的，可在设备启动阶段，用于验证分区表的完整性。

用户可通过 `gen_esp32part.py` 的 `--disable-md5sum` 选项或者*CONFIG_PARTITION_TABLE_MD5* 选项关闭 MD5 校验。

4.18.6 烧写分区表

- `idf.py partition-table-flash`：使用 esptool.py 工具烧写分区表。
- `idf.py flash`：会烧写所有内容，包括分区表。

在执行 `idf.py partition-table` 命令时，手动烧写分区表的命令也将打印在终端上。

备注： 分区表的更新并不会擦除根据旧分区表存储的数据。此时，可以使用 `idf.py erase-flash` 命令或者 `esptool.py erase_flash` 命令来擦除 flash 中的所有内容。

4.18.7 分区工具 (`parttool.py`)

partition_table 组件中有分区工具 parttool.py，可以在目标设备上完成分区相关操作。该工具有如下用途：

- 读取分区，将内容存储到文件中 (read_partition)
- 将文件中的内容写至分区 (write_partition)
- 擦除分区 (erase_partition)
- 检索特定分区的名称、偏移、大小和 flag（“加密”）标志等信息 (get_partition_info)

用户若想通过编程方式完成相关操作，可从另一个 Python 脚本导入并使用分区工具，或者从 Shell 脚本调用分区工具。前者可使用工具的 Python API，后者可使用命令行界面。

Python API

首先请确保已导入 *parttool* 模块。

```
import sys
import os

idf_path = os.environ["IDF_PATH"]  # 从环境中获取 IDF_PATH 的值
parttool_dir = os.path.join(idf_path, "components", "partition_table")  # parttool.
↪py 位于 $IDF_PATH/components/partition_table 下

sys.path.append(parttool_dir)  # 使能 Python 寻找 parttool 模块
from parttool import *  # 导入 parttool 模块内的所有名称
```

要使用分区工具的 Python API，第一步是创建 *ParttoolTarget*：

```
# 创建 parttool.py 的目标设备，并将目标设备连接到串行端口 /dev/ttyUSB1
target = ParttoolTarget("/dev/ttyUSB1")
```

现在，可使用创建的 *ParttoolTarget* 在目标设备上完成操作：

```
# 擦除名为 'storage' 的分区
target.erase_partition(PartitionName("storage"))

# 读取类型为 'data'、子类型为 'spiffs' 的分区，保存至文件 'spiffs.bin'
target.read_partition(PartitionType("data", "spiffs"), "spiffs.bin")

# 将 'factory.bin' 文件的内容写至 'factory' 分区
target.write_partition(PartitionName("factory"), "factory.bin")

# 打印默认启动分区的大小
storage = target.get_partition_info(PARTITION_BOOT_DEFAULT)
print(storage.size)
```

使用 *PartitionName*、*PartitionType* 或 PARTITION_BOOT_DEFAULT 指定要操作的分区。顾名思义，这三个参数可以指向拥有特定名称的分区、特定类型和子类型的分区或默认启动分区。

更多关于 Python API 的信息，请查看分区工具的代码注释。

命令行界面

parttool.py 的命令行界面具有如下结构：

```
parttool.py [command-args] [subcommand] [subcommand-args]

- command-args - 执行主命令 (parttool.py) 所需的实际参数，多与目标设备有关
- subcommand - 要执行的操作
- subcommand-args - 所选操作的实际参数
```

```
# 擦除名为 'storage' 的分区
parttool.py --port "/dev/ttyUSB1" erase_partition --partition-name=storage

# 读取类型为 'data'、子类型为 'spiffs' 的分区，保存到 'spiffs.bin' 文件
parttool.py --port "/dev/ttyUSB1" read_partition --partition-type=data --partition-
↪subtype=spiffs --output "spiffs.bin"

# 将 'factory.bin' 文件中的内容写入到 'factory' 分区
parttool.py --port "/dev/ttyUSB1" write_partition --partition-name=factory --input
↪"factory.bin"

# 打印默认启动分区的大小
parttool.py --port "/dev/ttyUSB1" get_partition_info --partition-boot-default --
↪info size
```

更多信息可用 *--help* 指令查看：

```
# 显示可用的子命令和主命令描述
parttool.py --help

# 显示子命令的描述
parttool.py [subcommand] --help
```

4.19 性能

ESP-IDF 预设了默认设置，旨在在性能、资源使用和可用功能之间进行权衡。

本指南详述了如何优化固件应用程序，以提高特定方面的性能。这一过程通常涉及到性能之间的权衡，例如限制部分可用功能，或者牺牲某性能以满足另一性能的要求，例如通过使用更多内存换取更快的运行速度。

4.19.1 如何优化性能

1. 确定应用程序的关键性能要求，例如控制某个网络操作的响应时间、控制启动时间、或在某个外设中实现特定的数据吞吐量。
2. 确定衡量该性能的方法（下文指南中概述了一些方法）。
3. 修改代码和项目配置，比较新旧测量结果。
4. 重复第三步，直到性能达到第一步中设定的要求为止。

4.19.2 指南

速度优化

概述 提高代码执行速度是提升软件性能的关键要素，该优化也可能带来其他积极影响，比如降低总体功耗。然而，提高代码执行速度可能会牺牲其他性能，如最小化二进制文件大小 。

决定优化目标 如果应用程序固件中的某个函数仅每周在后台执行一次，其执行时间是 10 ms 还是 100 ms 对整体性能的影响或可忽略不计。但如果某个函数以 10 Hz 的频率持续执行，其执行时间是 10 ms 还是 100 ms 就会对系统性能产生显著影响。

大多数应用程序固件中，只有一小部分函数需要优化性能，例如频繁执行的函数，或者必须满足应用程序对延迟或吞吐量的要求的函数。应针对这些特定函数优化其执行速度。

测量性能 想要提升某方面性能，首先要对其进行测量。

基本性能测量方法 可以直接测量与外部交互的性能，例如，测量一般网络性能可以参考 wifi/iperf 和 ethernet/iperf ，或者使用示波器或逻辑分析仪来测量与设备外设的交互时间。

此外，另一种测量性能的方法是在代码中添加计时测量：

```
#include "esp_timer.h"

void measure_important_function(void) {
    const unsigned MEASUREMENTS = 5000;
    uint64_t start = esp_timer_get_time();

    for (int retries = 0; retries < MEASUREMENTS; retries++) {
        important_function(); // 需要测量的代码
    }

    uint64_t end = esp_timer_get_time();

    printf("%u iterations took %llu milliseconds (%llu microseconds per␣
↪invocation)\n",
           MEASUREMENTS, (end - start)/1000, (end - start)/MEASUREMENTS);
}
```

通过多次执行目标代码可以降低其他因素的影响，例如实时操作系统 (RTOS) 的上下文切换、测量的开销等。

- 使用*esp_timer_get_time()* 可以生成微秒级精度的“墙钟”时间戳，但每次调用计时函数都会产生适量开销。
- 也可以使用标准 Unix 函数 `gettimeofday()` 和 `utime()` 来进行计时测量，尽管其开销略高一些。
- 此外，代码中包含 `hal/cpu_hal.h` 头文件，并调用 HAL 函数 `cpu_hal_get_cycle_count()` 可以返回已执行的 CPU 循环数。该函数开销较低，适用于高精度测量执行时间极短的代码。CPU 周期是各内核独立计数的，因此本方法仅适用于测量中断处理程序或固定在单个核上的任务。
- 在执行“微基准测试”时（即仅对运行时间不到 1-2 ms 的小代码段进行基准测试），二进制文件会影响 flash 缓存的性能，进而可能会导致计时测量出现较大差异。这是因为二进制布局可能会导致在特定的执行顺序中产生不同模式的缓存缺失。执行较大测试代码通常可以抵消这种影响。在基准测试时多次执行一个小函数可以减少 flash 缓存缺失的影响。另外，将该代码移到 IRAM 中（参见针对性优化 ）也可以解决这个问题。

外部跟踪 *应用层跟踪库* 可以在几乎不影响代码执行的情况下测量其执行速度。

任务 如果启用了选项*CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS* ，则可以使用 FreeRTOS API *vTaskGetRunTimeStats()* 来获取各个 FreeRTOS 任务运行时占用处理器的时间。

SEGGER SystemView 是一款出色的工具，可将任务执行情况可视化，也可用于排查系统整体的性能问题或改进方向。

提高整体速度 以下优化措施将提高几乎所有代码的执行效果，包括启动时间、吞吐量、延迟等：

- 设置*CONFIG_ESPTOOLPY_FLASHMODE* 为 QIO 或 QOUT 模式（四线 I/O 模式）。相较于默认的 DIO 模式，在这两种模式下，从 flash 加载或执行代码的速度几乎翻倍。如果两种模式都支持，QIO 会稍微快于 QOUT。请注意，flash 芯片以及 ESP32-P4 与 flash 芯片之间的电气连接都必须支持四线 I/O 模式，否则 SoC 将无法正常工作。
- 设置*CONFIG_COMPILER_OPTIMIZATION* 为 `Optimize for performance (-O2)`。相较于默认设置，这可能会略微增加二进制文件大小，但几乎必然会提高某些代码的性能。请注意，如果代码包含 C 或 C++ 的未定义行为，提高编译器优化级别可能会暴露出原本未发现的错误。
- 禁用*CONFIG_ESP_SYSTEM_HW_STACK_GUARD* 可能会小幅提高代码性能，尤其是在设备上出现大量中断的情况下。
- 避免使用浮点运算 `float`。尽管 ESP32-P4 具备单精度浮点运算器，但是浮点运算总是慢于整数运算。因此可以考虑使用不同的整数表示方法进行运算，如定点表示法，或者将部分计算用整数运算后再切换为浮点运算。
- 避免使用双精度浮点运算 `double`。ESP32-P4 通过软件模拟进行双精度浮点运算，因此速度非常慢。可以考虑使用基于整数的表示方法或单精度浮点数。

更改 cache 大小 在 ESP32-P4 上，通过下面列出的 Kconfig 选项增加 cache 的大小，“cache 缺失”的频率可能会降低，从而在一定程度上提高整体速度。

- *CONFIG_CACHE_L2_CACHE_SIZE*.

备注： 增加 cache 大小也将导致可用 RAM 的减少。

减少日志开销 尽管标准输出会先存储在缓冲区中，但缓冲区缺少可用空间时，应用程序将数据输出到日志的速度可能会受限。这点在程序启动并输出大量日志时尤为明显，但也可能随时发生。为解决这一问题，可以采取以下几种方法：

- 通过调低应用日志默认等级*CONFIG_LOG_DEFAULT_LEVEL*（引导加载程序日志等级的相应配置为*CONFIG_BOOTLOADER_LOG_LEVEL*）来减少日志输出量。这样做不仅可以减小二进制文件大小，还可以节省一些 CPU 用于格式化字符串的时间。
- 增加*CONFIG_ESP_CONSOLE_UART_BAUDRATE*，可以提高日志输出速度。如果使用内置 USB-CDC 作为串口控制台，那么串口传输速率不会受配置的波特率影响。

不建议的选项 以下选项也可以提高执行速度，但不建议使用，因为它们会降低固件应用程序的可调试性，并可能导致出现更严重的 bug。

- 禁用*CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL*。这也会略微减小固件二进制文件大小。然而，它可能导致出现更严重的 bug，甚至出现安全性 bug。如果为了优化特定函数而必须禁用该选项，可以考虑在该源文件的顶部单独添加 `#define NDEBUG`。

针对性优化 以下更改将提高固件应用程序特定部分的速度：

- 将频繁执行的代码移至 IRAM。应用程序中的所有代码都默认从 flash 中执行。这意味着缓存缺失时，CPU 需要等待从 flash 加载后续指令。如果将函数复制到 IRAM 中，则仅需要在启动时加载一次，然后始终以全速执行。
  IRAM 资源有限，使用更多的 IRAM 可能会减少可用的 DRAM。因此，将代码移动到 IRAM 需要有所取舍。更多信息参见*IRAM（指令 RAM）*。

- 针对不需要放置在 IRAM 中的单个源文件，可以重新启用跳转表优化。这将提高大型 `switch cases` 代码中的热路径性能。关于如何在编译单个源文件时添加 -fjump-tables -ftree-switch-conversion 选项，参见组件编译控制 。

减少启动时间 除了上述提高整体性能的方法外，还可以微调以下选项来专门减少启动时间：

- 最小化*CONFIG_LOG_DEFAULT_LEVEL* 和*CONFIG_BOOTLOADER_LOG_LEVEL* 可以大幅减少启动时间。如要在应用程序启动后获取更多日志，可以设置*CONFIG_LOG_MAXIMUM_LEVEL*，然后调用`esp_log_level_set()` 来恢复更高级别的日志输出。示例 system/startup_time 的主函数展示了如何实现这一点。
- 如果使用 Deep-sleep 模式，启用*CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP* 可以加快从睡眠中唤醒的速度。请注意，启用该选项后在唤醒时将不会执行安全启动验证，需要考量安全风险。
- 设置*CONFIG_BOOTLOADER_SKIP_VALIDATE_ON_POWER_ON* 可以在每次上电复位启动时跳过二进制文件验证，节省的时间取决于二进制文件大小和 flash 设置。请注意，如果 flash 意外损坏，此设置将有一定风险。更多关于使用该选项的解释和建议，参见项目配置 。
- 禁用 RTC 慢速时钟校准可以节省一小部分启动时间。设置*CONFIG_RTC_CLK_CAL_CYCLES* 为 0 可以实现该操作。设置后，以 RTC 慢速时钟为时钟源的固件部分精确度将降低。
- 使用外部内存（启用*CONFIG_SPIRAM*）时，启用外部内存 (*CONFIG_SPIRAM_MEMTEST*) 测试可能会大大增加启动时间（每测试 4 MB 的内存大约增加 1 秒）。禁用内存测试将减少启动时间，但将无法对外部存储器进行测试。
- 使用外部内存（启用*CONFIG_SPIRAM*）时，所有用作堆的内存（包括外部内存）都将被设为默认值，所以启用全面的 poisoning 将增加启动时间（每设置 4 MiB 的内存大约增加 300 毫秒）。

示例项目 system/startup_time 预配了优化启动时间的设置，文件 system/startup_time/sdkconfig.defaults 包含了所有相关设置。可以将这些设置追加到项目中 `sdkconfig` 文件的末尾并合并，但请事先阅读每个设置的相关说明。

任务优先级 ESP-IDF FreeRTOS 是实时操作系统，因此需确保高吞吐量或低延迟的任务获得更高优先级，以便立即运行。调用`xTaskCreate()` 或`xTaskCreatePinnedToCore()` 会设定优先级，并且可以在运行时调用`vTaskPrioritySet()` 进行更改。

此外，还需确保任务适时释放 CPU（通过调用`vTaskDelay()` 或 `sleep()`，或在信号量、队列、任务通知等方面进行阻塞），以避免低优先级任务饥饿并造成系统性问题。任务看门狗定时器 (TWDT) 提供任务饥饿自动检测机制，但请注意，正确的固件操作有时需要长时间运算，因此任务看门狗定时器超时并不总意味着存在问题。在这些情况下，可能需要微调超时时限，甚至禁用任务看门狗定时器。

内置任务优先级 ESP-IDF 启动的系统任务预设了固定优先级。启动时，一些任务会自动启动，而另一些仅在应用程序固件初始化特定功能时启动。为优化性能，请合理设置应用程序任务优先级，以确保它们不会被系统任务阻塞，同时需确保系统任务不会饥饿进而影响其他系统功能。

为此，可能需要分解特定任务。例如，可以在高优先级任务或中断处理程序中执行实时操作，并在较低优先级任务中处理非实时操作。

头文件 components/esp_system/include/esp_task.h 包含了用于设置 ESP-IDF 内置任务系统优先级的宏定义。更多系统任务详情，参见后台任务 。

常见优先级包括：

- 运行主任务 中执行 app_main 函数的主任务优先级最低 (1) 且默认固定在核 0 上执行（可配置 ）。
- 系统任务*ESP 定时器* 用于管理定时器事件并执行回调函数，优先级较高 (22, `ESP_TASK_TIMER_PRIO`) 且固定在核 0 上执行。
- FreeRTOS 初始化调度器时会创建定时器任务，用于处理 FreeRTOS 定时器的回调函数，优先级最低（1，可配置 ）且固定在核 0 上执行。
- 系统任务事件循环库 用于管理默认的系统事件循环并执行回调函数，优先级较高 (20, `ESP_TASK_EVENT_PRIO`) 且固定在核 0 上执行。此配置仅在应用程序调

用*esp_event_loop_create_default()* 时使用。可以调用*esp_event_loop_create()* 添加自定义任务配置。

- ***lwIP*** **TCP/IP 任务优先级较高 (18, ESP_TASK_TCPIP_PRIO) 且并未固定在特定内核上执行（可配置）。**
  - 堆栈事件回调任务 ("BTC") 优先级较高 (19)。
  - 堆栈 BTU 层任务优先级较高 (20)。
  - Host HCI 主任务优先级较高 (22)。

  所有 Bluedroid 任务默认固定在同一个核心上执行，即核 0（可配置）。
- 以太网驱动程序会创建一个 MAC 任务，用于接收以太网帧。如果使用默认配置 ETH_MAC_DEFAULT_CONFIG，则该任务为中高优先级 (15) 且并未固定在特定内核上执行。可以在以太网 MAC 初始化时输入自定义*eth_mac_config_t* 结构体来更改此设置。
- 如果使用*ESP-MQTT* 组件，它会创建优先级默认为 5 的任务（可配置，也可通过*CONFIG_MQTT_USE_CUSTOM_CONFIG* 调整）。该任务未固定在特定内核上执行（可配置）。
- 关于 mDNS 服务的任务优先级，参见 性能优化 。

设定应用程序任务优先级

备注： 对内置 SPI flash 芯片进行写入操作时，任务会完全暂停执行。只有*IRAM 安全中断处理程序* 会继续执行。

提高中断性能 ESP-IDF 支持动态中断分配 和中断抢占。系统中每个中断都有相应优先级，较高优先级的中断将优先执行。

只要其他任务不在临界区内，中断处理程序将优先于所有其他任务执行。因此，尽量减少中断处理程序的执行时间十分重要。

要以最佳性能执行特定中断处理程序，可以考虑：

- 调用*esp_intr_alloc()* 时使用 ESP_INTR_FLAG_LEVEL2 或 ESP_INTR_FLAG_LEVEL3 等标志，可以为更重要的中断设定更高优先级。
- 将中断分配到不运行内置 Wi-Fi/蓝牙任务的 CPU 上执行，即默认情况下，将中断分配到核 1 上执行，参见内置任务优先级 。调用*esp_intr_alloc()* 函数即可将中断分配到函数所在 CPU。
- 如果确定整个中断处理程序可以在 IRAM 中运行（参见*IRAM 安全中断处理程序*），那么在调用*esp_intr_alloc()* 分配中断时，请设置 ESP_INTR_FLAG_IRAM 标志，这样可以防止在应用程序固件写入内置 SPI flash 时临时禁用中断。
- 即使是非 IRAM 安全的中断处理程序，如果需要频繁执行，可以考虑将处理程序的函数移到 IRAM 中，从而尽可能规避执行中断代码时发生 flash 缓存缺失的可能性（参见针对性优化）。如果可以确保只有部分处理程序位于 IRAM 中，则无需添加 ESP_INTR_FLAG_IRAM 标志将程序标记为 IRAM 安全。

提高网络速度

- 关于提高 lwIP TCP/IP 网速，参见性能优化 。
- 示例 ethernet/iperf 中的配置针对以太网 TCP/IP 吞吐量进行了大量优化，但该配置会占用更多 RAM。如需了解详情，请查看 ethernet/iperf/sdkconfig.defaults。请注意，部分选项可能会导致可调试性降低、固件大小增加、内存使用增加或其他功能的性能降低等影响。为了获得最佳结果，请阅读上述链接文档，并据此确定哪些选项最适合当前应用程序。

提高 I/O 性能 使用标准 C 库函数，如 fread 和 fwrite 时，相较于使用平台特定的不带缓冲系统调用，I/O 性能可能更慢，如 read 和 write 。标准 C 库函数是为可移植性而设计的，它们会在执行时会引入一定开销和缓冲延迟，因此并不适用需要较高执行速度的场景。

FAT 文件系统 具体信息和提示如下：

- 读取/写入请求的最大大小等于 FatFS 簇大小（分配单元大小）。

- 使用 `read` 和 `write` 而非 `fread` 和 `fwrite` 可以提高性能。
- 要提高诸如 `fread` 和 `fgets` 等缓冲读取函数的执行速度，可以增加文件缓冲区的大小（Newlib 的默认值为 128 字节），例如 4096、8192 或 16384 字节。为此，可以在特定文件的指针上使用 `setvbuf` 函数进行局部更改，或者修改*CONFIG_FATFS_VFS_FSTAT_BLKSIZE* 实现全局应用。

> **备注：** 增加缓冲区的大小会增加堆内存的使用量。

最小化二进制文件大小

ESP-IDF 构建系统会编译项目和 ESP-IDF 中所有源文件，但只有程序实际引用的函数和变量才会链接到最终的二进制文件中。在某些情况下，需要减小固件二进制文件的总大小，例如，为使固件适配 flash 分区大小。

要减小固件二进制文件总大小，首先要找到导致其大小增加的原因。

测量静态数据大小 为了优化固件二进制文件大小和内存使用，需要测量项目中静态分配的 RAM（`data`，`bss`），代码（`text`）和只读数据（`rodata`）。

使用*idf.py* 的子命令 `size`，`size-components` 和 `size-files` 可以输出项目使用内存概况：

数据大小概况 `idf.py size`

```
$ idf.py size
[...]
Total sizes:
Used stat D/IRAM:   53743 bytes ( 122385 remain, 30.5% used)
      .data size:    6504 bytes
      .bss  size:    1984 bytes
      .text size:   44228 bytes
   .vectors size:    1027 bytes
Used Flash size :  118879 bytes
           .text:   83467 bytes
         .rodata:   35156 bytes
Total image size:  170638 bytes (.bin may be padded larger)
```

该输出结果细分了固件二进制文件中所有静态内存区域的大小：

```
$ idf.py size
[...]
Total sizes:
Used stat D/IRAM:   53743 bytes ( 122385 remain, 30.5% used)
      .data size:    6504 bytes
      .bss  size:    1984 bytes
      .text size:   44228 bytes
   .vectors size:    1027 bytes
Used Flash size :  118879 bytes
           .text:   83467 bytes
         .rodata:   35156 bytes
Total image size:  170638 bytes (.bin may be padded larger)
```

- `Used stat D/IRAM`：编译时使用的 D/IRAM 大小。`remain` 表示在运行时可用作堆内存的 D/IRAM 余量。请注意，由于元数据开销、实现限制和启动时的堆分配，实际的 DRAM 堆会更小。
  - `.data size`：编译时为 `.data`（即所有初始化值为非零值的静态变量）分配的 D/IRAM 大小。`.data` 还在二进制映像中占用空间来存储非零初始化值。
  - `.bss size`：编译时为 `.bss`（即所有初始化值为零的静态变量）分配的 D/IRAM 大小。`.bss` 不会在 flash 中占用额外空间。

  - `.text size`：用于 `.text` 的 D/IRAM 大小（即所有从内部 RAM 执行的代码）。由于代码最初存储在 `.text` 中，在启动时才会复制到 D/IRAM，因此 `.text` 在二进制映像中也会占用空间。
- `Used Flash size`：使用的 flash 总大小（不包括 D/IRAM 的使用量）。
  - `.text`：用于 `.text`（即通过 flash 缓存执行的所有代码，请参阅*IROM*）的 flash 大小。
  - `.rodata`：用于 `.rodata`（即通过 flash 缓存加载的只读数据，参阅*DROM*）的 flash 大小。
- `Total image size` is the estimated total size of the binary file.

组件使用概况 `idf.py size-components` `idf.py size` 的输出结果不够详细，无法找出导致二进制文件过大的主要原因。要进行更详细的分析，请使用 `idf.py size-components`。

```
$ idf.py size-components
[...]
    Total sizes:
 DRAM .data size:   14956 bytes
 DRAM .bss  size:   15808 bytes
Used static DRAM:   30764 bytes ( 149972 available, 17.0% used)
Used static IRAM:   83918 bytes (  47154 available, 64.0% used)
      Flash code:  559943 bytes
    Flash rodata:  176736 bytes
Total image size:~ 835553 bytes (.bin may be padded larger)
Per-archive contributions to ELF file:
            Archive File DRAM .data & .bss & other   IRAM   D/IRAM Flash code &␣
↪rodata   Total
           libnet80211.a       1267   6044        0   5490        0     107445   ␣
↪18484  138730
                liblwip.a         21   3838        0      0        0      97465   ␣
↪16116  117440
             libmbedtls.a         60    524        0      0        0      27655   ␣
↪69907   98146
          libmbedcrypto.a         64     81        0     30        0      76645   ␣
↪11661   88481
                  libpp.a       2427   1292        0  20851        0      37208    ␣
↪4708   66486
                   libc.a          4      0        0      0        0      57056    ␣
↪6455   63515
                 libphy.a       1439    715        0   7798        0      33074     ␣
↪  0   43026
     libwpa_supplicant.a         12    848        0      0        0      35505    ␣
↪1446   37811
            libfreertos.a       3104    740        0  15711        0        367    ␣
↪4228   24150
           libnvs_flash.a          0     24        0      0        0      14347    ␣
↪2924   17295
           libspi_flash.a       1562    294        0   8851        0       1840    ␣
↪1913   14460
          libesp_system.a        245    206        0   3078        0       5990    ␣
↪3817   13336
             libesp-tls.a          0      4        0      0        0       5637    ␣
↪3524    9165
[... removed some lines here ...]
            libesp_rom.a          0      0        0    112        0          0     ␣
↪  0     112
                 libcxx.a          0      0        0      0        0         47     ␣
↪  0      47
                    (exe)          0      0        0      3        0          3     ␣
↪ 12      18
              libesp_pm.a          0      0        0      0        0          8     ␣
↪  0       8
            libesp_eth.a          0      0        0      0        0          0     ␣
↪  0       0
```

(下页继续)

(续上页)

```
                libmesh.a          0       0       0       0       0           0     ␣
↪  0        0
```

idf.py size-components 输出的前几行与 idf.py size 相同，此外还会输出 Per-archive contributions to ELF file 表格，显示每个静态库对最终二进制文件大小的贡献程度。

通常，每个组件都会构建一个静态库归档文件，尽管部分是由特定组件包含的二进制库，例如，esp_wifi 组件包含了 libnet80211.a。此外，这里还列出了一些工具链库，例如 libc.a 和 libgcc.a，用于提供 C/C++ 标准库和工具链内置功能。

对于只有一个 main 组件的简单项目，可在 libmain.a 目录下找到所有项目代码。若项目包含其特有组件（参阅构建系统），则每个组件将单独在一行中显示。

该表格按静态库归档文件对最终二进制文件大小的贡献程度降序排序。

各列含义如下：

- DRAM .data & .bss & other - .data 和 .bss 分别与上方显示的总数相同。两者都是静态变量，且都会减少运行时的可用 RAM，但 .bss 不会增加二进制文件大小。other 列指任何会增加 RAM 大小的自定义数据段，该值通常为 0。
- IRAM - 与上方显示的总数相同，表示链接到从 IRAM 执行的代码，这些代码占用二进制文件空间，并且会减少运行时可用作堆内存的 DRAM 空间。
- Flash code & rodata - 这些值与上方显示总数相同，指通过 flash 缓存访问的 IROM 和 DROM 空间，对二进制文件大小的贡献。

源文件使用概况 idf.py size-files 要了解更多详情，请运行 idf.py size-files，获取每个目标文件对最终二进制文件大小的贡献概况。每个目标文件对应一个单独的源文件。

```
$ idf.py size-files
[...]
Total sizes:
 DRAM .data size:   14956 bytes
 DRAM .bss  size:   15808 bytes
Used static DRAM:   30764 bytes ( 149972 available, 17.0% used)
Used static IRAM:   83918 bytes (  47154 available, 64.0% used)
      Flash code:  559943 bytes
    Flash rodata:  176736 bytes
Total image size:~ 835553 bytes (.bin may be padded larger)
Per-file contributions to ELF file:
             Object File DRAM .data & .bss & other   IRAM   D/IRAM Flash code &␣
↪rodata   Total
     x509_crt_bundle.S.o          0      0       0      0        0          0   ␣
↪64212   64212
                wl_cnx.o          2   3183       0    221        0      13119    ␣
↪3286   19811
           phy_chip_v7.o        721    614       0   1642        0      16820     ␣
↪  0   19797
       ieee80211_ioctl.o        740     96       0    437        0      15325    ␣
↪2627   19225
                    pp.o       1142     45       0   8871        0       5030     ␣
↪537   15625
      ieee80211_output.o          2     20       0   2118        0      11617     ␣
↪914   14671
         ieee80211_sta.o          1     41       0   1498        0      10858    ␣
↪2218   14616
        lib_a-vfprintf.o          0      0       0      0        0      13829     ␣
↪752   14581
       lib_a-svfprintf.o          0      0       0      0        0      13251     ␣
↪752   14003
```

(下页继续)

(续上页)

```
             ssl_tls.c.o          60       0       0       0       0       12769
↪463    13292
             sockets.c.o           0     648       0       0       0       11096
↪1030    12774
                 nd6.c.o           8     932       0       0       0       11515
↪314    12769
       phy_chip_v7_cal.o         477      53       0    3499       0        8561
↪  0    12590
                  pm.o          32     364       0    2673       0        7788
↪782    11639
       ieee80211_scan.o          18     288       0       0       0        8889
↪1921    11116
     lib_a-svfiprintf.o           0       0       0       0       0        9654
↪1206    10860
      lib_a-vfiprintf.o           0       0       0       0       0       10069
↪734    10803
        ieee80211_ht.o           0       4       0    1186       0        8628
↪898    10716
      phy_chip_v7_ana.o         241      48       0    2657       0        7677
↪  0    10623
             bignum.c.o           0       4       0       0       0        9652
↪752    10408
             tcp_in.c.o           0      52       0       0       0        8750
↪1282    10084
                 trc.o         664      88       0    1726       0        6245
↪1108     9831
              tasks.c.o           8     704       0    7594       0           0
↪1475     9781
         ecp_curves.c.o          28       0       0       0       0        7384
↪2325     9737
                 ecp.c.o           0      64       0       0       0        8864
↪286     9214
     ieee80211_hostap.o           1      41       0       0       0        8578
↪585     9205
                 wdev.o         121     125       0    4499       0        3684
↪580     9009
            tcp_out.c.o           0       0       0       0       0        5686
↪2161     7847
                tcp.c.o           2      26       0       0       0        6161
↪1617     7806
      ieee80211_input.o           0       0       0       0       0        6797
↪973     7770
                wpa.c.o           0     656       0       0       0        6828
↪ 55     7539
[... additional lines removed ...]
```

文件总大小概况下方会显示 `Per-file contributions to ELF file` 表格。

该表格的列与上文运行 `idy.py size-components` 显示的列相同，但该表格的粒度更细，展示了每个目标文件对二进制文件大小的贡献。

例如，文件 `x509_crt_bundle.S.o` 对总固件大小贡献了 64,212 字节，全都存储在 flash 中的 `.rodata` 区域。由此可以推知，该应用程序正在使用*ESP x509 证书包* 功能。如果不使用该功能，固件大小至少可以减少 64,212 字节。

某些目标文件从二进制库中链接至此，因此无法找到对应源文件。要确定一个源文件属于哪个组件，通常可以在 ESP-IDF 源代码树中搜索，或者在链接器映射文件 中查找完整路径。

比较两个二进制文件 如果某些改动影响了二进制文件大小，可以使用 ESP-IDF 工具来详细分析文件大小的确切差异。

该操作不是通过运行 `idf.py` 进行的，而是需要直接运行 Python 工具 esp_idf_size 。

执行该操作，首先需要在构建目录中找到链接器映射文件 PROJECTNAME.map。esp_idf_size 工具会基于链接器映射文件的输出结果分析文件大小差异。

要与另一个二进制文件进行比较，还需要保存该文件对应的 .map 文件，该文件位于构建目录中。

例如，要比较两个构建文件，其中一个使用默认的*CONFIG_COMPILER_OPTIMIZATION* Debug (-Og) 配置，而另一个使用 Optimize for size (-Os) 配置：

```
$ python -m esp_idf_size --diff build_Og/https_request.map build_Os/https_request.
↪map
<CURRENT> MAP file: build_Os/https_request.map
<REFERENCE> MAP file: build_Og/https_request.map
Difference is counted as <CURRENT> - <REFERENCE>, i.e. a positive number means␣
↪that <CURRENT> is larger.
Total sizes of <CURRENT>:
↪<REFERENCE>     Difference
 DRAM .data size:   14516 bytes                                                ␣
↪14956           -440
 DRAM .bss  size:   15792 bytes                                                ␣
↪15808            -16
Used static DRAM:   30308 bytes ( 150428 available, 16.8% used)                ␣
↪30764           -456 (   +456 available,      +0 total)
Used static IRAM:   78498 bytes (  52574 available, 59.9% used)                ␣
↪83918          -5420 (  +5420 available,      +0 total)
      Flash code:  509183 bytes                                               ␣
↪559943         -50760
    Flash rodata:  170592 bytes                                               ␣
↪176736          -6144
Total image size:~ 772789 bytes (.bin may be padded larger)                    ␣
↪835553         -62764
```

从 Difference 列可以看出，改变该设置导致整个二进制文件减小了 60 KB 以上，并且可用的 RAM 增加了 5 KB 以上。

还可以使用 diff 模式来输出表格，显示组件级（静态库）的差异：

备注： 运行 esp_idf_size 时可以使用 --format 选项输出 JSON 或 CSV 格式的结果。

```
python -m esp_idf_size --archives --diff build_Og/https_request.map build_Oshttps_
↪request.map
```

同样适用于比较单个源文件级的差异：

```
python -m esp_idf_size --files --diff build_Og/https_request.map build_Oshttps_
↪request.map
```

了解将输出写入文件等其他选项，可以输入 --help 查看完整列表。

链接器失败时显示文件大小 如果被分配的静态内存大小超越上限，链接器会失败并显示错误信息，例如 DRAM segment data does not fit 和 region `iram0_0_seg' overflowed by 44 bytes 等。

在这些情况下，idf.py size 也无法成功执行。然而，通过手动运行 esp_idf_size，可以查看 **部分静态内存使用情况**。内存使用情况将不包含无法链接的变量，因此仍然会显示有部分可用空间。

映射文件参数为构建目录下的 <projectname>.map 文件。

```
python -m esp_idf_size build/project_name.map
```

还可以查看类似于 size-components 或 size-files 输出的内容：

```
python -m esp_idf_size --archives build/project_name.map
python -m esp_idf_size --files build/project_name.map
```

链接器映射文件

备注： 这是一种非常有用的进阶分析方法。可以先跳转到减小总体文件大小，以后再详阅这一部分。

分析工具 `idf.py size` 通过解析 GNU binutils 的“链接器映射文件”来输出结果，该文件囊括了链接器在创建（即链接到）最终固件二进制文件时的所有操作。

链接器映射文件本身是纯文本文件，因此可以进行读取并准确了解链接器的操作，但这些文件非常复杂冗长，通常有 100,000 行甚至更多。

映射文件分为多个部分，每个部分各有标题，包括：

- `Archive member included to satisfy reference by file (symbol)`
  - 该列表显示了链接器链接各个目标文件时所搜寻的特定符号（函数或变量）。
  - 要了解二进制文件包含特定目标文件的原因，可以查看该列表以及文件末尾的 `Cross Reference Table`。

  备注： 请注意，并非每个显示在该列表中的目标文件最后都会出现在最终二进制文件中，有些目标文件可能会列在 `Discarded input sections` 中。

- `Allocating common symbols`
  - 该列表显示了部分全局变量及其大小。常见符号在 ELF 二进制文件中具有特定含义，但 ESP-IDF 并未广泛使用这些符号。
- `Discarded input sections`
  - 在链接器读取目标文件时，会将一些输入段作为文件的一部分读取并准备链接到最终的二进制文件中，但由于没有其他部分引用这些输入段，这些段最终会被丢弃。
  - 对于 ESP-IDF 项目来说，这个列表可能会非常长，因为我们将每个函数和静态变量都编译到一个独立的段中，以最小化最终二进制文件的大小。具体而言，ESP-IDF 将使用编译器选项 `-ffunction-sections -fdata-sections` 和链接器选项 `--gc-sections`。
  - 在这个列表中出现的条目 **不会**对最终的二进制文件大小产生影响。
- `Memory Configuration` 和 `Linker script and memory map`
  - 这两部分相互关联。输出结果的一部分来自由构建系统 提供的链接器命令行和链接脚本，部分链接脚本由 ESP-IDF 项目通过链接器脚本生成机制 功能生成。
  - 在 map 文件的 `Linker script and memory map` 输出中，会显示链接到最终二进制文件中的每个符号（函数或静态变量）及其地址（以 16 位十六进制数字表示）和长度（也以十六进制表示），还有链接的库和目标文件（可以用于确定组件和源文件）。
  - 在所有占用最终 `.bin` 文件的输出段之后，`memory map` 还会显示一些 ELF 文件中用于调试的段，如 `.debug_*` 等。这些段不会对最终的二进制文件大小产生影响，且这些符号的地址数值很小，从 `0x0000000000000000` 开始递增。
- `Cross Reference Table`
  - 该表格显示了引用了各个符号（函数或静态变量）的目标文件。了解二进制文件包含某个特定符号的原因，可参考该表格以确定引用特定符号的目标文件。

  备注： `Cross Reference Table` 不仅包含最终二进制文件中的符号，还包含已丢弃的段内符号。因此，某个符号出现在该表中并不意味着最终二进制文件包含这一符号，需要单独检查。

备注： 链接器映射文件由 GNU binutils 的链接器 `ld` 而非由 ESP-IDF 生成。本快速概览专从 ESP-IDF 构建系统的角度编写而成，建议自行搜索更多关于链接器映射文件格式的信息。

减小总体文件大小 可以通过以下配置选项减小几乎所有 ESP-IDF 项目最终二进制文件的大小：

- 将*CONFIG_COMPILER_OPTIMIZATION* 设置为 `Optimize for size (-Os)`。在某些情况下，相较于默认设置，`Optimize for size (-Os)` 也可以减小二进制文件的大小。请注意，若代码包含 C 或 C++ 的未定义行为，提高编译器优化级别可能会暴露出原本不存在的错误。
- 通过降低应用程序的*CONFIG_LOG_DEFAULT_LEVEL*，可以减少编译时的日志输出。如果改变*CONFIG_LOG_MAXIMUM_LEVEL* 的默认选项，则可以控制二进制文件的大小。减少编译时的日志输出可以减少二进制文件中的字符串数量，并减小调用日志函数的代码大小。
- 将*CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL* 设置为 `Silent`，可以避免为所有可能失败的断言编译专门的断言字符串和源文件名。尽管如此，仍可以通过查看断言失败时的内存地址以在代码中找到失败断言。
- 除*CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL* 外，还可以通过设置*CONFIG_HAL_DEFAULT_ASSERTION_LEVEL* 单独禁用或静默 HAL 组件的断言。请注意，即使将*CONFIG_HAL_DEFAULT_ASSERTION_LEVEL* 设置为 full-assertion 级别，ESP-IDF 在引导加载程序中也会把 HAL 断言级别降为 silent，以减小引导加载程序的大小。
- 设置*CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT* 会移除针对 ESP-IDF 内部错误检查宏的特定错误消息。错误消息移除后，通过阅读日志输出来调试某些错误条件可能变得更加困难。
- 不要启用*CONFIG_COMPILER_CXX_EXCEPTIONS* 或*CONFIG_COMPILER_CXX_RTTI*，也不要将*CONFIG_COMPILER_STACK_CHECK_MODE* 设置为 Overall。这些选项已默认禁用，启用这些选项会大幅增加二进制文件的大小。
- 禁用*CONFIG_ESP_ERR_TO_NAME_LOOKUP* 将会移除查找表，该表用于将错误日志中的错误值转换成用户友好名称（参阅*错误处理*）。这样做可以减小二进制文件的大小，但错误值将只以整数形式输出。
- 将*CONFIG_ESP_SYSTEM_PANIC* 设置为 `Silent reboot` 可以减小一小部分二进制文件的大小，但此操作 **仅**建议在没有任何开发者使用 UART 输出来调试设备时进行。
- 设置*CONFIG_COMPILER_SAVE_RESTORE_LIBCALLS* 以库调用替代内联的入口/出口代码，可以减小二进制文件的大小。
- 如果应用程序的二进制文件只使用 protocomm 组件的某个安全版本，取消对其他版本的支持可以减小部分代码大小。请通过*CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0*、*CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1* 或者*CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERS* 方式，取消对应版本的支持。

备注： 除了上述众多配置项之外，还有一些配置选项在更改为非默认设置时会增加二进制文件的大小，这些选项未在此列出。配置项的帮助文本中通常会阐明显著增加二进制文件大小的设置。

针对性优化 以下二进制文件大小优化适用于特定的组件或函数：

lwIP IPv6

- 将*CONFIG_LWIP_IPV6* 设置为 `false` 可以减小 lwIP TCP/IP 堆栈的大小，但将仅支持 IPv4。

  备注： 如果禁用 IPv6，*ASIO 端口* 等组件将无法使用。

lwIP IPv4

- 如果不需要 IPv4 连接功能，将*CONFIG_LWIP_IPV4* 设置为 `false` 可以减小 lwIP 的大小，使其成为仅支持 IPv6 的 TCP/IP 堆栈。

  备注： 在禁用 IPv4 支持之前，请注意，仅支持 IPv6 的网络环境尚未普及，必须在本地网络中提供支持，例如，由互联网服务供应商提供支持，或使用受限制的本地网络设置。

Newlib Nano 格式化 ESP-IDF 的 I/O 函数（`printf()` 和 `scanf()` 等）默认使用 Newlib 的“完整”格式化功能。

启用配置选项*CONFIG_NEWLIB_NANO_FORMAT* 将使 Newlib 切换到 Nano 格式化模式。这种模式的代码量更小，并且大部分内容被编译到了 ESP32-P4 的 ROM 中，因此不需要将其添加至二进制文件中。

具体的二进制文件大小差异取决于固件使用的功能，但通常为 25 KB 到 50 KB。

启用 Nano 格式化会减少调用 `printf()` 或其他字符串格式化函数的堆栈使用量，参阅栈内存大小优化。

“Nano” 格式化不支持 64 位整数或 C99 格式化功能。请在 Newlib README 文件中搜索 `--enable-newlib-nano-formatted-io` 来获取完整的限制列表。

MbedTLS 功能 在 **Component Config > mbedTLS** 下有多个默认启用的 mbedTLS 功能，如果不需要，可以禁用相应功能以减小代码大小。

这些功能包括：

- *CONFIG_MBEDTLS_HAVE_TIME*
- *CONFIG_MBEDTLS_ECDSA_DETERMINISTIC*
- *CONFIG_MBEDTLS_SHA512_C*
- *CONFIG_MBEDTLS_SHA3_C*
- *CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS*
- *CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS*
- *CONFIG_MBEDTLS_SSL_CONTEXT_SERIALIZATION*
- *CONFIG_MBEDTLS_SSL_ALPN*
- *CONFIG_MBEDTLS_SSL_RENEGOTIATION*
- *CONFIG_MBEDTLS_CCM_C*
- *CONFIG_MBEDTLS_GCM_C*
- *CONFIG_MBEDTLS_ECP_C*（或者：启用此选项，但在子菜单中禁用部分椭圆曲线）
- *CONFIG_MBEDTLS_ECP_NIST_OPTIM*
- *CONFIG_MBEDTLS_ECP_FIXED_POINT_OPTIM*
- 如果不需要 mbedTLS 的服务器和客户端功能，可以修改*CONFIG_MBEDTLS_TLS_MODE*
- 可以考虑禁用在 `TLS Key Exchange Methods` 子菜单中列出的一些密码套件（例如*CONFIG_MBEDTLS_KEY_EXCHANGE_RSA*），以减小代码大小。

每个选项的帮助文本中都有更多信息可供参考。

重要：强烈建议不要禁用所有 mbedTLS 选项。仅在理解功能用途，并确定在应用程序中不需要此功能时，方可禁用相应选项。请特别注意以下两点：

- 确保设备连接的任何 TLS 服务器仍然可用。如果服务器由第三方或云服务控制，建议确保固件至少支持两种 TLS 密码套件，以防未来某次更新禁用了其中一种。
- 确保连接设备的任何 TLS 客户端仍然可以使用支持/推荐的密码套件进行连接。请注意，未来版本的客户端操作系统可能会移除对某些功能的支持，因此建议启用多个支持的密码套件或算法以实现冗余。

如果依赖于第三方客户端或服务器，请密切关注其有关支持的 TLS 功能的公告和变更。否则，当所支持功能变更时，ESP32-P4 设备可能无法访问。

备注： ESP-IDF 并未测试所有 mbedTLS 编译配置组合。如果发现某个组合无法编译或无法按预期执行，请在 GitHub 上报告详细信息。

虚拟文件系统 (VFS) 在 ESP-IDF 中，虚拟文件系统组件 功能允许使用标准的 I/O 函数（如 `open`、`read`、`write` 等）和 C 库函数（如 `fopen`、`fread`、`fwrite` 等）来访问多个文件系统驱动程序和类似文件的外设驱动程序。当应用程序中不需要文件系统或类似文件的外设驱动功能时，可以部分或完全禁用该功能。VFS 组件提供以下配置选项：

- *CONFIG_VFS_SUPPORT_TERMIOS* —如果应用程序不使用 `termios` 函数族，可以禁用此选项。目前，这些函数仅在 UART VFS 驱动程序中实现，大多数应用程序可以禁用此选项。禁用后可以减小约 1.8 KB 代码大小。

- *CONFIG_VFS_SUPPORT_SELECT* —如果应用程序不使用 `select` 函数处理文件描述符，可以禁用此选项。目前，只有 UART 和 eventfd VFS 驱动程序支持 `select` 函数。请注意，当禁用该选项时，仍然可以使用 `select` 处理套接字文件描述符。禁用此选项将减小约 2.7 KB 代码大小。
- *CONFIG_VFS_SUPPORT_DIR* —如果应用程序不使用与目录相关的函数，例如 `readdir`（参阅此选项的描述以获取完整列表），可以禁用此选项。如果应用程序只需打开、读取和写入特定文件，而不需要枚举或创建目录，可以禁用此选项，从而减少超过 0.5 KB 代码大小，具体减小多少取决于使用的文件系统驱动程序。
- *CONFIG_VFS_SUPPORT_IO* —如果应用程序不使用文件系统或类似文件的外设驱动程序，可以禁用此选项，这将禁用所有 VFS 功能，包括上述三个选项。当禁用此选项时，无法使用控制台终端。请注意，当禁用此选项时，应用程序仍然可以使用标准 I/O 函数处理套接字文件描述符。相较于默认配置，禁用此选项可以减小约 9.4 KB 代码大小。

HAL

- 启用*CONFIG_HAL_SYSTIMER_USE_ROM_IMPL* 可以通过链接 ROM 实现的 systimer HAL 驱动程序来减少 IRAM 使用和二进制文件大小。
- 启用*CONFIG_HAL_WDT_USE_ROM_IMPL* 可以通过链接 ROM 实现的看门狗 HAL 驱动程序来减少 IRAM 使用和二进制文件大小。

堆

- 启用*CONFIG_HEAP_TLSF_USE_ROM_IMPL* 可以将整个堆功能放置在 flash 中，从而减少 IRAM 使用和二进制文件大小。

引导加载程序大小 本文档仅涉及 ESP-IDF 应用程序的二进制文件大小，而不涉及 ESP-IDF 二级引导程序。

关于 ESP-IDF 引导加载程序二进制文件大小的讨论，参阅引导加载程序大小。

IRAM 二进制文件大小 如果二进制文件的 IRAM 部分过大，可以通过减少 IRAM 使用来解决这个问题，参阅*IRAM 优化*。

内存优化

固件应用程序的可用 RAM 在某些情况下可能处于低水平，甚至完全耗尽。为此，应调整这些情况下固件应用程序的内存使用情况。

固件应用程序通常需要为内部 RAM 保留备用空间，用于应对非常规情况，或在后续版本的更新中，适应 RAM 使用需求的变化。

背景 在进行 ESP-IDF 的内存优化前，应了解有关 ESP32-P4 内存类型的基础知识、C 语言中静态和动态内存使用的区别、以及 ESP-IDF 中栈和堆的使用方式。以上信息均可参阅堆内存分配。

测量静态内存使用情况 *idf.py* 工具可用于生成应用程序静态内存的使用情况报告，请参阅测量静态数据大小。

测量动态内存使用情况 ESP-IDF 包含一系列堆 API，可以在运行时测量空闲堆内存，请参阅堆内存调试。

备注： 在嵌入式系统中，除 RAM 使用总量外，也应重点关注堆碎片化问题。堆测量 API 提供了一些方法，可以测量最大空闲内存块。通过监测最大空闲内存块和总空闲字节数，可以快速了解是否存在堆碎片化问题。

静态内存优化

- 降低应用程序的静态内存使用，会增加运行时堆的可用 RAM 空间，反之亦然。
- 优化静态内存使用通常需要监测 `.data` 和 `.bss` 的大小，有关工具请参阅测量静态数据大小。
- 在 C 语言中，ESP-IDF 内部函数不会占用大量静态 RAM。在多数情况下（例如 Wi-Fi 库和蓝牙控制器），静态缓冲区仍从堆中分配。然而，这些分配只在功能初始化阶段进行一次，并在功能去初始化时释放，从而在应用程序生命周期中，优化不同阶段的可用内存。

要实现静态内存优化，请执行以下操作：

- 由于常量数据可以存储在 flash 中，不占用 RAM，建议尽量将结构体、缓冲区或其他变量声明为 `const`。为此，可能需要修改固件参数，使其接收 `const *` 参数而非可变指针参数。以上更改还可以减少某些函数的栈内存使用。
- 若使用 OpenThread，请设置*CONFIG_OPENTHREAD_PLATFORM_MSGPOOL_MANAGEMENT* 选项，OpenThread 将从外部 PSRAM 中分配消息池缓冲区，从而减少对内部静态内存的使用。

栈内存大小优化 在 FreeRTOS 操作系统中，任务栈通常从堆中分配。每个任务的栈大小固定，且会作为参数传递给*`xTaskCreate()`*。每个任务可用的栈内存不得超过为其分配的栈内存大小，否则将导致栈内存溢出或堆内存损坏，使原本可用的程序崩溃。

因此，确定每个任务栈内存的最佳大小、最小化每个任务栈内存大小、以及最小化任务栈内存的整体数量，都可以大幅减少 RAM 的使用。

要确定特定任务栈内存的最佳大小，请执行以下操作：

- 程序运行时，如你认为某任务有未使用的栈内存，可通过其任务句柄调用*`uxTaskGetStackHighWaterMark()`*。该函数将以字节为单位，返回任务中生命周期最短的空闲栈内存。
  - 从任务本身内部调用*`uxTaskGetStackHighWaterMark()`* 是调用该函数最容易的方式：在任务达到其栈内存使用峰值后，调用 `uxTaskGetStackHighWaterMark(NULL)` 获取当前任务的高水位标记，换言之，如果有主循环，请多次执行主循环来覆盖各种状态，随后调用*`uxTaskGetStackHighWaterMark()`*。
  - 通常可以用任务的栈内存总大小减去调用*`uxTaskGetStackHighWaterMark()`* 的返回值，计算任务实际使用的栈内存大小，但应留出一定的安全余量，应对运行时栈内存使用量的小幅意外增长。
- 程序运行时，调用*`uxTaskGetSystemState()`* 获取系统中所有任务的摘要，包括各栈内存的高水位标记值。
- 在未使用调试器的监视点时，可以设置*CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK* 选项。启用此选项时，系统会使用一个观察点，监视每个任务栈的最后一个字节。如果有新的数据覆盖了该字节（例如发生栈溢出），将立即触发 panic。相比默认*CONFIG_FREERTOS_CHECK_STACKOVERFLOW* 选项的 `Check using canary bytes`，这种方式更可靠，因其能够立即触发 panic，而不是在下一次 RTOS 上下文切换时触发。然而，两种选项都存在缺点，有时栈指针可能会跳过监视点或 canary 字节，损坏 RAM 的其他区域。

要减少特定任务栈内存大小，请执行以下操作：

- 避免占用过多栈内存的函数。字符串格式化函数（如 `printf()`）会使用大量栈内存，如果任务不调用这类函数，通常可以减小其占用的栈内存。
  - 启用*Newlib Nano 格式化*，可以在任务调用 `printf()` 或其他 C 语言字符串格式化函数时，减少这类任务的栈内存使用量。
- 避免在栈上分配大型变量。在 C 语言声明的默认作用域中，任何分配为自动变量的大型结构体或数组都会占用栈内存。要优化这些变量占用的栈内存大小，可以使用静态分配，或仅在需要时从堆中动态分配。
- 避免调用深度递归函数。尽管调用单个递归函数并不一定会占用大量栈内存，但若每个函数都包含大量基于栈的变量，那么调用这些函数的开销将会很高。

要减少任务的整体数量，请执行以下操作：

- 合并任务。如果从未创建某个特定任务，就不会分配该任务的栈内存，从而极大减少 RAM 使用。如果某些任务可以与另一个任务合并，通常可以将不必要的任务删除。在应用程序中，如果满足以下条件，通常可以合并或删除任务：

  - 任务所执行的内容可以按顺序分解为多个函数调用。
  - 任务所执行的内容可以分解为较小的工作，这些工作可以通过 FreeRTOS 队列或类似机制串行化，并由工作任务执行。

内部任务栈内存大小 为进行系统维护，或操作系统功能，ESP-IDF 分配了许多内部任务，一部分在启动过程中创建，一部分在初始化特定功能时创建。

为了确保支持所有常见的使用模式，这些任务栈内存的默认设置值较大。ESP-IDF 支持配置栈内存大小，因此可以减小任务栈内存，匹配其实际运行时的栈内存使用情况。

重要： 如果内部任务的栈内存设置得过小，可能会导致 ESP-IDF 发生无法预测的崩溃。即使任务栈内存溢出是导致崩溃的根本原因，在调试过程中也很难确定具体原因。因此，建议特别关注任务在负载高时的高水位标记，在必要情况下，谨慎减小内部任务的栈内存大小。如果在减小内部任务堆内存大小后，仍遇到问题，请在报告中提供以下信息，以及正在使用的具体配置。

- 运行主任务 的栈内存大小为*CONFIG_ESP_MAIN_TASK_STACK_SIZE*。
- 系统任务*ESP 定时器* 用于执行回调函数，其栈内存大小为*CONFIG_ESP_TIMER_TASK_STACK_SIZE*。
- 部分 FreeRTOS 定时器任务用于处理 FreeRTOS 定时器回调，其栈内存大小为*CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH*。
- 系统任务事件循环库 用于执行默认系统事件循环回调，其栈内存大小为*CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE*。
- TCP/IP 任务*lwIP* 的栈内存大小为*CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*。
- 以太网驱动程序会创建任务，用于使 MAC 接收以太网帧，在默认配置 `ETH_MAC_DEFAULT_CONFIG` 下，任务栈内存大小为 4 KB。在初始化以太网 MAC 时，传递自定义`eth_mac_config_t` 结构体可以更改此设置。
- FreeRTOS 空闲任务栈内存大小由*CONFIG_FREERTOS_IDLE_TASK_STACKSIZE* 配置。
- 使用*ESP-MQTT* 组件时会创建一个任务，其栈内存大小由*CONFIG_MQTT_TASK_STACK_SIZE* 配置。MQTT 栈内存大小也可以使用`esp_mqtt_client_config_t` 结构体中的 `task_stack` 字段配置。
- 有关使用 `mDNS` 时内存优化的详细信息，请参阅 优化内存使用。

备注： 除 ESP 定时器等内置系统功能外，若固件应用程序没有初始化 ESP-IDF 中特定功能，则不会创建相关任务。此时，相关任务的栈内存使用量为零，而这些功能没有与之关联的任务，因此无需考虑其栈内存大小配置。

堆内存优化 有关分析运行时堆内存使用的函数，请参阅堆内存调试。

通常，堆内存优化包含以下几个方面：分析堆内存使用情况、撤回未使用的 `malloc()` 调用、缩小相应的内存使用大小、或提早释放先前分配的缓冲区。

以下是一些 ESP-IDF 配置选项，有助于在运行时实现堆内存优化：

- lwIP 文档中的有关章节介绍了如何配置最小内存使用。
- 部分 Mbed TLS 配置选项也可用于堆内存优化，详情请参阅减少内存使用 的 Mbed TLS 部分。

备注： 如果将某些配置选项更改为非默认值，也会增加运行时的堆内存使用。这类选项未在上文中列出，但配置选项的帮助文档中给出了相应说明。

IRAM 优化 程序运行时，由于使用了静态 IRAM，用于堆内存使用的 DRAM 会相应减少。反之，可以通过减少 IRAM 使用，增加可用 DRAM。

如果应用程序分配的静态 IRAM 超过可用上限，应用程序将无法构建，并出现链接器错误，如 `section '.iram0.text' will not fit in region 'iram0_0_seg'`、`IRAM0 segment data does not fit` 以及 `region 'iram0_0_seg' overflowed by 84-bytes`。如果发生这种情况，应找到减少静态 IRAM 使用的方法，链接应用程序。

要分析固件应用程序二进制文件中的 IRAM 使用情况，请使用测量静态数据大小。如果固件应用程序链接失败，请参阅链接器失败时显示文件大小 中的步骤，分析失败原因。

要对某些 ESP-IDF 功能进行 IRAM 优化，请使用以下选项：

- 启用*CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH*。只要没有从 ISR 中错误地调用这些函数，就可以在所有配置中安全启用此选项。
- 启用*CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH*。只要没有从 ISR 中错误地调用这些函数，就可以在所有配置中安全启用此选项。
- 启用*CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH*。如果从 IRAM 中的中断上下文中使用 ISR ringbuf 函数，例如启用了*CONFIG_UART_ISR_IN_IRAM*，则无法安全使用此选项。在此情况下，安装 ESP-IDF 相关驱动程序时，将在运行时报错。
- 禁用*CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR* 可以防止从*IRAM 安全中断处理程序* 中发布 `esp_event` 事件，节省 IRAM 空间。
- 禁用*CONFIG_SPI_MASTER_ISR_IN_IRAM* 可以防止在写入 flash 时发生 spi_master 中断，节省 IRAM 空间，但可能影响 spi_master 的性能。
- 禁用*CONFIG_SPI_SLAVE_ISR_IN_IRAM* 可以防止在写入 flash 时发生 spi_slave 中断，节省 IRAM 空间。
- 设置*CONFIG_HAL_DEFAULT_ASSERTION_LEVEL* 为禁用 HAL 组件的断言，可以节省 IRAM 空间，对于经常调用 `HAL_ASSERT` 且位于 IRAM 中的 HAL 代码尤为如此。
- 要禁用不需要的 flash 驱动程序，节省 IRAM 空间，请参阅 sdkconfig 菜单中的 `Auto-detect Flash chips` 选项。
- 启用*CONFIG_HEAP_PLACE_FUNCTION_INTO_FLASH*。只要未启用*CONFIG_SPI_MASTER_ISR_IN_IRAM* 选项，且没有从 ISR 中错误地调用堆函数，就可以在所有配置中安全启用此选项。

备注： 将常用函数从 IRAM 移动到 flash，可能会增加函数的执行时间。

备注： 部分配置选项可以将一些功能移动到 IRAM 中，从而提高性能，但这类选项默认不进行配置，因此未在此列出。了解启用上述选项对 IRAM 大小造成的影响，请参阅配置项的帮助文本。

改变 cache 大小 ESP32-P4 RAM 内存可用大小取决于 cache 的大小。在下面列出的 Kconfig 选项中减少 cache 大小将会增加可用的 RAM。

- *CONFIG_CACHE_L2_CACHE_SIZE*

4.20 Reproducible Builds

4.20.1 Introduction

ESP-IDF build system has support for reproducible builds.

When reproducible builds are enabled, the application built with ESP-IDF does not depend on the build environment. Both the .elf file and .bin files of the application remains exactly the same, even if the following variables change:

- Directory where the project is located
- Directory where ESP-IDF is located (`IDF_PATH`)
- Build time

4.20.2 Reasons for Non-Reproducible Builds

There are several reasons why an application may depend on the build environment, even when the same source code and tools versions are used.

- In C code, `__FILE__` preprocessor macro is expanded to the full path of the source file.
- `__DATE__` and `__TIME__` preprocessor macros are expanded to compilation date and time.
- When the compiler generates object files, it adds sections with debug information. These sections help debuggers, like GDB, to locate the source code which corresponds to a particular location in the machine code. These sections typically contain paths of relevant source files. These paths may be absolute, and will include the path to ESP-IDF or to the project.

There are also other possible reasons, such as unstable order of inputs and non-determinism in the build system.

4.20.3 Enabling Reproducible Builds in ESP-IDF

Reproducible builds can be enabled in ESP-IDF using *CONFIG_APP_REPRODUCIBLE_BUILD* option.

This option is disabled by default. It can be enabled in `menuconfig`.

The option may also be added into `sdkconfig.defaults`. If adding the option into `sdkconfig.defaults`, delete the `sdkconfig` file and run the build again. See *自定义 sdkconfig 的默认值* for more information.

4.20.4 How Reproducible Builds Are Achieved

ESP-IDF achieves reproducible builds using the following measures:

- In ESP-IDF source code, `__DATE__` and `__TIME__` macros are not used when reproducible builds are enabled. Note, if the application source code uses these macros, the build will not be reproducible.
- ESP-IDF build system passes a set of `-fmacro-prefix-map` and `-fdebug-prefix-map` flags to replace base paths with placeholders:
  - Path to ESP-IDF is replaced with `/IDF`
  - Path to the project is replaced with `/IDF_PROJECT`
  - Path to the build directory is replaced with `/IDF_BUILD`
  - Paths to components are replaced with `/COMPONENT_NAME_DIR` (where `NAME` is the name of the component)
- Build date and time are not included into the *application metadata structure* and *bootloader metadata structure* if *CONFIG_APP_REPRODUCIBLE_BUILD* is enabled.
- ESP-IDF build system ensures that source file lists, component lists and other sequences are sorted before passing them to CMake. Various other parts of the build system, such as the linker script generator also perform sorting to ensure that same output is produced regardless of the environment.

4.20.5 Reproducible Builds and Debugging

When reproducible builds are enabled, file names included in debug information sections are altered as shown in the previous section. Due to this fact, the debugger (GDB) is not able to locate the source files for the given code location.

This issue can be solved using GDB `set substitute-path` command. For example, by adding the following command to GDB init script, the altered paths can be reverted to the original ones:

```
set substitute-path /COMPONENT_FREERTOS_DIR /home/user/esp/esp-idf/components/
↪freertos
```

ESP-IDF build system generates a file with the list of such `set substitute-path` commands automatically during the build process. The file is called `prefix_map_gdbinit` and is located in the project `build` directory.

When *idf.py gdb* is used to start debugging, this additional gdbinit file is automatically passed to GDB. When launching GDB manually or from and IDE, please pass this additional gdbinit script to GDB using `-x build/prefix_map_gdbinit` argument.

4.20.6 Factors Which Still Affect Reproducible Builds

Note that the built application still depends on:

- ESP-IDF version
- Versions of the build tools (CMake, Ninja) and the cross-compiler

IDF Docker 镜像 can be used to ensure that these factors do not affect the build.

4.21 线程局部存储

4.21.1 概述

线程局部存储 (TLS) 机制可以分配变量，使每个现有的线程都有相应变量实例。ESP-IDF 提供了以下三种方法，支持使用此类变量：

- *FreeRTOS 原生 API*：ESP-IDF FreeRTOS 原生 API。
- *Pthread API*：ESP-IDF pthread API。
- *C11 标准*：C11 标准引入了特殊关键字，将变量声明为线程局部变量。

4.21.2 FreeRTOS 原生 API

ESP-IDF FreeRTOS 提供以下 API，用于管理线程局部变量：

- *vTaskSetThreadLocalStoragePointer()*
- *pvTaskGetThreadLocalStoragePointer()*
- *vTaskSetThreadLocalStoragePointerAndDelCallback()*

此时，可以分配的最大变量数量受*CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS* 限制。变量保存在任务控制块 (TCB) 中，并由其索引访问。注意，索引 0 保留，供 ESP-IDF 内部使用。

通过使用上述 API，你可以分配任意大小的线程局部变量，并将其分配给任意数量的任务。不同任务可以拥有不同的 TLS 变量集。

如果变量大小超过 4 个字节，你需要负责为其分配/释放内存。在任务删除时，FreeRTOS 会释放变量，但必须提供回调函数来适当清理。

4.21.3 Pthread API

ESP-IDF 提供以下*POSIX Thread*，用于管理线程局部变量：

- `pthread_key_create()`
- `pthread_key_delete()`
- `pthread_getspecific()`
- `pthread_setspecific()`

Pthread API 具备 FreeRTOS 原生 API 的所有优点，并突破了 FreeRTOS 原生 API 的部分限制，如变量数量仅受堆上可用内存大小的限制。然而由于 Pthread API 具备动态性质，与原生 API 相比，这个 API 引入了额外的性能开销。

4.21.4 C11 标准

ESP-IDF FreeRTOS 支持基于 C11 标准的线程局部变量，使用 `__thread` 关键字指定。要了解 GCC 功能详情，请参阅 GCC 文档。

这类变量的存储空间分配在任务栈上。请注意，即使任务根本不使用这些变量，程序中所有这类变量的存储区域都会在系统中每个任务的栈上分配。例如，ESP-IDF 系统任务（如 ipc、timer 任务等）也有额外的栈空间分配。因此，使用这个特性时，需要谨慎考虑。

在使用 C11 线程局部变量时，可以针对以下特点进行权衡：在编程中使用 C11 线程局部变量非常方便，使用最少的 CPU 指令即可访问，但这一优势的代价是系统中所有任务都会额外使用栈。由于变量分配的静态性质，系统中的所有任务具有一组相同的 C11 线程局部变量。

4.22 工具

4.22.1 IDF 前端工具 - idf.py

idf.py 命令行工具提供了一个前端界面，管理工程构建、工程部署及工程调试等操作。该前端界面使用多项工具，如：

- CMake 用于配置要构建的工程。
- Ninja 用于构建工程。
- esptool.py 用于烧录目标芯片。

第五步：开始使用 ESP-IDF 吧 简要介绍了设置 idf.py 以配置、构建及烧录工程的操作流程。

重要： idf.py 应在 ESP-IDF 工程目录下运行，即包含 CMakeLists.txt 文件的目录。旧版本工程，即包含 Makefile 的目录，与 idf.py 不兼容。

常用命令

创建新工程：create-project

```
idf.py create-project <project name>
```

此命令将创建一个新的 ESP-IDF 工程。此外，使用 --path 选项可指定工程创建路径。

创建新组件：create-component

```
idf.py create-component <component name>
```

此命令将创建一个新的组件，包含构建所需的最基本文件集。使用 -C 选项可指定组件创建目录。有关组件的更多信息，请参阅*组件 CMakeLists 文件*。

选择目标芯片：set-target ESP-IDF 支持多个目标芯片，运行 idf.py --list-targets 查看当前 ESP-IDF 版本支持的所有目标芯片。

```
idf.py set-target <target>
```

此命令将设置当前工程的目标芯片。

重要： idf.py set-target 将清除构建目录，并重新生成 sdkconfig 文件，原来的 sdkconfig 文件保存为 sdkconfig.old。

备注： idf.py set-target 命令与以下操作效果相同：

1. 清除构建目录 (idf.py fullclean)
2. 删除 sdkconfig 文件 (mv sdkconfig sdkconfig.old)

3. 使用新的目标芯片重新配置工程 (`idf.py -DIDF_TARGET=esp32 reconfigure`)

所需的 `IDF_TARGET` 还可以作为环境变量（如 `export IDF_TARGET=esp32s2`）或 CMake 变量（如将 `-DIDF_TARGET=esp32s2` 作为 CMake 或 `idf.py` 的参数）传递。在经常使用同类芯片的情况下，设置环境变量将使操作更加便利。

要给指定工程设定 `IDF_TARGET` 的默认值，请将 `CONFIG_IDF_TARGET` 选项添加到该工程的 `sdkconfig.defaults` 文件(如 `CONFIG_IDF_TARGET="esp32s2"`)。若未通过使用环境变量、CMake 变量或 `idf.py set-target` 命令等方法指定 `IDF_TARGET`，则默认使用该选项的值。

若未通过以上任一方法设置目标芯片，构建系统将默认使用 `esp32`。

启动图形配置工具：`menuconfig`

```
idf.py menuconfig
```

构建工程：`build`

```
idf.py build
```

此命令将构建当前目录下的工程，具体步骤如下：

- 若有需要，创建构建子目录 `build` 保存构建输出文件，使用 `-B` 选项可改变子目录的路径。
- 必要时运行 CMake 配置工程，并为主要构建工具生成构建文件。
- 运行主要构建工具（Ninja 或 `GNU Make`）。默认情况下，构建工具会完成自动检测，也可通过将 `-G` 选项传递给 `idf.py` 来显式设置构建工具。

构建是增量行为，因此若上次构建结束后，源文件或配置并未发生更改，则不会执行任何操作。

此外，使用 `app`、`bootloader` 或 `partition-table` 参数运行此命令，可选择仅构建应用程序、引导加载程序或分区表。

清除构建输出：`clean`

```
idf.py clean
```

此命令可清除构建目录中的构建输出文件，下次构建时，工程将完全重新构建。注意，使用此选项不会删除构建文件夹内的 CMake 配置输出。

删除所有构建内容：`fullclean`

```
idf.py fullclean
```

此命令将删除所有 `build` 子目录内容，包括 CMake 配置输出。下次构建时，CMake 将重新配置其输出。注意，此命令将递归删除构建目录下的 *所有文件*（工程配置将保留），请谨慎使用。

烧录工程：`flash`

```
idf.py flash
```

此命令将在需要时自动构建工程，随后将其烧录到目标芯片。使用 `-p` 和 `-b` 选项可分别设置串口名称和烧录程序的波特率。

备注： 环境变量 `ESPPORT` 和 `ESPBAUD` 可分别设置 `-p` 和 `-b` 选项的默认值，在命令行上设置这些选项的参数可覆盖默认值。

与 `build` 命令类似，使用 `app`、`bootloader` 或 `partition-table` 参数运行此命令，可选择仅烧录应用程序、引导加载程序或分区表。

错误处理提示

idf.py 使用存储在 tools/idf_py_actions/hints.yml 中的提示数据库，当找到与给定错误相匹配的提示时，idf.py 会打印该提示以尝试提供解决方案。目前，错误处理提示不支持 menuconfig 对象。

若无需该功能，可以通过 idf.py 的 --no-hints 参数关闭提示。

重要提示

多个 idf.py 命令可以在同一行命令中组合使用。例如，idf.py -p COM4 clean flash monitor 可以清除源代码树、编译工程、并将其烧录到目标芯片，随后运行串行监视器。

在同一调用中，多个 idf.py 命令的顺序并不重要，它们将自动以正确的程序执行，以使全部操作生效（例如先构建后烧录、先擦除后烧录）。

idf.py 会尝试将未知命令作为构建系统目标执行。

命令 idf.py 支持 bash、zsh 和 fish shell 的 shell 自动补全。

调用命令 export 为 idf.py 启用自动补全（第四步：设置环境变量），按 TAB 键启动自动补全。输入 idf.py - 并按 TAB 键以自动补全选项。

预计未来版本将支持 PowerShell 自动补全。

高级命令

打开文档：docs

```
idf.py docs
```

此命令将在浏览器中打开工程目标芯片和 ESP-IDF 版本对应的文档。

显示大小：Size

```
idf.py size
```

此命令将显示应用程序大小，包括占用的 RAM 和 flash 及各部分（如.bss）的大小。

```
idf.py size-components
```

此命令将显示工程中各个组件的应用程序大小。

```
idf.py size-files
```

该命令将显示工程中每个源文件的大小。

选项

- --format 指定输出格式，可输出 text、csv、json 格式，默认格式为 text。
- --output-file 可选参数，可以指定命令输出文件的文件名，而非标准输出。

重新配置工程：reconfigure

```
idf.py reconfigure
```

此命令将重新运行 CMake。正常情况下并不会用到该命令，因为一般无需重新运行 CMake，但如果从源代码树中添加或删除了文件，或需要修改 CMake 缓存变量时，将有必要使用该命令。例如，idf.py -DNAME='VALUE' reconfigure 可将变量 NAME 在 CMake 缓存中设置为值 VALUE。

清除 Python 字节码：python-clean

```
idf.py python-clean
```

此命令将从 ESP-IDF 目录中删除生成的 Python 字节码。字节码在切换 ESP-IDF 和 Python 版本时可能会引起问题，建议在切换 Python 版本后运行此命令。

生成 UF2 二进制文件：uf2

```
idf.py uf2
```

此命令将在构建目录中生成一个 UF2（USB 烧录格式）二进制文件 `uf2.bin`，该文件包含所有烧录目标芯片所必需的二进制文件，即引导加载程序、应用程序和分区表。

在 ESP 芯片上运行 ESP USB Bridge 项目将创建一个 USB 大容量存储设备，用户可以将生成的 UF2 文件复制到该 USB 设备中，桥接 MCU 将使用该文件来烧录目标 MCU。这一操作十分简单，只需将文件复制（或“拖放”）到文件资源管理器访问的公开磁盘中即可。

如需仅为应用程序生成 UF2 二进制文件，即不包含加载引导程序和分区表，请使用 `uf2-app` 命令。

```
idf.py uf2-app
```

读取 Otadata 分区：read-otadata

```
idf.py read-otadata
```

此命令将打印 `otadata` 分区的内容，该分区存储当前所选 OTA 应用程序分区的信息。有关 `otadata` 分区的更多信息，请参阅*空中升级 (OTA)*。

全局选项

运行 `idf.py --help` 列出所有可用的根级别选项。要列出特定子命令的选项，请运行 `idf.py <command> --help`，如 `idf.py monitor --help`。部分常用选项如下：

- `-C <dir>` 支持从默认的当前工作目录覆盖工程目录。
- `-B <dir>` 支持从工程目录的默认 `build` 子目录覆盖构建目录。
- `--ccache` 可以在安装了 CCache 工具的前提下，在构建源文件时启用 CCache，减少部分构建耗时。

重要： 注意，某些旧版本 CCache 在某些平台上存在 bug，因此如果文件没有按预期重新构建，可禁用 CCache 并重新构建。可以通过将环境变量 `IDF_CCACHE_ENABLE` 设置为非零值来默认启用 CCache。

- `-v` 会使 `idf.py` 和构建系统生成详细的构建输出，有助于调试构建错误。
- `--cmake-warn-uninitialized`（或 `-w`）将使 CMake 只显示在工程目录中发现的变量未初始化的警告，该选项仅控制 CMake 内部的 CMake 变量警告，不控制其他类型的构建警告。将环境变量 `IDF_CMAKE_WARN_UNINITIALIZED` 设置为非零值，可永久启用该选项。
- `--no-hints` 用于禁用有关错误处理的提示并禁用捕获输出。

通过 `@file` 传递参数 可以通过文件向 `idf.py` 传递多个参数。该文件或文件路径须在开头使用 `@` 进行标注。文件中的参数支持通过换行或空格分隔，并按其在 idf.py 命令行中的顺序扩展。

例如，当前有文件 *custom_flash.txt*：

```
flash --baud 115200
```

运行命令：`idf.py @custom_flash.txt monitor`

文件中的参数可以与额外的命令行参数结合使用，也支持同时使用带有 @ 标注的多个文件。例如，另有一个文件 another_config.txt，此时，可以通过指定 idf.py @custom_flash.txt @another_config.txt monitor 同时使用两个文件。

关于参数文件的更多示例，如通过 @filename 创建配置文件概要，请参阅 多个构建配置示例 。

4.22.2 IDF 监视器

IDF 监视器是一个串行终端程序，使用了 esp-idf-monitor 包，用于收发目标设备串口的串行数据，IDF 监视器同时还兼具 ESP-IDF 的其他特性。

在 ESP-IDF 中调用 idf.py monitor 可以启用此监视器。

操作快捷键

为了方便与 IDF 监视器进行交互，请使用表中给出的快捷键。这些快捷键可以自定义，请查看配置文件章节了解详情。

快捷键	操作	描述
Ctrl +]	退出监视器程序	
Ctrl + T	菜单退出键	按下如下给出的任意键之一，并按指示操作。
• Ctrl + T	将菜单字符发送至远程	
• Ctrl +]	将 exit 字符发送至远程	
• Ctrl + P	重置目标设备，进入引导加载程序，通过 RTS 和 DTR 线暂停应用程序	重置目标设备，通过 RTS 和 DTR 线（如已连接）进入引导加载程序。这会阻止开发板运行任何程序，在等待其他设备启动时可以使用此操作。更多详细信息，请参考复位目标到引导加载程序。
• Ctrl + R	通过 RTS 线重置目标设备	重置设备，并通过 RTS 线（如已连接）重新启动应用程序。
• Ctrl + F	编译并烧录此项目	暂停 idf_monitor，运行 `flash` 目标，然后恢复 idf_monitor。任何改动的源文件都会被重新编译，然后重新烧录。如果 idf_monitor 是以参数 `-E` 启动的，则会运行目标 `encrypted-flash`。
• Ctrl + A (或者 A)	仅编译及烧录应用程序	暂停 idf_monitor，运行 `app-flash` 目标，然后恢复 idf_monitor。这与 `flash` 类似，但只有主应用程序被编译并被重新烧录。如果 idf_monitor 是以参数 `-E` 启动的，则会运行目标 `encrypted-flash`。
• Ctrl + Y	停止/恢复在屏幕上打印日志输出	激活时，会丢弃所有传入的串行数据。允许在不退出监视器的情况下快速暂停和检查日志输出。
• Ctrl + L	停止/恢复向文件写入日志输出	在工程目录下创建一个文件，用于写入日志输出。可使用快捷键停止/恢复该功能（退出 IDF 监视器也会终止该功能）。
• Ctrl + I (或者 I)	停止/恢复打印时间标记	IDF 监视器可以在每一行的开头打印一个时间标记。时间标记的格式可以通过 `--timestamp-format` 命令行参数来改变。
• Ctrl + H (或者 H)	显示所有快捷键	
• Ctrl + X (或者 X)	退出监视器程序	
Ctrl + C	中断正在运行的应用程序	暂停 IDF 监视器并运行 GDB 项目调试器，从而在运行时调试应用程序。这需要启用:ref: *CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME* 选项。

除了 `Ctrl-]` 和 `Ctrl-T`，其他快捷键信号会通过串口发送到目标设备。

兼具 ESP-IDF 特性

自动解码地址 每当芯片输出指向可执行代码的十六进制地址时，IDF 监视器将查找该地址在源代码中的位置（文件名和行号），并在下一行用黄色打印出该位置。

ESP-IDF 应用程序发生 crash 和 panic 事件时，将产生如下的寄存器转储和回溯:

```
abort() was called at PC 0x42067cd5 on core 0

Stack dump detected
Core  0 register dump:
MEPC    : 0x40386488  RA      : 0x40386b02  SP      : 0x3fc9a350  GP      :␣
↪0x3fc923c0
TP      : 0xa5a5a5a5  T0      : 0x37363534  T1      : 0x7271706f  T2      :␣
↪0x33323130
S0/FP   : 0x00000004  S1      : 0x3fc9a3b4  A0      : 0x3fc9a37c  A1      :␣
↪0x3fc9a3b2
A2      : 0x00000000  A3      : 0x3fc9a3a9  A4      : 0x00000001  A5      :␣
↪0x3fc99000
A6      : 0x7a797877  A7      : 0x76757473  S2      : 0xa5a5a5a5  S3      :␣
↪0xa5a5a5a5
S4      : 0xa5a5a5a5  S5      : 0xa5a5a5a5  S6      : 0xa5a5a5a5  S7      :␣
↪0xa5a5a5a5
S8      : 0xa5a5a5a5  S9      : 0xa5a5a5a5  S10     : 0xa5a5a5a5  S11     :␣
↪0xa5a5a5a5
T3      : 0x6e6d6c6b  T4      : 0x6a696867  T5      : 0x66656463  T6      :␣
↪0x62613938
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000007  MTVAL   :␣
↪0x00000000

MHARTID : 0x00000000

Stack memory:
3fc9a350: 0xa5a5a5a5 0xa5a5a5a5 0x3fc9a3b0 0x403906cc 0xa5a5a5a5 0xa5a5a5a5␣
↪0xa5a5a5a50
3fc9a370: 0x3fc9a3b4 0x3fc9423c 0x3fc9a3b0 0x726f6261 0x20292874 0x20736177␣
↪0x6c6c61635
3fc9a390: 0x43502074 0x34783020 0x37363032 0x20356463 0x63206e6f 0x2065726f␣
↪0x000000300
3fc9a3b0: 0x00000030 0x36303234 0x35646337 0x3c093700 0x0000002a 0xa5a5a5a5␣
↪0x3c0937f48
3fc9a3d0: 0x00000001 0x3c0917f8 0x3c0937d4 0x0000002a 0xa5a5a5a5 0xa5a5a5a5␣
↪0xa5a5a5a5e
3fc9a3f0: 0x0001f24c 0x000006c8 0x00000000 0x0001c200 0xffffffff 0xffffffff␣
↪0x000000200
3fc9a410: 0x00001000 0x00000002 0x3c093818 0x3fccb470 0xa5a5a5a5 0xa5a5a5a5␣
↪0xa5a5a5a56
.....
```

通过分析堆栈转储 IDF 监视器为寄存器转储补充如下信息:

```
abort() was called at PC 0x42067cd5 on core 0
0x42067cd5: __assert_func at /builds/idf/crosstool-NG/.build/riscv32-esp-elf/src/
↪newlib/newlib/libc/stdlib/assert.c:62 (discriminator 8)

Stack dump detected
Core  0 register dump:
MEPC    : 0x40386488  RA      : 0x40386b02  SP      : 0x3fc9a350  GP      :␣
↪0x3fc923c0
0x40386488: panic_abort at /home/marius/esp-idf_2/components/esp_system/panic.c:367

0x40386b02: rtos_int_enter at /home/marius/esp-idf_2/components/freertos/port/
↪riscv/portasm.S:35

TP      : 0xa5a5a5a5  T0      : 0x37363534  T1      : 0x7271706f  T2      :␣
↪0x33323130
S0/FP   : 0x00000004  S1      : 0x3fc9a3b4  A0      : 0x3fc9a37c  A1      :␣
↪0x3fc9a3b2
A2      : 0x00000000  A3      : 0x3fc9a3a9  A4      : 0x00000001  A5      :␣
↪0x3fc99000
```

(下页继续)

(续上页)

```
A6      : 0x7a797877  A7      : 0x76757473  S2      : 0xa5a5a5a5  S3      :␣
↪0xa5a5a5a5
S4      : 0xa5a5a5a5  S5      : 0xa5a5a5a5  S6      : 0xa5a5a5a5  S7      :␣
↪0xa5a5a5a5
S8      : 0xa5a5a5a5  S9      : 0xa5a5a5a5  S10     : 0xa5a5a5a5  S11     :␣
↪0xa5a5a5a5
T3      : 0x6e6d6c6b  T4      : 0x6a696867  T5      : 0x66656463  T6      :␣
↪0x62613938
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000007  MTVAL   :␣
↪0x00000000

MHARTID : 0x00000000

Backtrace:
panic_abort (details=details@entry=0x3fc9a37c "abort() was called at PC 0x42067cd5␣
↪on core 0") at /home/marius/esp-idf_2/components/esp_system/panic.c:367
367     *((int *) 0) = 0; // NOLINT(clang-analyzer-core.NullDereference) should be␣
↪an invalid operation on targets
#0  panic_abort (details=details@entry=0x3fc9a37c "abort() was called at PC␣
↪0x42067cd5 on core 0") at /home/marius/esp-idf_2/components/esp_system/panic.
↪c:367
#1  0x40386b02 in esp_system_abort (details=details@entry=0x3fc9a37c "abort() was␣
↪called at PC 0x42067cd5 on core 0") at /home/marius/esp-idf_2/components/esp_
↪system/system_api.c:108
#2  0x403906cc in abort () at /home/marius/esp-idf_2/components/newlib/abort.c:46
#3  0x42067cd8 in __assert_func (file=file@entry=0x3c0937f4 "", line=line@entry=42,
↪ func=func@entry=0x3c0937d4 <__func__.8540> "",␣
↪failedexpr=failedexpr@entry=0x3c0917f8 "") at /builds/idf/crosstool-NG/.build/
↪riscv32-esp-elf/src/newlib/newlib/libc/stdlib/assert.c:62
#4  0x4200729e in app_main () at ../main/iperf_example_main.c:42
#5  0x42086cd6 in main_task (args=<optimized out>) at /home/marius/esp-idf_2/
↪components/freertos/port/port_common.c:133
#6  0x40389f3a in vPortEnterCritical () at /home/marius/esp-idf_2/components/
↪freertos/port/riscv/port.c:129
```

IDF 监视器在后台运行以下命令，解码各地址:

```
riscv32-esp-elf-addr2line -pfiaC -e build/PROJECT.elf ADDRESS
```

如果在应用程序源代码中找不到匹配的地址，IDF 监视器还会检查 ROM 代码。此时不会打印源文件名和行号，只显示 函数名 in ROM:

```
abort() was called at PC 0x400481c1 on core 0
0x400481c1: ets_rsa_pss_verify in ROM

Stack dump detected
Core  0 register dump:
MEPC    : 0x4038051c  RA      : 0x40383840  SP      : 0x3fc8f6b0  GP      :␣
↪0x3fc8b000
0x4038051c: panic_abort at /Users/espressif/esp-idf/components/esp_system/panic.
↪c:452
0x40383840: __ubsan_include at /Users/espressif/esp-idf/components/esp_system/
↪ubsan.c:313

TP      : 0x3fc8721c  T0      : 0x37363534  T1      : 0x7271706f  T2      :␣
↪0x33323130
S0/FP   : 0x00000004  S1      : 0x3fc8f714  A0      : 0x3fc8f6dc  A1      :␣
↪0x3fc8f712
A2      : 0x00000000  A3      : 0x3fc8f709  A4      : 0x00000001  A5      :␣
↪0x3fc8c000
A6      : 0x7a797877  A7      : 0x76757473  S2      : 0x00000000  S3      :␣
↪0x3fc8f750
```

(下页继续)

(续上页)

```
S4      : 0x3fc8f7e4  S5      : 0x00000000  S6      : 0x400481b0  S7      :␣
↪0x3c025841
0x400481b0: ets_rsa_pss_verify in ROM
.....
```

ROM ELF 文件会根据 IDF_PATH 和 ESP_ROM_ELF_DIR 环境变量的路径自动加载。如需覆盖此行为，可以通过调用 esp_idf_monitor 并指定特定的 ROM ELF 文件路径：python -m esp_idf_monitor --rom-elf-file [ROM ELF 文件的路径]。

备注： 将环境变量 ESP_MONITOR_DECODE 设置为 0 或者调用 esp_idf_monitor 的特定命令行选项 python -m esp_idf_monitor --disable-address-decoding 来禁止地址解码。

连接时复位目标芯片 默认情况下，IDF 监视器会在目标芯片连接时通过 DTR 和 RTS 串行线自动复位芯片。要防止 IDF 监视器在连接时自动复位，请在调用 IDF 监视器时加上选项 --no-reset，如 idf.py monitor --no-reset，或者将环境变量 ESP_IDF_MONITOR_NO_RESET 设置成 1。

备注： --no-reset 选项在 IDF 监视器连接到特定端口时可以实现同样的效果，如 idf.py monitor --no-reset -p [PORT]。

复位目标到引导加载程序 IDF 监视器可以通过预定义的复位序列将芯片复位到引导加载程序，该序列已经经过调整，可以在大多数环境中正常工作。此外，用户可以设置自定义复位序列。通过对复位序列进行微调，使其适应各种情况。

使用预定义的复位序列 IDF 监视器的默认复位序列可在大多数环境中使用。使用默认序列复位芯片到引导加载程序中，无需进行额外配置。

自定义复位序列 对于高级用户或特定用例，IDF 监视器支持使用配置文件 配置自定义复位序列。这在默认序列可能不足的极端情况下特别有用。

复位序列可通过以下格式的字符串定义：

- 各个命令由 | 分隔（例如 R0|D1|W0.5）。
- 命令（例如 R0）由代码（R）和参数（0）定义。

代码	操作	参数
D	设置 DTR 控制线	1/0
R	设置 RTS 控制线	1/0
U	同时设置 DTR 和 RTS 控制线（仅适用于类 Unix 系统）	0,0/0,1/1,0/1,1
W	等待 N 秒（其中 N 为浮点数）	N

示例：

```
[esp-idf-monitor]
custom_reset_sequence = U0,1|W0.1|D1|R0|W0.5|D0
```

有关更多详细信息，请参阅 Esptool 文档中 custom reset sequence 章节。请注意，IDF 监视器只使用了 Esptool 配置中的 custom_reset_sequence 值，其他值会被 IDF 监视器忽略。

IDF 监视器和 Esptool 之间共享配置 自定义复位序列的配置可以在 IDF 监视器和 Esptool 之间的共享配置文件中指定。在这种情况下，为了使两个工具都能识别配置文件，其名称应为 `setup.cfg` 或 `tox.ini`。

共享配置文件的示例：

```
[esp-idf-monitor]
menu_key = T
skip_menu_key = True

[esptool]
custom_reset_sequence = U0,1|W0.1|D1|R0|W0.5|D0
```

备注： 当在 `[esp-idf-monitor]` 部分和 `[esptool]` 部分都使用 `custom_reset_sequence` 参数时，IDF 监视器会优先使用 `[esp-idf-monitor]` 部分的配置。`[esptool]` 部分中任何与之冲突的配置都将被忽略。

当配置分散在多个文件中时，此优先规则也适用。全局 esp-idf-monitor 配置将优先于本地 esptool 配置。

配置 GDBStub 以启用 GDB GDBStub 支持在运行时进行调试。GDBStub 在目标上运行，并通过串口连接到主机从而接收调试命令。GDBStub 支持读取内存和变量、检查调用堆栈帧等命令。虽然没有 JTAG 调试通用，但由于 GDBStub 完全通过串行端口完成通信，故不需要使用特殊硬件（如 JTAG/USB 桥接器）。

通过设置*CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME*，可以将目标配置为在后台运行 GDBStub。GDBStub 将保持在后台运行，直到通过串行端口发送 `Ctrl+C` 导致应用程序中断（即停止程序执行），从而让 GDBStub 处理调试命令。

此外，还可以通过设置*CONFIG_ESP_SYSTEM_PANIC* 为 `GDBStub on panic` 来配置 panic 处理程序，使其在发生 crash 事件时运行 GDBStub。当 crash 发生时，GDBStub 将通过串口输出特殊的字符串模式，表示 GDBStub 正在运行。

无论是通过发送 `Ctrl+C` 还是收到特殊字符串模式，IDF 监视器都会自动启动 GDB，从而让用户发送调试命令。GDB 退出后，通过 RTS 串口线复位目标。如果未连接 RTS 串口线，请按复位键，手动复位开发板。

备注： IDF 监视器在后台运行如下命令启用 GDB:

```
riscv32-esp-elf-gdb -ex "set serial baud BAUD" -ex "target remote PORT" -ex␣
↪interrupt build/PROJECT.elf :idf_target:`Hello NAME chip`
```

输出筛选 可以调用 `idf.py monitor --print-filter="xyz"` 启动 IDF 监视器，其中，`--print-filter` 是输出筛选的参数。参数默认值为空字符串，即打印所有内容。支持使用环境变量 `ESP_IDF_MONITOR_PRINT_FILTER` 调整筛选设置。

备注： 同时使用环境变量 `ESP_IDF_MONITOR_PRINT_FILTER` 和参数 `--print-filter` 时，通过命令行输入的 CLI 参数 `--print-filter` 优先级更高。

若需对打印内容设置限制，可指定 `<tag>:<log_level>` 等选项，其中 `<tag>` 是标签字符串，`<log_level>` 是 `{N, E, W, I, D, V, *}` 集合中的一个字母，指的是日志级别。

例如，`--print_filter="tag1:W"` 只匹配并打印 `ESP_LOGW("tag1", ...)` 所写的输出，或者写在较低日志详细度级别的输出，即 `ESP_LOGE("tag1", ...)`。请勿指定 `<log_level>` 或使用详细级别默认值 `*`。

备注： 编译时，可以使用主日志在日志库 中禁用不需要的输出。也可以使用 IDF 监视器筛选输出来调整筛选设置，且无需重新编译应用程序。

应用程序标签不能包含空格、星号 *、冒号 :，以便兼容输出筛选功能。

如果应用程序输出的最后一行后面没有回车，可能会影响输出筛选功能，即，监视器开始打印该行，但后来发现该行不应该被写入。这是一个已知问题，可以通过添加回车来避免此问题（特别是在没有输出紧跟其后的情况下）。

筛选规则示例

- `*` 可用于匹配任何类型标签。但 `--print_filter="*:I tag1:E"` 打印关于 `tag1` 的输出时会报错，这是因为 `tag1` 规则比 `*` 规则的优先级高。
- 默认规则（空）等价于 `*:V`，因为在详细级别或更低级别匹配任意标签即意味匹配所有内容。
- `"*:N"` 不仅抑制了日志功能的输出，也抑制了 `printf` 的打印输出。为了避免这一问题，请使用 `*:E` 或更高的冗余级别。
- 规则 `"tag1:V"`、`"tag1:v"`、`"tag1:"`、`"tag1:*"` 和 `"tag1"` 等同。
- 规则 `"tag1:W tag1:E"` 等同于 `"tag1:E"`，这是因为后续出现的具有相同名称的标签会覆盖掉前一个标签。
- 规则 `"tag1:I tag2:W"` 仅在 Info 详细度级别或更低级别打印 `tag1`，在 Warning 详细度级别或更低级别打印 `tag2`。
- 规则 `"tag1:I tag2:W tag3:N"` 在本质上等同于上一规则，这是因为 `tag3:N` 指定 `tag3` 不打印。
- `tag3:N` 在规则 `"tag1:I tag2:W tag3:N *:V"` 中更有意义，这是因为如果没有 `tag3:N`，`tag3` 信息就可能打印出来了；`tag1` 和 `tag2` 错误信息会打印在指定的详细度级别（或更低级别），并默认打印所有内容。

高级筛选规则示例 如下日志是在没有设置任何筛选选项的情况下获得的:

```
load:0x40078000,len:13564
entry 0x40078d4c
E (31) esp_image: image at 0x30000 has invalid magic byte
W (31) esp_image: image at 0x30000 has invalid SPI mode 255
E (39) boot: Factory app partition is not bootable
I (568) cpu_start: Pro cpu up.
I (569) heap_init: Initializing. RAM available for dynamic allocation:
I (603) cpu_start: Pro cpu start user code
D (309) light_driver: [light_init, 74]:status: 1, mode: 2
D (318) vfs: esp_vfs_register_fd_range is successful for range <54; 64) and VFS ID␣
↪1
I (328) wifi: wifi driver task: 3ffdbf84, prio:23, stack:4096, core=0
```

`--print_filter="wifi esp_image:E light_driver:I"` 筛选选项捕获的输出如下所示:

```
E (31) esp_image: image at 0x30000 has invalid magic byte
I (328) wifi: wifi driver task: 3ffdbf84, prio:23, stack:4096, core=0
```

`--print_filter="light_driver:D esp_image:N boot:N cpu_start:N vfs:N wifi:N *:V"` 选项的输出如下:

```
load:0x40078000,len:13564
entry 0x40078d4c
I (569) heap_init: Initializing. RAM available for dynamic allocation:
D (309) light_driver: [light_init, 74]:status: 1, mode: 2
```

配置文件

esp-idf-monitor 使用 C0 控制字符 与控制台进行交互。配置文件中的字符会被转换为对应的 C0 控制代码。可用字符包括英文字母 (A-Z) 和特殊符号：[、]、\、^、和 _.

警告： 注意，一些字符可能无法在所有平台通用，或被保留作为其他用途的快捷键。请谨慎使用此功能。

文件位置 配置文件的默认名称为 esp-idf-monitor.cfg。首先，在 esp-idf-monitor 路径中检测配置文件并运行。

如果此目录中没有检测到配置文件，则检查当前用户操作系统的配置目录：

- Linux: /home/<user>/.config/esp-idf-monitor/
- MacOS /Users/<user>/.config/esp-idf-monitor/
- Windows: c:\Users\<user>\AppData\Local\esp-idf-monitor\

如仍未检测到配置文件，会最后再检查主目录：

- Linux: /home/<user>/
- MacOS /Users/<user>/
- Windows: c:\Users\<user>\

在 Windows 中，可以使用 HOME 或 USERPROFILE 环境变量设置主目录，因此，Windows 配置目录的位置也取决于这些变量。

还可以使用 ESP_IDF_MONITOR_CFGFILE 环境变量为配置文件指定一个不同的位置，例如 ESP_IDF_MONITOR_CFGFILE = ~/custom_config.cfg。这一设置的检测优先级高于上述所有位置检测的优先级。

如果没有使用其他配置文件，esp-idf-monitor 会从其他常用的配置文件中读取设置。如果存在 setup.cfg 或 tox.ini 文件，esp-idf-monitor 会自动从这些文件中读取设置。

配置选项 下表列出了可用的配置选项：

选项名称	描述	默认值
menu_key	访问主菜单	T
exit_key	退出监视器	]
chip_reset_key	初始化芯片重置	R
recompile_upload_key	重新编译并上传	F
recompile_upload_app_key	仅重新编译并上传应用程序	A
toggle_output_key	切换输出显示	Y
toggle_log_key	切换日志功能	L
toggle_timestamp_key	切换时间戳显示	I
chip_reset_bootloader_key	将芯片重置为引导加载模式	P
exit_menu_key	从菜单中退出监视器	X
skip_menu_key	设置使用菜单命令时无需按下主菜单键	False
custom_reset_sequence	复位目标到引导加载程序的自定义复位序列	无默认值

语法 配置文件为.ini 文件格式，必须以 [esp-idf-monitor] 标头引入才能被识别为有效文件。以下语法以“配置名称 = 配置值”形式列出。以 # 或 ; 开头的行是注释，将被忽略。

```
# esp-idf-monitor.cfg file to configure internal settings of esp-idf-monitor
[esp-idf-monitor]
menu_key = T
exit_key = ]
chip_reset_key = R
```

(下页继续)

(续上页)

```
recompile_upload_key = F
recompile_upload_app_key = A
toggle_output_key = Y
toggle_log_key = L
toggle_timestamp_key = I
chip_reset_bootloader_key = P
exit_menu_key = X
skip_menu_key = False
```

IDF 监视器已知问题

如果在使用 IDF 监视器过程中遇到任何问题，请查看我们的 GitHub 仓库 以获取已知问题列表及其当前状态。如果遇到的问题没有相关记录，请创建一个新的问题报告。

4.22.3 IDF Docker 镜像

IDF Docker 镜像 (`espressif/idf`) 为使用特定版本的 ESP-IDF 自动化构建应用程序和库而设计。

该镜像包含以下内容：

- 常见的实用工具，如 `git`、`wget`、`curl` 和 `zip`。
- Python 3.8 或更高版本。
- 特定版本 ESP-IDF 的副本。有关版本信息，请参阅下文。该副本中设置了 `IDF_PATH` 环境变量，并指向容器中 ESP-IDF 的位置。
- 构建特定版本 ESP-IDF 所需工具：CMake、Ninja、交叉编译器工具链等。
- ESP-IDF 需要的所有 Python 软件包。这些软件包均已安装在虚拟环境中。

镜像 `ENTRYPOINT` 会设置 `PATH` 环境变量，指向正确版本的工具，并激活 Python 虚拟环境。此时，环境已经准备好，可以使用 ESP-IDF 构建系统。

如需使用其他工具，可用该镜像作为基础自定义镜像。

标签

该镜像维护了以下多个标签：

- `latest`：跟踪 ESP-IDF 的 `master` 分支
- `vX.Y`：对应 ESP-IDF 的版本 `vX.Y`
- `release-vX.Y`：跟踪 ESP-IDF 的 `release/vX.Y` 分支

备注： 在引入镜像功能前发布的 ESP-IDF 版本没有对应的 Docker 镜像版本。要查找最新可用标签列表，请参阅 https://hub.docker.com/r/espressif/idf/tags。

使用 Docker

设置 Docker 在本地使用 `espressif/idf` Docker 镜像前，请确保已安装 Docker。如果本地未安装 Docker，请按 https://docs.docker.com/install/ 提供的说明完成安装。

如果在 CI 环境中使用该镜像，请参阅 CI 服务说明文档，了解如何指定用于构建的镜像。

使用 CMake 构建项目 在项目目录下，运行以下命令：

```
docker run --rm -v $PWD:/project -w /project -u $UID -e HOME=/tmp espressif/idf␣
↪idf.py build
```

该命令具体内容如下：

- `docker run`：运行 Docker 镜像。此为 `docker container run` 命令的缩写形式。
- `--rm`：构建完成后删除相应容器。
- `-v $PWD:/project`：将主机当前目录 (`$PWD`) 挂载为容器中的 `/project` 目录。
- `-w /project`：使 `/project` 成为当前命令的工作目录。
- `-u $UID`：以当前用户的 ID 运行命令，使文件以当前用户而非 root 用户的身份创建。
- `-e HOME=/tmp`：为用户提供一个主目录，用于将 `idf.py` 创建的临时文件保存在 `~/.cache` 中。
- `espressif/idf`：使用标签为 `latest` 的 Docker 镜像 `espressif/idf`。未指定标签时，Docker 会隐式添加 `latest` 标签。
- `idf.py build`：在容器内运行此命令。

备注： 如果挂载目录 `/project` 包含的 git 仓库的用户 (`UID`) 不同于运行 Docker 容器的用户，在 `/project` 中执行 git 命令可能会失败，并显示错误信息 `fatal: detected dubious ownership in repository at '/project'`。如需解决此问题，可以在启动 Docker 容器时设置 IDF_GIT_SAFE_DIR 环境变量，将 `/project` 目录指定为安全目录。例如，可以将 `-e IDF_GIT_SAFE_DIR='/project'` 作为参数包含，还可以使用分隔符 `:` 指定多个目录，或使用 `*` 完全禁用此项 git 安全检查。

要以特定 Docker 镜像标签进行构建，请将其指定为 `espressif/idf:TAG`，示例如下：

```
docker run --rm -v $PWD:/project -w /project -u $UID -e HOME=/tmp espressif/
↪idf:release-v4.4 idf.py build
```

要查看最新可用标签列表，请参阅 https://hub.docker.com/r/espressif/idf/tags。

交互使用镜像 Docker 也支持以交互方式进行构建，以调试构建问题或测试自动构建脚本。请使用 `-i -t` 标志启动容器，示例如下：

```
docker run --rm -v $PWD:/project -w /project -u $UID -e HOME=/tmp -it espressif/idf
```

接着在容器内部照常使用 `idf.py`：

```
idf.py menuconfig
idf.py build
```

备注： 若未将串行接口传递到容器中，则 `idf.py flash` 和 `idf.py monitor` 等与开发板通信的命令在容器中无法正常工作。对于 Linux 系统，可以使用 设备选项 将串行接口传递到容器中。然而，目前 Windows 系统 (https://github.com/docker/for-win/issues/1018) 和 Mac 系统 (https://github.com/docker/for-mac/issues/900) 中 Docker 不支持此功能。可以使用 远程串行接口 克服此限制。有关如何执行此操作，请参阅以下*使用远程串行接口* 章节。

备注： 对于 Linux 系统，如果使用 `--device` 或 `--privileged` 等选项将主机的串行接口设备添加到容器，并使用 `-u $UID` 以特定用户启动容器时，请确保此用户对设备具有读/写访问权限。使用 `--group-add` 选项可以将容器用户添加到分配给主机设备的组 ID 中。例如，如果主机设备被分配到 `dialout` 组，你可以使用 `--group-add $(getent group dialout | cut -d':' -f3)` 将容器用户添加到主机的 `dialout` 组。

使用远程串行接口 RFC2217 (Telnet) 协议可用于远程连接到串行接口，详情请参阅 ESP 工具项目的 远程串行接口 文档。如果无法直接访问 Docker 容器内的串行接口，也可使用该协议进行访问。以下示例展

示了如何从 Docker 容器内部使用烧写命令。

在主机上安装并启动 esp_rfc2217_server：

- 在 Windows 系统中，该软件包以一个文件的形式提供，这个文件是由 pyinstaller 创建的可执行文件，可以从 ESP 工具版本 页面以 ZIP 压缩文件的形式与其他 ESP 工具一起下载：

```
esp_rfc2217_server -v -p 4000 COM3
```

- 在 Linux 或 macOS 系统中，该软件包是 esptool 的组成部分，可以在 ESP-IDF 环境中找到，或使用以下 pip 命令安装：

```
pip install esptool
```

随后执行以下命令启动服务器：

```
esp_rfc2217_server.py -v -p 4000 /dev/ttyUSB0
```

此时，便可使用以下命令，从 Docker 容器内部烧写连接到主机的设备：

```
docker run --rm -v <host_path>:/<container_path> -w /<container_path> espressif/
↪idf idf.py --port 'rfc2217://host.docker.internal:4000?ign_set_control' flash
```

请确保将 <host_path> 正确设置为主机上的项目路径，并使用 -w 选项将 <container_path> 设置为容器内的工作目录。host.docker.internal 为特殊的 Docker DNS 名称，用于访问主机。如有需要，可以将其替换为主机的 IP 地址。

构建自定义镜像

ESP-IDF 库中的 Docker 文件提供了以下构建参数，可用于构建自定义 Docker 镜像：

- IDF_CLONE_URL：克隆 ESP-IDF 存储库的 URL。在使用 ESP-IDF 分支时，可以将该参数设置为自定义 URL，默认值为 https://github.com/espressif/esp-idf.git。
- IDF_CLONE_BRANCH_OR_TAG：克隆 ESP-IDF 时使用的 git 分支或标签的名称。该参数将作为 git clone 命令的 --branch 参数传递，默认值为 master。
- IDF_CHECKOUT_REF：如果将此参数设置为非空值，在克隆之后会执行 git checkout $IDF_CHECKOUT_REF 命令。可以将此参数设置为特定 commit 的 SHA 值，以便切换到所需的版本分支或 commit。例如，在希望使用特定版本分支上的某个 commit 时，就可以将此参数设置为该 commit 的 SHA 值。
- IDF_CLONE_SHALLOW：如果将此参数设置为非空值，则会在执行 git clone 时使用 --depth=1 --shallow-submodules 参数。浅克隆的深度可以使用 IDF_CLONE_SHALLOW_DEPTH 设置。浅克隆可以极大减少下载的数据量及生成的 Docker 镜像大小。然而，如果需要切换到此类“浅层”存储库中的其他分支，必须先执行额外的 git fetch origin <branch> 命令。
- IDF_CLONE_SHALLOW_DEPTH：此参数指定进行浅克隆时要使用的深度值。如未设置，将使用 --depth=1。此参数仅在使用 IDF_CLONE_SHALLOW 时有效。如果要为分支构建 Docker 镜像，并且该镜像必须包含该分支上的最新标签，则需使用此参数。要确定所需的深度，请在特定的分支运行 git describe 命令，并注意偏移值。将偏移值加 1 后即可将其用作 IDF_CLONE_SHALLOW_DEPTH 参数的值。此过程将确保生成的镜像包含分支上的最新标签，且 Docker 镜像内部的 git describe 命令也会按预期工作。
- IDF_INSTALL_TARGETS：以逗号分隔的 ESP-IDF 目标列表，用于安装工具链，或者使用 all 安装所有目标的工具链。选择特定目标可以减少下载的数据量和生成的 Docker 镜像的大小。该参数默认值为 all。

要使用以上参数，请通过 --build-arg 命令行选项传递。例如，以下命令使用 ESP-IDF v4.4.1 的浅克隆以及仅适用于 ESP32-C3 的工具链构建了 Docker 镜像：

```
docker build -t idf-custom:v4.4.1-esp32c3 \
    --build-arg IDF_CLONE_BRANCH_OR_TAG=v4.4.1 \
    --build-arg IDF_CLONE_SHALLOW=1 \
    --build-arg IDF_INSTALL_TARGETS=esp32c3 \
    tools/docker
```

4.22.4 IDF Windows 安装程序

命令行参数

IDF Windows 安装程序 `esp-idf-tools-setup` 提供以下命令行参数：

- `/CONFIG=[PATH]` - 指定 `ini` 配置文件的路径，覆盖安装程序的默认配置。默认值：`config.ini`。
- `/GITCLEAN=[yes|no]` - 在以离线模式安装时，执行 `git clean` 命令，并删除未跟踪的目录。默认值：`yes`。
- `/GITRECURSIVE=[yes|no]` - 递归克隆所有 Git 仓库子模块。默认值：`yes`。
- `/GITREPO=[URL|PATH]` - 指定克隆 ESP-IDF 仓库的 URL。默认值：`https://github.com/espressif/esp-idf.git`。
- `/GITRESET=[yes|no]` - 在安装过程中，启用或禁用对仓库的 `git reset` 操作。默认值：`yes`。
- `/HELP` - 显示 Inno Setup 安装程序提供的命令行选项。
- `/IDFDIR=[PATH]` - 指定安装目录的路径。默认值：`{userdesktop}\esp-idf}`。
- `/IDFVERSION=[v4.3|v4.1|master]` - 使用指定的 ESP-IDF 版本，如 v4.1、v4.2、master。默认值：`empty`，选取列表中的第一个版本。
- `/IDFVERSIONSURL=[URL]` - 使用 URL 下载 ESP-IDF 版本列表。默认值：`https://dl.espressif.com/dl/esp-idf/idf_versions.txt`。
- `/LOG=[PATH]` - 在指定目录中存储安装日志文件。默认值：`empty`。
- `/OFFLINE=[yes|no]` - 在离线模式下，使用 `pip` 执行。通过设置环境变量 `PIP_NO_INDEX`，也可在离线模式下安装 Python 软件包。默认值：`no`。
- `/USEEMBEDDEDPYTHON=[yes|no]` - 使用嵌入式 Python 版本完成安装。将此参数设置为 `no`，可以在安装程序中选择 Python 版本。默认值：`yes`。
- `/PYTHONNOUSERSITE=[yes|no]` - 在启动任意 Python 命令前，设置 `PYTHONNOUSERSITE` 变量，避免从 AppDataRoaming 加载 Python 软件包。默认值：`yes`。
- `/PYTHONWHEELSURL=[URL]` - 指定 PyPi 存储库的 URL，以解析二进制 Python Wheel 依赖关系。设置环境变量 `PIP_EXTRA_INDEX_URL` 可以实现相同效果。默认值：`https://dl.espressif.com/pypi`。
- `/SKIPSYSTEMCHECK=[yes|no]` - 跳过系统检查页面。默认值：`no`。
- `/VERYSILENT /SUPPRESSMSGBOXES /SP- /NOCANCEL` - 执行静默安装。

静默安装

通过设置以下命令行参数，可以静默安装 ESP-IDF：

```
esp-idf-tools-setup-x.x.exe /VERYSILENT /SUPPRESSMSGBOXES /SP- /NOCANCEL
```

在命令行中运行安装程序时，它会在后台启动一个独立的进程执行安装操作，而不会阻塞命令行的使用。通过以下 PowerShell 脚本可以等待安装程序完成：

```
esp-idf-tools-setup-x.x.exe /VERYSILENT /SUPPRESSMSGBOXES /SP- /NOCANCEL
$InstallerProcess = Get-Process esp-idf-tools-setup
Wait-Process -Id $InstallerProcess.id
```

自定义 Python 版本及 Python Wheel 位置

IDF 安装程序默认使用嵌入的 Python 版本，并参考预定义的 Python Wheel 镜像获取所需软件包。

以下参数支持自定义 Python 版本及 Python Wheel 位置：

```
esp-idf-tools-setup-x.x.exe /USEEMBEDDEDPYTHON=no /PYTHONWHEELSURL=https://pypi.
↪org/simple/
```

4.22.5 IDF 组件管理器

IDF 组件管理器工具用于下载 ESP-IDF CMake 项目的依赖项，该下载在 CMake 运行期间自动完成。IDF 组件管理器可以从 组件注册表 或 Git 仓库获取组件。

要获取组件列表，请参阅 https://components.espressif.com/.

有关 IDF 组件管理器的详细信息，请参阅 IDF 组件管理器及 ESP 组件注册表文档。

在项目中使用 IDF 组件管理器

项目中各组件的依赖项定义在单独的清单文件中，命名为 `idf_component.yml`，位于组件根目录。运行 `idf.py create-manifest` 可以为组件创建清单文件模板。默认情况下将为 main 组件创建清单文件。使用 `--path` 选项，可以显式指定创建清单文件的目录路径。使用 `--component=my_component` 选项可以指定组件名称，这样系统将会在 `components` 文件夹下为该组件创建清单文件。`create-manifest` 命令支持以下运行方式：

- `idf.py create-manifest` 为 main 组件创建清单文件
- `idf.py create-manifest --component=my_component` 在 `components` 目录下，为组件 **my_component** 创建清单文件
- `idf.py create-manifest --path="../../my_component"` 在 `my_component` 目录下，为组件 **my_component** 创建清单文件

在向项目的某个组件添加新的清单时，必须先运行 `idf.py reconfigure`，手动重新配置项目。随后，构建过程将跟踪 `idf_component.yml` 清单的变更，并在必要时自动触发 CMake。

要为 ESP-IDF 项目中的组件（如 `my_component`）添加依赖项，可以运行命令 `idf.py add-dependency DEPENDENCY`。`DEPENDENCY` 参数代表一个由 IDF 组件管理器管理的额外组件，而 `my_component` 也依赖于这个组件。`DEPENDENCY` 参数的格式为 `namespace/name=1.0.0`，*namespace/name* 代表组件名称，*=1.0.0* 是组件的版本范围，详情请参阅 版本文档。默认情况下，依赖项会添加到 main 组件。通过使用 `--path` 选项，可以显式指定包含清单的目录，也可以使用 `--component=my_component`，在 `components` 文件夹中指定组件。`add-dependency` 命令支持以下运行方式：

- `idf.py add-dependency example/cmp` 为 main 组件添加依赖项，依赖项为 `example/cmp` 的最新版本
- `idf.py add-dependency --component=my_component example/cmp<=3.3.3` 将依赖项添加到位于 `components` 目录下名为 `my_component` 的组件中，依赖项为版本号 `<=3.3.3` 的 `example/cmp`
- `idf.py add-dependency --path="../../my_component" example/cmp^3.3.3` 将依赖项添加到位于目录 `my_component` 下名为 `my_component` 的组件中，依赖项为版本号 `^3.3.3` 的 `example/cmp`

备注： `add-dependency` 命令会从 乐鑫组件注册表 将依赖项显式添加到你的项目中。

要更新 ESP-IDF 项目的依赖项，请运行命令 `idf.py update-dependencies`。你也可以使用 `--project-dir PATH` 选项，指定项目目录的路径。

应用程序示例 build_system/cmake/component_manager 使用了由组件管理器安装的组件。

对于不需要受管理依赖项的组件，则无需提供清单文件。

在 CMake 配置项目（如 `idf.py reconfigure`）时，组件管理器会执行以下操作：

- 处理项目中每个组件的 `idf_component.yml` 清单，并递归解析依赖项。
- 在项目根目录中创建 `dependencies.lock` 文件，包含完整的依赖项列表。
- 将所有依赖项下载至 `managed_components` 目录。

请勿更改 dependencies.lock 锁文件和 managed_components 目录的内容。组件管理器运行时，会始终确保这些文件处于最新状态。如果意外修改了这些文件，可以通过使用 idf.py reconfigure 触发 CMake，重新运行组件管理器。

设置构建属性 DEPENDENCIES_LOCK 可以指定顶层 CMakeLists.txt 文件中的锁文件路径。例如，在 project(PROJECT_NAME) 前添加 idf_build_set_property(DEPENDENCIES_LOCK dependencies.lock.${IDF_TARGET})，可以为不同目标生成不同锁文件。

从示例创建项目

组件注册表中，部分组件包含示例项目。要从示例创建一个新项目，可以运行命令 idf.py create-project-from-example EXAMPLE。EXAMPLE 参数格式为 namespace/name=1.0.0:example，*namespace/name* 代表组件名称，*=1.0.0* 是组件的版本范围（详情请参阅 版本文档），而 *example* 代表示例名称。在 乐鑫组件注册表 中，可以找到各组件的示例列表，以及启动组件示例的相应命令。

在清单文件中定义依赖项

通过在文本编辑器直接编辑，你可以轻松定义清单文件 idf_component.yml 中的依赖项。以下是有关定义依赖项的简单示例：

你可以通过指定组件名称和版本范围，定义来自注册表的依赖项：

```
dependencies:
  # 定义来自注册表 (https://components.espressif.com/component/example/cmp)␣
↪的依赖项
  example/cmp: ">=1.0.0"
```

要从 Git 仓库定义依赖关系，请提供组件在仓库中的路径和仓库的 URL：

```
dependencies:
  # 从 Git 仓库定义依赖项
  test_component:
    path: test_component
    git: ssh://git@gitlab.com/user/components.git
```

在开发组件时，可以通过指定相对或绝对路径，使用本地目录中的组件：

```
dependencies:
  # 通过相对路径定义本地依赖项
  some_local_component:
    path: ../../projects/component
```

有关清单文件格式的详细信息，请参阅 清单文件格式文档。

禁用组件管理器

将环境变量 IDF_COMPONENT_MANAGER 设置为 0，可以显式禁用组件管理器。

4.22.6 IDF clang-tidy

IDF clang-tidy 是使用 clang-tidy 对当前应用程序进行静态分析的工具。

警告： IDF clang-tidy 的功能及其依赖的工具链尚在开发中，最终版本发布前可能有重大变更。

目前仅支持基于 clang 的工具链。在配置项目前，必须在环境变量或 CMake 缓存中设置 `IDF_TOOLCHAIN=clang` 进行激活。

警告： 当前工具尚不支持基于 RISC-V 的芯片。目前，乐鑫尚未针对 RISC-V 提供基于 clang 的工具链。

准备工作

初次运行此工具时，请按照以下步骤准备该工具：

1. 运行 `idf_tools.py install esp-clang` 安装 clang-tidy 所需的二进制文件。

   备注： 该工具链尚在开发中，最终版本发布后，将无需手动安装工具链。

2. 再次运行导出脚本（如 `export.sh`、`export.bat` 等），刷新环境变量。

其他命令

`clang-check` 运行 `idf.py clang-check` 可以重新生成编译数据库，并在当前项目文件夹下运行 `clang-tidy`，所得输出写入 `<project_dir>/warnings.txt`。

运行 `idf.py clang-check --help` 查看完整文档。

`clang-html-report`

1. 运行 `pip install codereport` 安装附加依赖关系。
2. 运行 `idf.py clang-html-report` 会根据 `warnings.txt` 在 `<project_dir>/html_report` 文件夹内生成 HTML 报告。请在浏览器中打开 `<project_dir>/html_report/index.html` 查看报告。

错误报告

此工具托管在 espressif/clang-tidy-runner。如遇到任何错误，或有任何功能请求，请通过 Github issues 提交报告。

4.22.7 可下载的 ESP-IDF 工具

构建过程中，ESP-IDF 依赖许多工具，如交叉编译工具链、CMake 构建系统等。

如快速入门 所述，若所需工具版本可用，首选使用当前操作系统的软件包管理器（如 apt、yum、brew 等）安装相关工具。例如，在 Linux 和 macOS 系统中，建议用 Linux 和 macOS 系统的软件包管理器安装 CMake。

但部分 ESP-IDF 的特定工具在操作系统软件包存储库中不可用，且不同版本的 ESP-IDF 需相应使用不同版本的工具运行。为解决以上两个问题，ESP-IDF 提供了一组脚本，可以下载和安装正确的工具版本，并设置相应运行环境。

下文中，这类可下载的工具简称为 “工具”。除此类工具外，ESP-IDF 还使用以下工具：

- ESP-IDF 捆绑的 Python 脚本，如 `idf.py`
- 从 PyPI 安装的 Python 软件包

以下各小节介绍了可下载工具的安装方法，并提供了在不同平台上安装的工具列表。

备注： 本文档面向需要自定义其安装过程的高级用户、希望了解安装过程的用户以及 ESP-IDF 开发人员。要了解如何安装 ESP-IDF 工具，请参阅快速入门。

工具元数据文件

各平台所需工具及工具版本列表存放在 tools/tools.json 文件中，tools/tools_schema.json 定义了该文件的模式。

在安装工具或设置环境变量时，tools/idf_tools.py 脚本将使用上述文件。

工具安装目录

`IDF_TOOLS_PATH` 环境变量指定下载及安装工具的位置。若未设置该变量，Linux 和 macOS 系统的默认下载安装位置为 `HOME/.espressif`，Windows 系统的默认下载安装位置为 `%USER_PROFILE%\.espressif`。

在 `IDF_TOOLS_PATH` 目录下，工具安装脚本会创建以下子目录和文件：

- `dist` —工具存档下载位置。
- `tools` —工具解压缩位置。工具会解压缩到子目录 `tools/TOOL_NAME/VERSION/` 中，该操作支持同时安装不同版本的工具。
- `idf-env.json` — “目标 (target)” 和 “功能 (feature)” 等用户安装选项均存储在此文件中。“目标” 为选择需要安装和保持更新的工具的芯片目标；“功能” 则决定应安装哪些 Python 软件包。有关用户安装选项的详情，请参阅下文。
- `python_env` — 与工具无关；虚拟 Python 环境安装在其子目录中。注意，设置 `IDF_PYTHON_ENV_PATH` 环境变量可以将 Python 环境目录放置到其他位置。
- `espidf.constraints.*.txt` —每个 ESP-IDF 版本都有的约束文件，包含 Python 包版本要求。

GitHub 资源镜像

工具下载器下载的工具大多属于 GitHub 发布的资源，即在 GitHub 上伴随软件发布的文件。

如果无法访问 GitHub 下载或访问速度较慢，可以配置一个 GitHub 资源镜像。

要使用乐鑫下载服务器，请将环境变量 `IDF_GITHUB_ASSETS` 设置为 `dl.espressif.com/github_assets`，在国内下载时，也可设置为 `dl.espressif.cn/github_assets` 加快下载速度。安装过程中，当从 `github.com` 下载工具时，URL 将重写为使用乐鑫下载服务器。

只要 URL 与 `github.com` 的下载 URL 格式匹配，任何镜像服务器均可使用，安装过程中下载的 GitHub 资源 URL 将把 `https://github.com` 替换为 `https://${IDF_GITHUB_ASSETS}`。

备注： 目前，乐鑫下载服务器不会镜像 GitHub 上的所有内容，只镜像部分发布版本的附件资源文件及源文件。

`idf_tools.py` 脚本

ESP-IDF 随附的 tools/idf_tools.py 脚本具备以下功能：

- `install`：将工具下载到 `${IDF_TOOLS_PATH}/dist` 目录，并解压缩到 `${IDF_TOOLS_PATH}/tools/TOOL_NAME/VERSION`。
  `install` 命令接收 `TOOL_NAME` 或 `TOOL_NAME@VERSION` 格式的安装工具列表。如果给定参数 `all`，则会安装列表上的所有工具，包括必须项和可选项。如果没有给定参数，或给定参数为 `required`，则只安装必须项。

- download：与 install 类似，但不会解压缩工具。使用可选项 --platform 可下载特定平台的工具。
- export：列出使用已安装工具前应设置的环境变量。对多数工具而言，只需要设置环境变量 PATH，但也有些工具需要设置额外的环境变量。
  环境变量可以以 shell 或 key-value 格式列出，使用 --format 参数设置该选项。
  - export 可选参数：
    - --unset：该参数可用于创建语句，取消特定全局变量设置，使环境恢复到调用 export.{sh/fish} 前的状态。
    - --add_paths_extras：该参数将 $PATH 中与 ESP-IDF 相关的额外路径添加到 ${IDF_TOOLS_PATH}/esp-idf.json 中，以保证在退出当前 ESP-IDF 环境时删除全局变量。例如，在运行 export.{sh/fish} 脚本时，如果在全局变量 $PATH 中添加了新的路径，在命令中添加该参数可以将这些新路径保存到 ${IDF_TOOLS_PATH}/esp-idf.json 文件中。
  - shell：生成适合在 shell 中执行的输出，例如，在 Linux 和 macOS 上生成以下输出

    ```
    export PATH="/home/user/.espressif/tools/tool/v1.0.0/bin:$PATH"
    ```

    在 Windows 上生成以下输出

    ```
    set "PATH=C:\Users\user\.espressif\tools\v1.0.0\bin;%PATH%"
    ```

    备注： 当前不支持以 Powershell 格式导出环境变量，可以用 key-value 格式代替。

    如果 shell 支持，则该命令的输出可用于更新环境变量。例如

    ```
    eval $($IDF_PATH/tools/idf_tools.py export)
    ```

  - key-value：以 VARIABLE=VALUE 格式生成输出，以便其他脚本解析

    ```
    PATH=/home/user/.espressif/tools/tool/v1.0.0:$PATH
    ```

    注意，用于处理此输出的脚本必须对输出中的 $VAR 或 %VAR% 模式进行扩展，即解析成对应变量。
- list：列出已知的工具版本，并指示哪些版本已安装。
  以下选项可用于自定义输出。
  - --outdated：仅列出安装在 IDF_TOOLS_PATH 中的过时版本工具。
- check：检查每个工具是否在系统路径和 IDF_TOOLS_PATH 中可用。
- install-python-env：在 ${IDF_TOOLS_PATH}/python_env 目录或直接在 IDF_PYTHON_ENV_PATH 环境变量设置的目录中创建 Python 虚拟环境，并在其中安装所需的 Python 软件包。
  - 参数 --features 为可选项，用于指定要添加或删除的功能列表，功能之间用逗号分隔。
    1. 该参数将删除以 - 开头的功能，添加以 + 开头或无符号标记的功能。例如，要删除功能 XY，示例语法为 --features=-XY；要添加功能 XY，示例语法为 --features=+XY 或 --features=XY。如果为同一功能同时提供了删除和添加选项，则不执行任何操作。
    2. 每个功能都必须有依赖文件。例如，只有当 ${IDF_PATH}/tools/requirements/requirements.XY.txt 文件已存在，并包含要安装的 Python 包列表时，功能 XY 才有效。
    3. core 功能为必须项，确保 ESP-IDF 的核心功能，如控制台中的构建、烧录、监视器、调试等。
    4. 用户可选择任意数量的可选功能，已选功能列表存储在 idf-env.json 中。
    5. 依赖文件中存储了需要安装的 Python 包以及 espidf.constraints.*.txt 文件，该约束文件从 https://dl.espressif.com 下载，并存储在 ${IDF_TOOLS_PATH} 目录，包含了针对特定 ESP-IDF 版本的安装包版本要求。

  备注： 可以通过使用 --no-constraints 参数或将环境变量 IDF_PYTHON_CHECK_CONSTRAINTS 设置为 no，禁用约束文件的下载和使用，但 **并不建议此做法**。

- check-python-dependencies：检查所有必需的 Python 包是否均已安装。该命令会对比检查由 idf-env.json 功能列表从 ${IDF_PATH}/tools/requirements/requirements.*.

txt 所选择的软件包与 espidf.constraints.*.txt 文件指定的软件包版本是否一致。

备注： 约束文件可通过 install-python-env 命令下载。与 install-python-env 命令类似，可以通过使用 --no-constraints 参数或将环境变量 IDF_PYTHON_CHECK_CONSTRAINTS 设置为 no，禁止使用约束文件。

- uninstall：打印并删除当前 ESP-IDF 版本未使用的工具。
  - --dry-run 打印已安装但未使用的工具。
  - --remove-archives 删除过去下载的所有旧版本软件安装包。

安装脚本

ESP-IDF 的根目录中提供了针对不同 shell 的用户安装脚本，包括：

- install.bat 适用于 Windows 命令提示符
- install.ps1 适用于 Powershell
- install.sh 适用于 Bash
- install.fish 适用于 Fish

这些脚本除了下载和安装 IDF_TOOLS_PATH 中的工具外，还会准备一个 Python 虚拟环境，并在此虚拟环境中安装所需软件包。

为启用相应功能，这些脚本可以选择性地接受一组以逗号分隔的芯片目标列表及 --enable-* 参数，这类参数会传递给 idf_tools.py 脚本，并由该脚本将这类参数存储在 idf-env.json 中，从而逐步启用芯片目标及功能。

要为所有芯片目标安装工具，请在不使用任何可选参数的情况下，运行 idf_tools.py install --targets=all。要安装具备 ESP-IDF 核心功能的 Python 软件包，请运行 idf_tools.py install-python-env --features=core。

也可为特定芯片安装工具，例如，运行 install.sh esp32 可以只为 ESP32 安装工具。更多相关示例，请参阅第三步：设置工具。

运行 idf_tools.py install-python-env --features=core,XY，install.sh --enable-XY 可以启用 XY 功能。

导出脚本

由于安装好的工具并非永久添加到用户或系统的 PATH 环境变量中，因此，要在命令行中使用这些工具，还需要额外步骤。以下脚本会修改当前 shell 的环境变量，从而使用正确版本的工具：

- export.bat 适用于 Windows 命令提示符
- export.ps1 适用于 Powershell
- export.sh 适用于 Bash
- export.fish 适用于 Fish

备注： 在 Bash 中修改 shell 环境时，必须使用 . ./export.sh 命令加载 export.sh，注意添加前面的点和空格。

export.sh 可以在除了 Bash 外的其他 shell（如 zsh）中使用。但在这种情况下，必须在运行脚本前设置 IDF_PATH 环境变量。在 Bash 中使用时，脚本会从当前目录猜测 IDF_PATH 的值。

除了调用 idf_tools.py，这些脚本还会列出已经添加到 PATH 的目录。

其他安装方法

为适用于不同环境，ESP-IDF 提供了更多用户友好的 idf_tools.py 包装工具：

- *ESP-IDF 工具安装器* 支持下载和安装工具，其内部使用 idf_tools.py 实现功能。

- ESP-IDF Eclipse 插件 包括了一个用于设置工具的菜单项，该插件内部调用 `idf_tools.py`。
- VSCode ESP-IDF 扩展 提供了设置工具的入门流程。尽管此扩展包不依赖 `idf_tools.py`，但安装方法相同。

自定义安装

推荐用户使用上述方法安装 ESP-IDF 工具，但也可以选择其他方式来构建 ESP-IDF 应用程序。自定义安装时，用户需将所有必要的工具都安装在某个位置，并在 `PATH` 中定义，以保证 ESP-IDF 构建系统可用。

卸载 ESP-IDF

卸载 ESP-IDF 需要删除安装过程中配置的工具和环境变量。

- 使用 *Windows ESP-IDF 工具安装器* 的 Windows 用户可以直接运行卸载向导卸载 ESP-IDF。
- *工具安装目录* 下包含了已下载及安装的工具，删除该目录即可删除此前通过运行*安装脚本* 安装的内容。通过*导出脚本* 设置的环境变量不具备永久性，新环境中不会存在此类环境变量。
- 如在安装过程中进行了自定义设置，除删除上述工具外，可能还涉及手动恢复此前为适用 ESP-IDF 工具而修改的环境变量及系统路径，例如 `IDF_PYTHON_ENV_PATH` 或 `IDF_TOOLS_PATH`。如存在通过手动复制安装的工具，则也需手动追踪并删除相关文件。
- 如安装了 ESP-IDF Eclipse 或 VSCode ESP-IDF 扩展程序 等插件，则需按照对应插件文档中的特定卸载说明进行操作。

备注： 卸载 ESP-IDF 工具不会删除任何项目文件或用户代码。为防止意外丢失其他文件，请在删除文件时谨慎操作。如果对某个步骤的操作有所疑问，请参考前述安装说明。

上述卸载指南默认需卸载的工具是按本文档中的步骤进行安装的。如果使用了自定义安装，可能需要进行相应调整。

ESP-IDF 工具列表

xtensa-esp-elf-gdb GDB for Xtensa

License: GPL-3.0-or-later

More info: https://github.com/espressif/binutils-gdb

Platform	Required	Download
linux-amd64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-x86_64-linux-gnu.tar.gz SHA256: 9d68472d4cba5cf8c2b79d94f86f92c828e76a632bd1e6be5e7706e5b304d36e
linux-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-aarch64-linux-gnu.tar.gz SHA256: bdabc3217994815fc311c4e16e588b78f6596b5ad4ffa46c80b40e982cfb1e66
linux-armel	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-arm-linux-gnueabi.tar.gz SHA256: d54b8d703ba897b28c627da3d27106a3906dd01ba298778a67064710bc33c76d
linux-armhf	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-arm-linux-gnueabihf.tar.gz SHA256: 6187d1dd54e57927f7a7b804ff431fe0a295d5d5638c7654ee2bb7c3e0e84d4b
linux-i686	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-i586-linux-gnu.tar.gz SHA256: 64d3bc992ed8fdec383d49e8b803ac494605a38117c8293db8da055037de96b0
macos	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-x86_64-apple-darwin14.tar.gz SHA256: 023e74b3fda793da4bc0509b02de776ee0dad6efaaac17bef5916fb7dc9c26b9
macos-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-aarch64-apple-darwin21.1.tar.gz SHA256: ea757c6bf8c25238f6d2fdcc6bbab25a1b00608a0f9e19b7ddd2f37ddbdc3fb1
win32	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-i686-w64-mingw32.zip SHA256: 322e8d9b700dc32d8158e3dc55fb85ec55de48d0bb7789375ee39a28d5d655e2
win64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/xtensa-esp-elf-gdb-14.2_20240403-x86_64-w64-mingw32.zip SHA256: a27a2fe20f192f8e0a51b8936428b4e1cf8935cfe008ee445cc49f6fc7f6db2e

riscv32-esp-elf-gdb GDB for RISC-V

License: GPL-3.0-or-later

More info: https://github.com/espressif/binutils-gdb

Platform	Required	Download
linux-amd64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-x86_64-linux-gnu.tar.gz SHA256: ce004bc0bbd71b246800d2d13b239218b272a38bd528e316f21f1af2db8a4b13
linux-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-aarch64-linux-gnu.tar.gz SHA256: ba10f2866c61410b88c65957274280b1a62e3bed05131654ed9b6758efe18e55
linux-armel	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-arm-linux-gnueabi.tar.gz SHA256: 88539db5d987f28827efac7e26080a2803b9b539342ccd2963ccfdd56d7f08f7
linux-armhf	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-arm-linux-gnueabihf.tar.gz SHA256: b45b9711d6a87d4c2f688a9599ce850ce02f477756e3e797c4a6c1c549127fcb
linux-i686	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-i586-linux-gnu.tar.gz SHA256: 0e628ee37438ab6ba05eb889a76d09e50cb98e0020a16b8e2b935c5cf19b4ed2
macos	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-x86_64-apple-darwin14.tar.gz SHA256: 8f6bda832d70dad5860a639d55aba4237bd10cbac9f4822db1eece97357b34a9
macos-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-aarch64-apple-darwin21.1.tar.gz SHA256: d88b6116e86456c8480ce9bc95aed375a35c0d091f1da0a53b86be0e6ef3d320
win32	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-i686-w64-mingw32.zip SHA256: d6e7ce05805b0d8d4dd138ad239b98a1adf8da98941867d60760eb1ae5361730
win64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v14.2_20240403/riscv32-esp-elf-gdb-14.2_20240403-x86_64-w64-mingw32.zip SHA256: 5c9f211dc46daf6b96fad09d709284a0f0186fef8947d9f6edd6bca5b5ad4317

xtensa-esp-elf Toolchain for 32-bit Xtensa based on GCC

License: GPL-3.0-with-GCC-exception

More info: https://github.com/espressif/crosstool-NG

Platform	Required	Download
linux-amd64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-x86_64-linux-gnu.tar.xz SHA256: fcef03d87eac44c0dbee2bbee98443ed2fcf82720dcd8ebfe00640807b0f07c2
linux-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-aarch64-linux-gnu.tar.xz SHA256: cfe55b92b4baeaa4309a948ba65e2adfc2d17a542c64856e36650869b419574a
linux-armel	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-arm-linux-gnueabi.tar.xz SHA256: c57a062969ec3d98b02a97cd9240eb31091957788509b60c356b0a6f23032669
linux-armhf	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-arm-linux-gnueabihf.tar.xz SHA256: 1adc660f4d7bcf863f54051c5843719456fabc7203c1d4ccbb855924fda82987
linux-i686	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-i586-linux-gnu.tar.xz SHA256: f9203673aa0c42b041847c86b07e6f5b4aa9c90e6ff03d3cd3146928784447ea
macos	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-x86_64-apple-darwin.tar.xz SHA256: 39ee7df749f4ceb93624d73627688d5b86269a7429022f986f2940499936aacd
macos-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-aarch64-apple-darwin.tar.xz SHA256: d967e49a64f823e18fbae273efb1b094ac55e2207aa21fd3947c9d59f999f47e
win32	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-i686-w64-mingw32_hotfix.zip SHA256: d6b227c50e3c8e21d62502b3140e5ab74a4cb502c2b4169c36238b9858a8fb88
win64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/xtensa-esp-elf-13.2.0_20240530-x86_64-w64-mingw32_hotfix.zip SHA256: 155ee97b531236e6a7c763395c68ca793e55e74d2cb4d38a23057a153e01e7d0

esp-clang Toolchain for all Espressif chips based on clang

License: Apache-2.0

More info: https://github.com/espressif/llvm-project

Platform	Required	Download
linux-amd64	optional	https://github.com/espressif/llvm-project/releases/download/esp-16.0.0-20230516/llvm-esp-16.0.0-20230516-linux-amd64.tar.xz SHA256: 3dbd8dd290913a93e8941da8a451ecd49f9798cc2d74bb9b63ef5cf5c4fee37f
linux-arm64	optional	https://github.com/espressif/llvm-project/releases/download/esp-16.0.0-20230516/llvm-esp-16.0.0-20230516-linux-arm64.tar.xz SHA256: 4b115af6ddd04a9bffc1908fc05837998ee71d450891d741c446186f2aa9b961
linux-armhf	optional	https://github.com/espressif/llvm-project/releases/download/esp-16.0.0-20230516/llvm-esp-16.0.0-20230516-linux-armhf.tar.xz SHA256: 935082bb0704420c5ca42b35038bba8702135348a50cac454ae2fb55af0b4c32
macos	optional	https://github.com/espressif/llvm-project/releases/download/esp-16.0.0-20230516/llvm-esp-16.0.0-20230516-macos.tar.xz SHA256: d9824acafd3e7b1d17ace084243b82a95bbdcb149a26b085bba487ab3d3716d7
macos-arm64	optional	https://github.com/espressif/llvm-project/releases/download/esp-16.0.0-20230516/llvm-esp-16.0.0-20230516-macos-arm64.tar.xz SHA256: ed5621396dc3e48413e14e8b6caed8e2993e7f2ab5fca1410081f40c940a1060
win64	optional	https://github.com/espressif/llvm-project/releases/download/esp-16.0.0-20230516/llvm-esp-16.0.0-20230516-win64.tar.xz SHA256: 598c8241c8bf10fd1be8bd21845307cfc404e127041b4ba4e828350a88692883

riscv32-esp-elf Toolchain for 32-bit RISC-V based on GCC

License: GPL-3.0-with-GCC-exception

More info: https://github.com/espressif/crosstool-NG

Platform	Required	Download
linux-amd64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-x86_64-linux-gnu.tar.xz SHA256: f69a491d2f42f63e119f9077da995f7743ea8e1bf6944166a42a312cf60728a8
linux-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-aarch64-linux-gnu.tar.xz SHA256: 276351b883a53e81b695d858be74114a8b627bbe4fc9c69ef46a7127ab143680
linux-armel	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-arm-linux-gnueabi.tar.xz SHA256: 14890f2a624e70f11da7268347adf25b6c396f42bcd4d8ac3c5bfa4050b7c934
linux-armhf	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-arm-linux-gnueabihf.tar.xz SHA256: b61ca9ceff25986ec1d166a01319bff09639be1d4ee5bf117502ce564fdae7e9
linux-i686	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-i586-linux-gnu.tar.xz SHA256: 12ef50f96deb9040ce360974a4237c64ae0706b0c429b90cecc8ab664cf6dbb4
macos	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-x86_64-apple-darwin.tar.xz SHA256: cfbf5deaba05bf217701c8ceab7396bb0c2ca95ab58e134d4b2e175b86c2fd6c
macos-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-aarch64-apple-darwin.tar.xz SHA256: 230628fcf464ca8856c82c55514e40a8919e97fbc5e66b7165ca42c9653d2302
win32	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-i686-w64-mingw32.zip SHA256: 590bfb10576702639825581cc00c445da6e577012840a787137417e80d15f46d
win64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-13.2.0_20240530/riscv32-esp-elf-13.2.0_20240530-x86_64-w64-mingw32.zip SHA256: 413eb9f6adf8fdaf25544d014c850fc09eb38bb93a2fc5ebd107ab1b0de1bb3a

esp32ulp-elf Toolchain for ESP32 ULP coprocessor

License: GPL-3.0-or-later

More info: https://github.com/espressif/binutils-gdb

Platform	Required	Download
linux-amd64	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-linux-amd64.tar.gz SHA256: d13a808365b78465fa6591636dfbbb9604d9d15a397c3d9cd22626d54828ac2c
linux-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-linux-arm64.tar.gz SHA256: ecce0788ce1000e5c669c5adaf2fd5bf7f9bf96dcdbd3555d1d9ce4dcb311038
linux-armel	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-linux-armel.tar.gz SHA256: 7228b01277f7908d72eb659470f82e143c4c66b444538a464290d88ece16130e
linux-armhf	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-linux-armhf.tar.gz SHA256: 951b089c66561bc2190a8d57c316dfaef985a778728a7c30e1edcd29fe180016
linux-i686	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-linux-i686.tar.gz SHA256: df323d40962313168f6feeb2d9471c6010ff23a7896f40244e62991517d9745b
macos	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-macos.tar.gz SHA256: b2aeba8eaafdf156e9e30be928dde1f133b00eaf33802d96827ec544ac7c864c
macos-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-macos-arm64.tar.gz SHA256: e3a4dfea043e2bce8cd00b3a0b260a59249fa61ca5931bf02f18a3d43c18deb4
win32	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-win32.zip SHA256: d33b64f49df27dcfa4a24d3af1a5ead77b020f85f33448994c31b98f88e66bb4
win64	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-2.38_20240113/esp32ulp-elf-2.38_20240113-win64.zip SHA256: 3a7627008ac92d1580542b95c696449e56aaa1d0881dc3ef5fd5c60afc77a49d

cmake CMake build system

On Linux and macOS, it is recommended to install CMake using the OS-specific package manager (like apt, yum, brew, etc.). However, for convenience it is possible to install CMake using idf_tools.py along with the other tools.

License: BSD-3-Clause

More info: https://github.com/Kitware/CMake

Platform	Required	Download
linux-amd64	optional	https://github.com/Kitware/CMake/releases/download/v3.30.2/cmake-3.30.2-linux-x86_64.tar.gz SHA256: cdd7fb352605cee3ae53b0e18b5929b642900e33d6b0173e19f6d4f2067ebf16
linux-arm64	optional	https://github.com/Kitware/CMake/releases/download/v3.30.2/cmake-3.30.2-linux-aarch64.tar.gz SHA256: d18f50f01b001303d21f53c6c16ff12ee3aa45df5da1899c2fe95be7426aa026
linux-armel	optional	https://dl.espressif.com/dl/cmake/cmake-3.30.2-Linux-armv7l.tar.gz SHA256: 446650c69ea74817a770f96446c162bb7ad24ffecaacb35fcd4845ec7d3c9099
linux-armhf	optional	https://dl.espressif.com/dl/cmake/cmake-3.30.2-Linux-armv7l.tar.gz SHA256: 446650c69ea74817a770f96446c162bb7ad24ffecaacb35fcd4845ec7d3c9099
macos	optional	https://github.com/Kitware/CMake/releases/download/v3.30.2/cmake-3.30.2-macos-universal.tar.gz SHA256: c6fdda745f9ce69bca048e91955c7d043ba905d6388a62e0ff52b681ac17183c
macos-arm64	optional	https://github.com/Kitware/CMake/releases/download/v3.30.2/cmake-3.30.2-macos-universal.tar.gz SHA256: c6fdda745f9ce69bca048e91955c7d043ba905d6388a62e0ff52b681ac17183c
win32	required	https://github.com/Kitware/CMake/releases/download/v3.30.2/cmake-3.30.2-windows-x86_64.zip SHA256: 48bf4b3dc2d668c578e0884cac7878e146b036ca6b5ce4f8b5572f861b004c25
win64	required	https://github.com/Kitware/CMake/releases/download/v3.30.2/cmake-3.30.2-windows-x86_64.zip SHA256: 48bf4b3dc2d668c578e0884cac7878e146b036ca6b5ce4f8b5572f861b004c25

openocd-esp32 OpenOCD for ESP32

License: GPL-2.0-only

More info: https://github.com/espressif/openocd-esp32

Platform	Required	Download
linux-amd64	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20241016/openocd-esp32-linux-amd64-0.12.0-esp32-20241016.tar.gz SHA256: e82b0f036dc99244bead5f09a86e91bb2365cbcd1122ac68261e5647942485df
linux-arm64	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20241016/openocd-esp32-linux-arm64-0.12.0-esp32-20241016.tar.gz SHA256: 8f8daf5bd22ec5d2fa9257b0862ec33da18ee677e023fb9a9eb17f74ce208c76
linux-armel	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20241016/openocd-esp32-linux-armel-0.12.0-esp32-20241016.tar.gz SHA256: bc9c020ecf20e2000f76cffa44305fd5bc44d2e688ea78cce423399d33f19767
linux-armhf	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20241016/openocd-esp32-linux-armhf-0.12.0-esp32-20241016.tar.gz SHA256: 2cd6436465333e998000b1c311d301b295d8ebaa3fb1c9aa9d4393539dc16ec6
macos	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20241016/openocd-esp32-macos-0.12.0-esp32-20241016.tar.gz SHA256: 02a2dffe801a2d005fa9e614d80ff8173395b2cb0b5d3118d0229d094a9946a7
macos-arm64	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20241016/openocd-esp32-macos-arm64-0.12.0-esp32-20241016.tar.gz SHA256: c382f9e884d6565cb6089bff5f200f4810994667d885f062c3d3c5625a0fa9d6
win32	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20241016/openocd-esp32-win32-0.12.0-esp32-20241016.zip SHA256: 3b5d615e0a72cc771a45dd469031312d5881c01d7b6bc9edb29b8b6bda8c2e90
win64	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20241016/openocd-esp32-win64-0.12.0-esp32-20241016.zip SHA256: 5e7b2fd1947d3a8625f6a11db7a2340cf2f41ff4c61284c022c7d7c32b18780a

ninja Ninja build system

On Linux and macOS, it is recommended to install ninja using the OS-specific package manager (like apt, yum, brew, etc.). However, for convenience it is possible to install ninja using idf_tools.py along with the other tools.

License: Apache-2.0

More info: https://github.com/ninja-build/ninja

Platform	Required	Download
linux-amd64	optional	https://github.com/ninja-build/ninja/releases/download/v1.12.1/ninja-linux.zip SHA256: 6f98805688d19672bd699fbbfa2c2cf0fc054ac3df1f0e6a47664d963d530255
macos	optional	https://github.com/ninja-build/ninja/releases/download/v1.12.1/ninja-mac.zip SHA256: 89a287444b5b3e98f88a945afa50ce937b8ffd1dcc59c555ad9b1baf855298c9
macos-arm64	optional	https://github.com/ninja-build/ninja/releases/download/v1.12.1/ninja-mac.zip SHA256: 89a287444b5b3e98f88a945afa50ce937b8ffd1dcc59c555ad9b1baf855298c9
win64	required	https://github.com/ninja-build/ninja/releases/download/v1.12.1/ninja-win.zip SHA256: f550fec705b6d6ff58f2db3c374c2277a37691678d6aba463adcbb129108467a

idf-exe IDF wrapper tool for Windows

License: Apache-2.0

More info: https://github.com/espressif/idf_py_exe_tool

Platform	Required	Download
win32	required	https://github.com/espressif/idf_py_exe_tool/releases/download/v1.0.3/idf-exe-v1.0.3.zip SHA256: 7c81ef534c562354a5402ab6b90a6eb1cc8473a9f4a7b7a7f93ebbd23b4a2755
win64	required	https://github.com/espressif/idf_py_exe_tool/releases/download/v1.0.3/idf-exe-v1.0.3.zip SHA256: 7c81ef534c562354a5402ab6b90a6eb1cc8473a9f4a7b7a7f93ebbd23b4a2755

ccache Ccache (compiler cache)

License: GPL-3.0-or-later

More info: https://github.com/ccache/ccache

Platform	Required	Download
win64	required	https://github.com/ccache/ccache/releases/download/v4.10.2/ccache-4.10.2-windows-x86_64.zip SHA256: 6252f081876a9a9f700fae13a5aec5d0d486b28261d7f1f72ac11c7ad9df4da9

dfu-util dfu-util (Device Firmware Upgrade Utilities)

License: GPL-2.0-only

More info: http://dfu-util.sourceforge.net/

Platform	Required	Download
win64	required	https://dl.espressif.com/dl/dfu-util-0.11-win64.zip SHA256: 652eb94cb1c074c6dbead9e47adb628922aeb198a4d440a346ab32e7a0e9bf64

esp-rom-elfs ESP ROM ELFs

License: Apache-2.0

More info: https://github.com/espressif/esp-rom-elfs

Platform	Required	Download
any	required	https://github.com/espressif/esp-rom-elfs/releases/download/20240305/esp-rom-elfs-20240305.tar.gz SHA256: a26609b415710f0163d785850c769752717004059c129c472e9a0cbd54e0422c

qemu-xtensa QEMU for Xtensa

Some ESP-specific instructions for running QEMU for Xtensa chips are here: https://github.com/espressif/esp-toolchain-docs/blob/main/qemu/esp32/README.md

License: GPL-2.0-only

More info: https://github.com/espressif/qemu

Platform	Required	Download
linux-amd64	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-xtensa-softmmu-esp_develop_9.0.0_20240606-x86_64-linux-gnu.tar.xz SHA256: 071d117c44a6e9a1bc8664ab63b592d3e17ceb779119dcb46c59571a4a7a88c9
linux-arm64	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-xtensa-softmmu-esp_develop_9.0.0_20240606-aarch64-linux-gnu.tar.xz SHA256: 43552f32b303a6820d0d9551903e54fc221aca98ccbd04e5cbccbca881548008
macos	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-xtensa-softmmu-esp_develop_9.0.0_20240606-x86_64-apple-darwin.tar.xz SHA256: 0096734280ce04f558cd9bd72f35db39667f80d44309a35565f2f8c02d1f9cc3
macos-arm64	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-xtensa-softmmu-esp_develop_9.0.0_20240606-aarch64-apple-darwin.tar.xz SHA256: fb4ca6be7b1a4dbcf153879cf0582300f974371def0826c0c5b728f12812ad08
win64	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-xtensa-softmmu-esp_develop_9.0.0_20240606-x86_64-w64-mingw32.tar.xz SHA256: 281659f7a1d49761ac6f54d0aeb14366cb93c002f21948b847a0e15c0b8f5425

qemu-riscv32 QEMU for RISC-V

Some ESP-specific instructions for running QEMU for RISC-V chips are here: https://github.com/espressif/esp-toolchain-docs/blob/main/qemu/esp32c3/README.md

License: GPL-2.0-only

More info: https://github.com/espressif/qemu

Platform	Required	Download
linux-amd64	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-riscv32-softmmu-esp_develop_9.0.0_20240606-x86_64-linux-gnu.tar.xz SHA256: 47120e826cfec7180db8cb611a7a4aed2e9b2191c2a739194f8ce085e63cdd8d
linux-arm64	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-riscv32-softmmu-esp_develop_9.0.0_20240606-aarch64-linux-gnu.tar.xz SHA256: 3b6221a8b1881d2c9b9fa0b0bf8d7065c84153d2a54e429307bde9feae235c27
macos	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-riscv32-softmmu-esp_develop_9.0.0_20240606-x86_64-apple-darwin.tar.xz SHA256: 3afa55d5abea52ccf18d0bc41fe819d568bd4ee1582989b1ee9b1ee4a609a31e
macos-arm64	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-riscv32-softmmu-esp_develop_9.0.0_20240606-aarch64-apple-darwin.tar.xz SHA256: 69ba5154594fb2922d5490a49ea6b4925c024c6c37f875b42f9885f513e0bcdd
win64	optional	https://github.com/espressif/qemu/releases/download/esp-develop-9.0.0-20240606/qemu-riscv32-softmmu-esp_develop_9.0.0_20240606-x86_64-w64-mingw32.tar.xz SHA256: f49bb5c8f4d6e2cfbf7eeec21eb8ef190a57307778705bc689536ac13bde511c

4.23 ESP32-P4 中的单元测试

ESP-IDF 提供以下方法测试软件。

- 一种是基于目标的测试，该测试使用运行在 esp32p4 上的中央单元测试应用程序。这些测试使用的是基于 Unity 的单元测试框架。通过把测试用例放在组件的 `test` 子目录，可以将其集成到 ESP-IDF 组件中。本文档主要介绍这种基于目标的测试方法。
- 另一种是基于 Linux 主机的单元测试，其中所有硬件行为都通过 Mock 组件进行模拟。此测试方法目前仍在开发中，暂且只有一小部分 ESP-IDF 组件支持 Mock，具体请参考*基于 Linux 主机的单元测试*。

4.23.1 添加常规测试用例

单元测试被添加在相应组件的 `test` 子目录中，测试用例写在 C 文件中，一个 C 文件可以包含多个测试用例。测试文件的名字要以“test”开头。

测试文件需要包含 `unity.h` 头文件，此外还需要包含待测试 C 模块需要的头文件。

测试用例需要通过 C 文件中特定的函数来添加，如下所示：

```
TEST_CASE("test name", "[module name]")
{
        // 在这里添加测试用例
}
```

- 第一个参数是此测试的描述性名称。
- 第二个参数是用方括号括起来的标识符。标识符用来对相关测试或具有特定属性的测试进行分组。

备注： 没有必要在每个测试用例中使用 `UNITY_BEGIN()` 和 `UNITY_END()` 来声明主函数的区域，`unity_platform.c` 会自动调用 `UNITY_BEGIN()`，然后运行测试用例，最后调用 `UNITY_END()`。

`test` 子目录应包含*组件 CMakeLists.txt*，因为他们本身就是一种组件（即测试组件）。ESP-IDF 使用了 Unity 测试框架，位于 `unity` 组件里。因此，每个测试组件都需要通过 `REQUIRES` 参数将 `unity` 组件设为依赖项。通常，组件需要*手动指定待编译的源文件*，但是，对于测试组件来说，这个要求被放宽为仅建议将参数 `SRC_DIRS` 用于 `idf_component_register`。

总的来说，test 子目录下最小的 CMakeLists.txt 文件可能如下所示:

```
idf_component_register(SRC_DIRS "."
                       INCLUDE_DIRS "."
                       REQUIRES unity)
```

更多关于如何在 Unity 下编写测试用例的信息，请查阅 http://www.throwtheswitch.org/unity。

4.23.2 添加多设备测试用例

常规测试用例会在一个在试设备 (Device Under Test，DUT) 上执行。但是，由于要求互相通信的组件（比如 GPIO、SPI）需要与其他设备进行通信，因此不能使用常规测试用例进行测试。多设备测试用例包括写入多个测试函数，并在多个 DUT 运行测试。

以下是一个多设备测试用例:

```
void gpio_master_test()
{
    gpio_config_t slave_config = {
            .pin_bit_mask = 1 << MASTER_GPIO_PIN,
            .mode = GPIO_MODE_INPUT,
    };
    gpio_config(&slave_config);
    unity_wait_for_signal("output high level");
    TEST_ASSERT(gpio_get_level(MASTER_GPIO_PIN) == 1);
}

void gpio_slave_test()
{
    gpio_config_t master_config = {
            .pin_bit_mask = 1 << SLAVE_GPIO_PIN,
            .mode = GPIO_MODE_OUTPUT,
    };
    gpio_config(&master_config);
    gpio_set_level(SLAVE_GPIO_PIN, 1);
    unity_send_signal("output high level");
}

TEST_CASE_MULTIPLE_DEVICES("gpio multiple devices test example", "[driver]", gpio_
↪master_test, gpio_slave_test);
```

宏 TEST_CASE_MULTIPLE_DEVICES 用来声明多设备测试用例。

- 第一个参数指定测试用例的名字。
- 第二个参数是测试用例的描述。
- 从第三个参数开始，可以指定最多 5 个测试函数，每个函数都是单独运行在一个 DUT 上的测试入口点。

在不同的 DUT 上运行的测试用例需要相互之间进行同步。可以通过 unity_wait_for_signal 和 unity_send_signal 这两个函数使用 UART 进行同步操作。上例的场景中，slave 应该在 master 设置好 GPIO 电平后再去读取 GPIO 电平，DUT 的 UART 终端会打印提示信息，并要求用户进行交互。

DUT1 (master) 终端:

```
Waiting for signal: [output high level]!
Please press "Enter" key once any board send this signal.
```

DUT2 (slave) 终端:

```
Send signal: [output high level]!
```

一旦 DUT2 发送了该信号，你需要在 DUT1 的终端按回车键，然后 DUT1 会从 unity_wait_for_signal 函数中解除阻塞，并开始更改 GPIO 的电平。

4.23.3 添加多阶段测试用例

常规的测试用例无需重启就会结束（或者仅需要检查是否发生了重启），可有些时候我们想在某些特定类型的重启事件后运行指定的测试代码。例如，在深度睡眠唤醒后检查复位的原因是否正确。首先我们需要触发深度睡眠复位事件，然后检查复位的原因。为了实现这一点，可以通过定义多阶段测试用例来将这些测试函数组合在一起:

```
static void trigger_deepsleep(void)
{
    esp_sleep_enable_timer_wakeup(2000);
    esp_deep_sleep_start();
}

void check_deepsleep_reset_reason()
{
    soc_reset_reason_t reason = esp_rom_get_reset_reason(0);
    TEST_ASSERT(reason == RESET_REASON_CORE_DEEP_SLEEP);
}

TEST_CASE_MULTIPLE_STAGES("reset reason check for deepsleep", "[esp32p4]", trigger_
↪deepsleep, check_deepsleep_reset_reason);
```

多阶段测试用例向用户呈现了一组测试函数，它需要用户进行交互（选择用例并选择不同的阶段）来运行。

4.23.4 应用于不同芯片的单元测试

某些测试（尤其与硬件相关的）不支持在所有的芯片上执行。请参照本节，让你的单元测试只在其中一部分芯片上执行。

1. 使用宏 `!(TEMPORARY_)DISABLED_FOR_TARGETS()` 包装你的测试代码，并将其放于原始的测试文件中，或将代码分成按功能分组的文件。但请确保所有这些文件都会由编译器处理。例:

   ```
   #if !TEMPORARY_DISABLED_FOR_TARGETS(ESP32, ESP8266)
   TEST_CASE("a test that is not ready for esp32 and esp8266 yet", "[]")
   {
   }
   #endif //!TEMPORARY_DISABLED_FOR_TARGETS(ESP32, ESP8266)
   ```

如果需要将其中某个测试在特定芯片上编译，只需要修改禁止的芯片列表。推荐使用一些能在 `soc_caps.h` 中被清楚描述的通用概念来禁止某些单元测试。如果已经进行上述操作，但一些测试在芯片中的调试暂未通过，请同时使用上述两种方法，当调试完成后再移除 `!(TEMPORARY_)DISABLED_FOR_TARGETS()`。例:

```
#if SOC_SDIO_SLAVE_SUPPORTED
#if !TEMPORARY_DISABLED_FOR_TARGETS(ESP64)
TEST_CASE("a sdio slave tests that is not ready for esp64 yet", "[sdio_slave]")
{
    //available for esp32 now, and will be available for esp64 in the future
}
#endif //!TEMPORARY_DISABLED_FOR_TARGETS(ESP64)
#endif //SOC_SDIO_SLAVE_SUPPORTED
```

2. 对于某些你确定不会支持的测试（例如，芯片根本没有该外设），使用 `DISABLED_FOR_TARGETS` 来禁止该测试；对于其他只是临时性需要关闭的（例如，没有 runner 资源等），使用 `TEMPORARY_DISABLED_FOR_TARGETS` 来暂时关闭该测试。

一些禁用目标芯片测试用例的旧方法，由于它们具有明显的缺陷，已经被废弃，请勿继续使用:

- 请勿将测试代码放在 `test/target` 目录下并用 CMakeLists.txt 来选择其中一个进行编译。这是因为测试代码比实现代码更容易被复用。如果你将一些代码放在 `test/esp32` 目录下来避免 esp32s2 芯片执行它，一旦你需要在新的芯片（比如 esp32s3）中启用该测试，这种结构很难保持代码整洁。

- 请勿继续使用 CONFIG_IDF_TARGET_xxx 宏来禁用测试。这种方法会让被禁用的测试项目难以追踪和重新打开。并且，相比于白名单式的 #if CONFIG_IDF_TARGET_xxx，黑名单式的 #if !disabled 不会导致在新芯片引入时这些测试被自动禁用。但对于测试实现，仍可使用 #if CONFIG_IDF_TARGET_xxx 给不同芯片版本选择实现代码。测试项目和测试实现区分如下：
  - 测试项目：那些会在一些芯片上执行，而在另外一些上跳过的项目，例如：
    有三个测试项目 SD 1-bit、SD 4-bit 和 SDSPI。对于不支持 SD Host 外设的 ESP32-S2 芯片，只有 SDSPI 一个项目需要被执行。
  - 测试实现：一些始终会发生的代码，但采取的实现方式不同。例如：
    ESP8266 芯片没有 SDIO_PKT_LEN 寄存器。如果在测试过程中需要从 slave 设备的数据长度，你可以用不同方式读取的 #if CONFIG_IDF_TARGET_ 宏来保护不同的实现代码。
    但请注意避免使用 #else 宏。这样当新芯片被引入时，测试就会在编译阶段失败，提示维护者去显示选择一个正确的测试实现。

4.23.5 编译单元测试程序

按照 esp-idf 顶层目录的 README 文件中的说明进行操作，请确保 IDF_PATH 环境变量已经被设置指向了 esp-idf 的顶层目录。

切换到 tools/unit-test-app 目录下进行配置和编译：

- idf.py menuconfig - 配置单元测试程序。
- idf.py -T all build - 编译单元测试程序，测试每个组件 test 子目录下的用例。
- idf.py -T "xxx yyy" build - 编译单元测试程序，对以空格分隔的特定组件进行测试（如 idf.py -T heap build - 仅对 heap 组件目录下的单元测试程序进行编译）。
- idf.py -T all -E "xxx yyy" build - 编译单元测试程序，测试除指定组件之外的所有组件（例如 idf.py -T all -E "ulp mbedtls" build - 编译所有的单元测试，不包括 ulp 和 mbedtls 组件。）。

备注： 由于 Windows 命令提示符固有限制，需使用以下语法来编译多个组件的单元测试程序：idf.py -T xxx -T yyy build 或者在 PowerShell 中使用 idf.py -T \`"xxx yyy\`" build，在 Windows 命令提示符中使用 idf.py -T \^"ssd1306 hts221\^" build。

当编译完成时，它会打印出烧写芯片的指令。你只需要运行 idf.py flash 即可烧写所有编译输出的文件。

你还可以运行 idf.py -T all flash 或者 idf.py -T xxx flash 来编译并烧写，所有需要的文件都会在烧写之前自动重新编译。

使用 menuconfig 可以设置烧写测试程序所使用的串口。更多信息，见 tools/unit-test-app/README.md。

4.23.6 运行单元测试

备注： 我们还提供基于 pytest 的框架 pytest-embedded，以便更方便、高效地运行单元测试。如需在 CI 中运行测试或连续运行多个测试，不妨尝试这一框架。了解更多信息，请查看 pytest-embedded 文档 和*ESP-IDF pytest 指南*。

烧写完成后重启 ESP32-P4，它将启动单元测试程序。

当单元测试应用程序空闲时，输入回车键，它会打印出测试菜单，其中包含所有的测试项目：

```
Here is the test menu, pick your combo:
(1)     "esp_ota_begin() verifies arguments" [ota]
(2)     "esp_ota_get_next_update_partition logic" [ota]
(3)     "Verify bootloader image in flash" [bootloader_support]
(4)     "Verify unit test app image" [bootloader_support]
(5)     "can use new and delete" [cxx]
```

(下页继续)

(续上页)

```
(6)     "can call virtual functions" [cxx]
(7)     "can use static initializers for non-POD types" [cxx]
(8)     "can use std::vector" [cxx]
(9)     "static initialization guards work as expected" [cxx]
(10)    "global initializers run in the correct order" [cxx]
(11)    "before scheduler has started, static initializers work correctly" [cxx]
(12)    "adc2 work with wifi" [adc]
(13)    "gpio master/slave test example" [ignore][misc][test_env=UT_T2_1][multi_
↪device]
        (1)     "gpio_master_test"
        (2)     "gpio_slave_test"
(14)    "SPI Master clockdiv calculation routines" [spi]
(15)    "SPI Master test" [spi][ignore]
(16)    "SPI Master test, interaction of multiple devs" [spi][ignore]
(17)    "SPI Master no response when switch from host1 (SPI2) to host2 (SPI3)"␣
↪[spi]
(18)    "SPI Master DMA test, TX and RX in different regions" [spi]
(19)    "SPI Master DMA test: length, start, not aligned" [spi]
(20)    "reset reason check for deepsleep" [esp32p4][test_env=UT_T2_1][multi_stage]
        (1)     "trigger_deepsleep"
        (2)     "check_deepsleep_reset_reason"
```

常规测试用例会打印用例名字和描述，主从测试用例还会打印子菜单（已注册的测试函数的名字）。

可以输入以下任意一项来运行测试用例：

- 引号中写入测试用例的名字，运行单个测试用例。
- 测试用例的序号，运行单个测试用例。
- 方括号中的模块名字，运行指定模块所有的测试用例。
- 星号，运行所有测试用例。

`[multi_device]` 和 `[multi_stage]` 标签告诉测试运行者该用例是多设备测试还是多阶段测试。这些标签由 `TEST_CASE_MULTIPLE_STAGES` 和 `TEST_CASE_MULTIPLE_DEVICES` 宏自动生成。

一旦选择了多设备测试用例，它会打印一个子菜单：

```
Running gpio master/slave test example...
gpio master/slave test example
        (1)     "gpio_master_test"
        (2)     "gpio_slave_test"
```

你需要输入数字以选择在 DUT 上运行的测试。

与多设备测试用例相似，多阶段测试用例也会打印子菜单：

```
Running reset reason check for deepsleep...
reset reason check for deepsleep
        (1)     "trigger_deepsleep"
        (2)     "check_deepsleep_reset_reason"
```

第一次执行此用例时，输入 1 来运行第一阶段（触发深度睡眠）。在重启 DUT 并再次选择运行此用例后，输入 2 来运行第二阶段。只有在最后一个阶段通过并且之前所有的阶段都成功触发了复位的情况下，该测试才算通过。

4.23.7 带缓存补偿定时器的定时代码

存储在外部存储器（如 SPI flash 和 SPI RAM）中的指令和数据是通过 CPU 的统一指令和数据缓存来访问的。当代码或数据在缓存中时，访问速度会非常快（即缓存命中）。

然而，如果指令或数据不在缓存中，则需要从外部存储器中获取（即缓存缺失）。访问外部存储器的速度明显较慢，因为 CPU 在等待从外部存储器获取指令或数据时会陷入停滞，从而导致整体代码执行速度会依据缓存命中或缓存缺失的次数而变化。

在不同的编译中，代码和数据的位置可能会有所不同，一些可能会更有利于缓存访问（即最大限度地减少缓存缺失）。理论上，这会影响执行速度，但这些因素通常无关紧要，因为它们的影响会在设备的运行过程中“平均化”。

然而，高速缓存对执行速度的影响可能与基准测试场景（尤其是微基准测试）有关。每次运行时间和构建时的测量时间可能会有所差异，减少差异的方法之一是将代码和数据分别放在指令或数据 RAM (IRAM/DRAM) 中。CPU 可以直接访问 IRAM 和 DRAM，从而消除了高速缓存的影响因素。然而，由于 IRAM 和 DRAM 容量有限，该方法并不总是可行。

缓存补偿定时器是上述方法的替代方法，该计时器使用处理器的内部事件计数器来确定在发生高速缓存未命中时等待代码/数据所花费的时间，然后从记录的实时时间中减去该时间。

```
// Start the timer
ccomp_timer_start();

// Function to time
func_code_to_time();

// Stop the timer, and return the elapsed time in microseconds relative to
// ccomp_timer_start
int64_t t = ccomp_timer_stop();
```

缓存补偿定时器的限制之一是基准功能必须固定在一个内核上。这是由于每个内核都有自己的事件计数器，这些事件计数器彼此独立。例如，如果在一个内核上调用 `ccomp_timer_start`，使调度器进入睡眠状态，唤醒并在在另一个内核上重新调度，那么对应的 `ccomp_timer_stop` 将无效。

4.23.8 Mocks

备注： 目前，只有一些特定的组件在 Linux 主机上运行时才能 Mock。未来我们计划，无论是在 Linux 主机上运行还是在目标芯片 ESP32-P4 上运行，ESP-IDF 所有重要的组件都可以实现 Mock。

嵌入式系统中单元测试的最大问题之一是对硬件依赖性极强。直接在 ESP32-P4 上运行单元测试对于上层组件来说存在极大的困难，原因如下：

- 受下层组件和/或硬件设置的影响，测试可靠性降低。
- 由于下层组件和/或硬件设置的限制，测试边缘案例的难度提高。
- 由于数量庞大的依赖关系影响了行为，识别根本原因的难度提高。

当测试一个特定的组件（即被测组件）时，通过软件进行 Mock 能让所有被测组件的依赖在软件中被完全替换（即 Mock）。为了实现该功能，ESP-IDF 集成了 CMock 的 Mock 框架作为组件。通过在 ESP-IDF 的构建系统中添加一些 CMake 函数，可以方便地 Mock 整个（或部分）ESP-IDF 组件。

理想情况下，被测组件所依赖的所有组件都应该被 Mock，从而让测试环境完全控制与被测组件之间的所有交互。然而，如果 Mock 所有的组件过于复杂或冗长（例如需要模拟过多的函数调用），以下做法可能会有帮助：

- 在测试代码中包含更多“真正”（非模拟）代码。这样做可能有效，但同时也会增加对“真正”代码行为的依赖。此外，一旦测试失败，很难判断失败原因是因为实际测试代码还是“真正”的 ESP-IDF 代码。
- 重新评估被测代码的设计，尝试将被测代码划分为更易于管理的组件来减少其依赖性。这可能看起来很麻烦，但众所周知，单元测试经常暴露软件设计的弱点。修复设计上的弱点不仅在短期内有助于进行单元测试，而且还有助于长期的代码维护。

请参考 cmock/CMock/docs/CMock_Summary.md 了解 CMock 工作原理以及如何创建和使用 Mock。

要求

目前 Mock 只支持基于 Linux 主机的单元测试。生成 Mock 需要满足如下要求：

- 已安装 ESP-IDF 及使用 ESP-IDF 的所有依赖项
- 满足系统软件包需求 (libbsd、libbsd-dev)
- Linux 或 macOS 和 GCC 编译器已更新至足够新的版本
- 应用程序所依赖的所有组件必须受 Linux 目标（Linux/POSIX 模拟器）支持，或可进行模拟

对于在 Linux 目标上运行的应用程序，需要在应用程序根目录的 CMakeLists.txt 文件中，设置 COMPONENTS 变量为 main，具体操作如下：

```
set(COMPONENTS main)
```

为方便起见，应用程序会在构建过程中，自动包含 ESP-IDF 的所有组件，执行上述代码则可以防止此类情况。

对组件进行 Mock

如果 ESP-IDF 中已对组件进行 Mock（也称为 *组件模拟*），那么只要满足要求，该版本即可立即投入使用。已进行 Mock 的组件列表，可参考*组件 Linux/Mock 支持情况概述*。具体组件模拟的使用方法，请参考*修改单元测试文件*。

如果 ESP-IDF 尚未提供任何组件模拟，则需要创建组件的 Mock 版本，以特定方式覆盖组件。覆盖组件时，需要创建一个与原始组件名称完全相同的组件，让构建系统先发现原始组件，再发现这个具有相同名称的新组件。具体可参考*同名组件*。

在组件模拟中需要指定如下部分：

- 头文件，头文件中提供了需要生成模拟的函数
- 上述头文件的路径
- 模拟组件的依赖（如果头文件中包含了其他组件的文件，那么这点非常必要）

以上这些部分都需要使用 ESP-IDF 构建系统函数 idf_component_mock 指定。你可以使用 ESP-IDF 构建系统函数 idf_component_get_property，并加上标签 COMPONENT_OVERRIDEN_DIR 来访问原始组件的组件目录，然后使用 idf_component_mock 注册模拟组件。

```
idf_component_get_property(original_component_dir <original-component-name>␣
↪COMPONENT_OVERRIDEN_DIR)
...
idf_component_mock(INCLUDE_DIRS "${original_component_dir}/include"
    REQUIRES freertos
    MOCK_HEADER_FILES ${original_component_dir}/include/header_containing_
↪functions_to_mock.h)
```

组件模拟还需要一个单独的 mock 目录，里面包含一个 mock_config.yaml 文件用于配置 CMock。以下是一份简单的 mock_config.yaml 文件：

```
:cmock:
  :plugins:
    - expect
    - expect_any_args
```

更多关于 CMock yaml 类型配置文件的详细信息，请查看 cmock/CMock/docs/CMock_Summary.md。

请注意，组件模拟不一定要对原始组件进行整体模拟。只要组件模拟满足测试项目的依赖以及其他代码对原始组件的依赖，部分模拟就足够了。事实上，ESP-IDF 中 tools/mocks 中的大多数组件模拟都只是部分地模拟了原始组件。

可在 ESP-IDF 目录的 tools/mocks 下找到组件模拟的示例。有关如何 *覆盖 ESP-IDF 组件*，可查看*同名组件*。

- NVS 页面类的单元测试 。
- esp_event 的单元测试 。
- mqtt 的单元测试 。

修改单元测试文件

单元测试需要通知 cmake 构建系统对依赖的组件进行模拟（即用模拟组件来覆盖原始组件）。这可以通过将组件模拟放到项目的 `components` 目录，或者在项目的根目录 `CMakeLists.txt` 文件中使用以下代码来添加模拟组件的目录来实现：

```
list(APPEND EXTRA_COMPONENT_DIRS "<mock_component_dir>")
```

这两种方法都会让组件模拟覆盖 ESP-IDF 中的现有组件。如果你使用的是 ESP-IDF 提供的组件模拟，则第二个方法更加方便。

可参考 `esp_event` 基于主机的单元测试及其 esp_event/host_test/esp_event_unit_test/CMakeLists.txt 作为组件模拟的示例。

4.24 在主机上运行 ESP-IDF 应用程序

备注： 在主机上运行 ESP-IDF 应用程序的功能仍处于试验阶段，无法保证 API 的稳定性。欢迎用户通过 ESP-IDF GitHub 仓库 或 ESP32 论坛 提供反馈意见，助力未来 ESP-IDF 基于主机的应用程序设计。

本文档概述了在 Linux 上运行 ESP-IDF 应用程序的方法，并介绍了可以在 Linux 上运行的常见 ESP-IDF 应用程序类型。

4.24.1 介绍

ESP-IDF 应用程序通常在主机上进行构建（交叉编译），然后上传（即烧录）到 ESP 芯片上运行，并由主机通过 UART/USB 端口监控。然而，在 ESP 芯片上运行 ESP-IDF 应用程序在各种开发/使用/测试场景下可能存在限制。

因此，ESP-IDF 支持在同一台 Linux 主机上构建和运行其应用程序。在主机上运行 ESP-IDF 应用程序具有以下几个优点：

- 无需上传到目标芯片。
- 应用程序在主机上的运行速度比在 ESP 芯片上快。
- 除主机本身外，无特定硬件要求。
- 更易进行软件测试的自动化和设置。
- 提供大量用于代码和运行分析的工具，如 Valgrind。

许多 ESP-IDF 组件依赖支持特定芯片的硬件，因此在主机上运行应用程序时，必须对这些硬件依赖文件进行模拟或仿真。目前，ESP-IDF 支持以下模拟和仿真方法：

1. 使用 FreeRTOS POSIX/Linux 模拟器 可以模拟 FreeRTOS 调度。在此模拟的基础上，在主机上运行应用程序时也会模拟或使用其他 API。
2. 使用 CMock 可以模拟所有依赖文件，并在完全独立的情况下运行代码。

原则上，这两种方法（POSIX/Linux 模拟器和使用 CMock 模拟）可以混用，但此功能在 ESP-IDF 中尚未实现。注意，尽管名称中包含 POSIX/Linux，但目前的 FreeRTOS POSIX/Linux 模拟器也支持在 macOS 系统中运行。在主机上运行 ESP-IDF 应用程序通常用于测试，但模拟环境和模拟依赖文件并不能完全代表目标设备。因此，仍然需要在目标设备上测试，此时测试的侧重点通常在集成和系统测试上。

备注： 在主机上运行应用程序的另一方法是使用 QEMU 模拟器，但 ESP-IDF 的 QEMU 模拟器仍在开发中，尚未发布相关文档。

基于 CMock 的模拟

该方法使用 CMock 框架解决缺少硬件和软件依赖文件的问题。在主机上运行基于 CMock 的应用程序具备一大优势：在主机上运行的应用程序通常只编译必要代码，即模拟了最主要代码的依赖文件，而非编译整个系统。有关 Mocks 的总体介绍及其在 ESP-IDF 的配置和使用，请参阅*Mocks*。

POSIX/Linux 模拟器的模拟

ESP-IDF 已支持使用 FreeRTOS POSIX/Linux 模拟器 预览应用程序在目标芯片上的运行效果。使用该模拟器可以在主机上运行 ESP-IDF 组件，并使这类组件可用于在主机上运行的 ESP-IDF 应用程序。目前，只有一部分组件可以在 Linux 上构建。此外，各组件移植到 Linux 上后，其功能可能也会受到限制，或与在芯片目标上构建该组件的功能有所不同。有关所需组件在 Linux 上是否受支持的更多信息，请参阅*组件 Linux/Mock 支持情况概述*。

4.24.2 使用模拟器的前提

- 已安装 ESP-IDF 及使用 ESP-IDF 的所有依赖项
- 满足系统软件包需求 (`libbsd`、`libbsd-dev`)
- Linux 或 macOS 和 GCC 编译器已更新至足够新的版本
- 应用程序所依赖的所有组件必须受 Linux 目标（Linux/POSIX 模拟器）支持，或可进行模拟

对于在 Linux 目标上运行的应用程序，需要在应用程序根目录的 CMakeLists.txt 文件中，设置 `COMPONENTS` 变量为 `main`，具体操作如下：

```
set(COMPONENTS main)
```

为方便起见，应用程序会在构建过程中，自动包含 ESP-IDF 的所有组件，执行上述代码则可以防止此类情况。

如果使用了任意模拟器，则应设置变量 `Ruby`。

4.24.3 构建和运行

要在 Linux 上构建并运行应用程序，需要将目标芯片设置为 `linux`：

```
idf.py --preview set-target linux
idf.py build
idf.py monitor
```

4.24.4 组件 Linux/Mock 支持情况概述

注意，下表中的“是”并不代表完全实现或模拟，它也可以表示功能的部分实现或模拟。实现或模拟通常只进行到可以提供足够功能、可以构建和运行测试应用程序的程度。

组件	模拟	仿真
cmock	否	是
driver	是	否
esp_common	否	是
esp_event	是	是
esp_http_client	否	是
esp_http_server	否	是
esp_https_server	否	是
esp_hw_support	是	是
esp_netif	是	是

下页继续

表 8 – 续上页

组件	模拟	仿真
esp_netif_stack	否	是
esp_partition	是	否
esp_rom	否	是
esp_system	否	是
esp_timer	是	否
esp_tls	是	否
fatfs	否	是
freertos	是	是
hal	否	是
heap	否	是
http_parser	是	是
json	否	是
linux	否	是
log	否	是
lwip	是	是
mbedtls	否	是
mqtt	否	是
nvs_flash	否	是
partition_table	否	是
protobuf-c	否	是
pthread	否	是
soc	否	是
spiffs	否	是
spi_flash	是	否
tcp_transport	是	否
unity	否	是

4.25 USB 串行/JTAG 控制器控制台

ESP 芯片通常使用 UART 实现串口，并可以通过外部 USB-UART 桥接芯片，连接到主机/PC 上的串口控制台仿真器。然而，在具有 USB 串行/JTAG 控制器的 ESP 芯片上，控制器的 CDC-ACM 部分即可实现串口，该串口直接连接到主机/PC，不需要外部 USB-UART 桥接芯片。

拥有 USB 串行/JTAG 控制器时，ESP32-P4 支持以下功能：

- 双向串行控制台，可与*IDF 监视器* 或其他串行监视器一起使用。
- 使用 `esptool.py` 和 `idf.py flash` 烧录。
- 使用 OpenOCD 等工具进行 JTAG 调试，同时进行串行操作。

备注： USB 串行/JTAG 控制器是一个 USB 设备，其功能固定，完全由硬件实现。也就是说，即便重新配置，它也只能执行串口和 JTAG 调试功能。这不同于某些 ESP 芯片中的 USB OTG 控制器，后者经配置，可以执行多种 USB 设备功能。

4.25.1 硬件要求

将 ESP32-P4 连接到 USB 端口，连接方式如下：

GPIO	USB
未更新!	D+（绿）
未更新!	D-（白）
GND	GND（黑）
5V（或由外部提供）	+5V（红）

部分开发板可能会为 USB 串行/JTAG 控制器提供 USB 连接器，此时无需额外连接。

4.25.2 软件配置

在*CONFIG_ESP_CONSOLE_UART* 选项中选择 `CONFIG_ESP_CONSOLE_USB_SERIAL_JTAG`，可以将 USB 串行/JTAG 控制器用作串口。选择该选项后，即可正常构建和烧录项目。

另外，也可以通过 `usb_serial_jtag` 端口访问输出，但请确保在*CONFIG_ESP_CONSOLE_SECONDARY* 选项中选择了 `CONFIG_ESP_CONSOLE_SECONDARY_USB_SERIAL_JTAG`。

警告： 除输出外，如果还想用控制台输入或使用交互式解释器 (REPL)，请选择 `CONFIG_ESP_CONSOLE_USB_SERIAL_JTAG` 选项。

4.25.3 上传应用程序

USB 串行/JTAG 控制器可以自动将 ESP32-P4 置于下载模式。用户只需正常烧录芯片，但需要在系统上指定 USB 串行/JTAG 控制器端口：`idf.py flash -p PORT`，`PORT` 是正确的端口名称。

备注： USB 串行/JTAG 控制器的串行端口通常：

- 在 Linux 系统上显示为 `/dev/ttyACM*`
- 在 MAC 系统上显示为 `/dev/cu*`
- 在 Windows 设备管理器上显示为 `COM*` 端口

4.25.4 限制

USB 串行/JTAG 控制台功能存在一些限制，这些限制的影响程度各不相同，具体取决于开发中的应用程序类型，以及开发的工作流程。

重新配置 USB 管脚

如果在意外情况下，应用程序重新配置了 USB 外设管脚，或禁用了 USB 串行/JTAG 控制器，设备会从系统消失。应用程序中的问题修复后，需要将未更新！拉低，并重置芯片，手动将 ESP32-P4 置于下载模式。

如果应用程序进入 Deep-sleep 模式，USB 串行/JTAG 设备会从系统中消失。

数据缓冲

对于从 ESP32-P4 发送到 PC 终端的数据（例如 stdout、日志），ESP32-P4 会先写入一个小型内部缓冲区。如果该缓冲区达到最大容量（例如没有连接 PC 终端），ESP32-P4 会进行一次等待，时长约 50 ms。在此期间，PC 终端可以请求数据，但可能会导致应用程序出现非常短暂的暂停。

对于从 PC 终端发送到 ESP32-P4 的数据（如控制台命令），许多 PC 终端会等待 ESP32-P4 接收字节后，再允许发送更多数据。这与使用 USB 转串口 (URAT) 桥接芯片不同，URAT 桥接芯片始终接收字节，并将其发送到（可能未在监听）的 ESP32-P4。

备注： 在极少数情况下，从 ESP32-P4 发送到主机的数据可能会“阻塞”在主机内存中。继续发送数据即可“取消阻塞”，但如果应用程序未继续发送数据，则需要手动将这些数据刷新到主机。能否继续发送数据取决于驱动程序，默认配置的非阻塞驱动程序和 VFS 实现会在换行后自动刷新，而基于中断的阻塞驱动程序会在发送缓冲区变空时自动刷新。

睡眠模式考虑因素

USB 串行/JTAG 控制器及其关联的 USB PHY 均由特定时钟控制（如 APB 和 USB PHY 时钟），并属于特定的电源域（如数字电源域）。因此，对 USB 串行/JTAG 控制器关联的时钟和电源域进行任何更改，如进入另一睡眠模式，都可能影响控制器操作。

Deep-sleep 进入 Deep-sleep 模式时，USB 串行/JTAG 控制器及 USB PHY 关闭，导致 USB PHY 的 D+ 线不再变成高电平。因此：

- 进入 Deep-sleep 模式时，USB 串行/JTAG 设备会对主机/PC 显示为断开连接。即使 USB 电缆在物理层面上处于连接状态，仍会显示为断开连接。
- 退出 Deep-sleep 模式时，USB 串行/JTAG 设备会重新连接到主机/PC。

Light-sleep 进入 Light-sleep 模式时，APB 和 USB PHY 时钟关闭。因此，USB 串行/JTAG 控制器将无法接收或响应任何来自相连主机的 USB 事务，包括周期性的 CDC Data IN 事务。因此：

- 进入 Light-sleep 模式时，USB 串行/JTAG 设备无法响应主机/PC 的 USB CDC 驱动程序。主机/PC 可能会报告 USB 串行/JTAG 设备已断开连接，或连接出错。即使 USB 电缆在物理层面上处于连接状态，仍会显示为断开连接。
- 退出 Light-sleep 模式时，由于 USB PHY 的 D+ 线在 Light-sleep 期间始终处于高电平，可能导致主机/PC 不会重新枚举（即重新连接）USB 串行/JTAG 设备。用户可能需要在物理层面上断开 USB 电缆，并重新连接。

自动与手动进入睡眠模式 如果通过`esp_light_sleep_start()`或`esp_deep_sleep_start()`手动进入睡眠模式，需注意，USB 串行/JTAG 控制器在睡眠期间不工作。ESP-IDF **没有添加任何拒绝进入睡眠的安全检查**，即使 USB 串行/JTAG 控制器已连接，也会进入睡眠模式。如果在连接了 USB 串行/JTAG 控制器的情况下进入睡眠，可以通过拔出并重新插入 USB 电缆来重新建立连接。

如果通过`esp_pm_configure()`自动进入睡眠模式，请启用*CONFIG_USJ_NO_AUTO_LS_ON_CONNECTION*选项，该选项支持 ESP32-P4 自动检测 USB 串行/JTAG 控制器与主机的连接情况，并在连接持续时，阻止程序自动进入睡眠模式。注意，此选项会增加功耗。

4.26 低功耗模式使用指南

对于物联网应用场景，终端的待机性能表现十分重要，本文档旨在介绍 ESP32-P4 低功耗的基本原理，同时介绍 ESP32-P4 支持的低功耗模式，需注意本文档主要针对 station mode。文档还会具体给出每种模式的配置步骤、推荐配置和功耗表现，以帮助用户根据实际需求快速配置适合的低功耗模式。

4.26.1 系统低功耗模式介绍

低功耗模式不仅涉及到系统相关问题，还涉及到芯片具体的工作场景，如处在 Wi-Fi 工作场景就会与处在蓝牙工作场景时产生不同。为此本节将首先介绍纯系统角度，即不涉及具体场景的低功耗模式，主要有 DFS、Light-sleep、Deep-sleep。纯系统下的低功耗模式主要思想就是在休眠时关闭或门控一些功能模块来降低功耗。

DFS

DFS (Dynamic frequency scaling) 即动态频率切换，是 ESP-IDF 中集成的电源管理机制的基础功能。DFS 可以根据应用程序持有电源锁的情况，调整外围总线 (APB) 频率和 CPU 频率。持有高性能锁就使用高频，空闲状态不持有电源锁时则使用低频来降低功耗，以此来尽可能减少运行应用程序的功耗。

DFS 的调频机制即根据持有电源锁的最大频率需求来调整频率，同时，freertos tick rates 的数值也会对 DFS 调频产生影响。系统任务调度的灵敏度越大，则意味着系统能更及时的根据需求调整频率。有关调频机制的详细信息，请参见电源管理。

下图为 DFS 调频机制运行的理想电流情况。

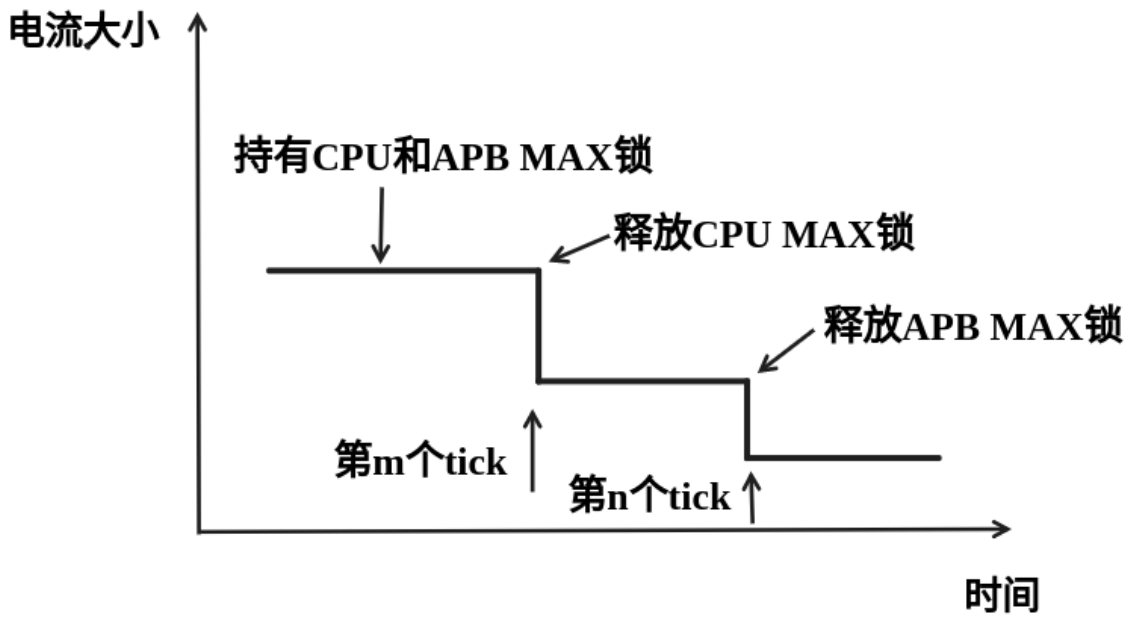


图 28: 理想 DFS 机制调频电流图

DFS 适用于 CPU 必须处于工作状态但是对低功耗有需求的场景，因此 DFS 经常与其他低功耗模式共同开启，下文会详细介绍。

Light-sleep

Light-sleep 模式是 ESP32-P4 预设的一种低功耗模式，其核心思想就是在休眠时关闭或门控一些功能模块来降低功耗。从纯系统方面来说，Light-sleep 模式有两种进入方式，一种是通过 API 调用进入休眠，一种是自动进入的 auto 模式。两种模式都需要配置唤醒源进行唤醒，同时在进入休眠后会门控或关闭一些模块。这里主要介绍 Auto Light-sleep 模式。

Auto Light-sleep 模式是 ESP-IDF 电源管理机制和 Light-sleep 模式的结合。开启电源管理机制是其前置条件，auto 体现在系统进入空闲状态 (IDLE) 超过设定时间后，自动进入 Light-sleep。空闲状态下，应用程序释放所有电源锁，此时，DFS 将降频以减小功耗。

Auto Light-sleep 依赖于电源管理机制，系统经过提前判断，发现空闲时间超过设定时间时，则直接进入休眠。该过程为自动进行。休眠时会自动关闭 RF、8 MHz 振荡器、40 MHz 高速晶振、PLL、门控数字内核时钟，暂停 CPU 工作。

Auto Light-sleep 模式需配置唤醒源。该模式拥有多种唤醒源，支持相互组合，此时任何一个唤醒源都可以触发唤醒。唤醒后，会从进入休眠的位置继续执行程序。若不配置唤醒源，进入 Light-sleep 休眠后，芯片将一直处在睡眠状态，直到外部复位。具体唤醒源有 RTC 定时器、触摸传感器、外部唤醒 (ext0)、外部唤醒 (ext1)、ULP 协处理器、SDIO、GPIO、UART、Wi-Fi、BT 唤醒等。

Auto Light-sleep 模式工作流程相对复杂，但是进入休眠状态是自动进行，同时需注意在进入前配置好唤醒源，防止芯片一直处在休眠状态。

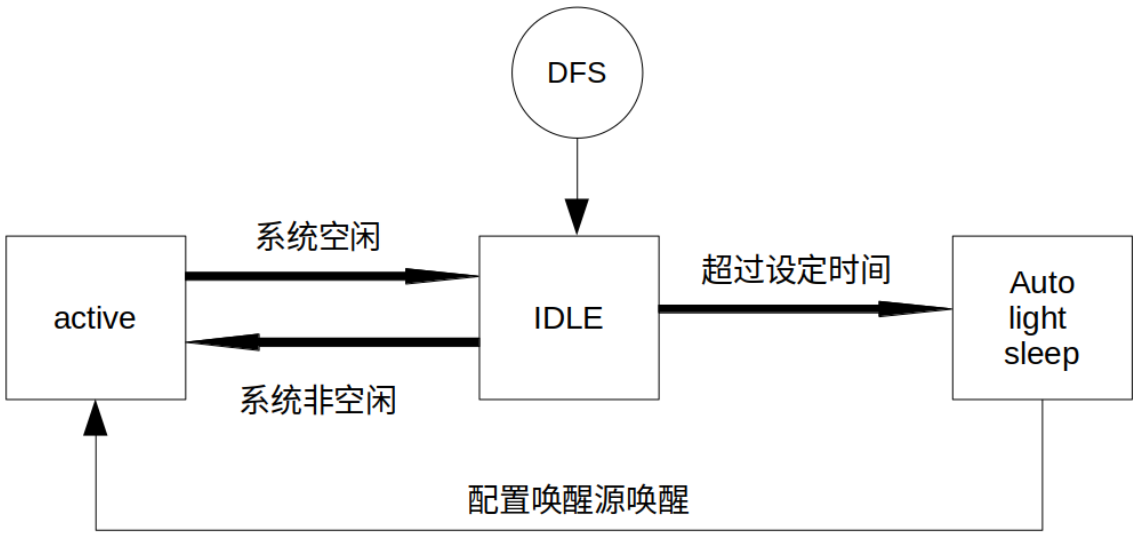


图 29: Auto Light-sleep 模式工作流程图

根据 Auto Light-sleep 的工作流程可得其理想电流图，关键节点均在图上标出。

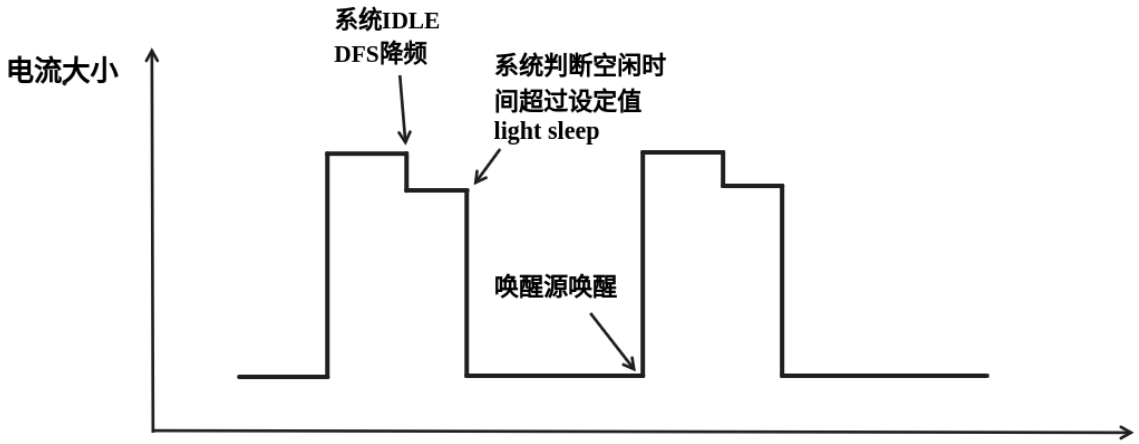


图 30: Auto Light-sleep 模式模式理想电流图

备注： 为更加清晰地展现出 Auto Light-sleep 的主要变化，图中省略了 DFS 降频过程。

Auto Light-sleep 模式适用于不需要实时响应外界需求的场景。

Deep-sleep

Deep-sleep 模式是为了追求更好的功耗表现所设计，休眠时仅保留 RTC 控制器、RTC 外设（可配置）、ULP 协处理器、RTC 高速内存、RTC 低速内存，其余模块全部关闭。与 Light-sleep 类似，Deep-sleep 同样通过 API 进入，且需要配置唤醒源进行唤醒。

Deep-sleep 通过调用 API 进入，休眠时会关闭除 RTC 控制器、RTC 外设、ULP 协处理器、RTC 高速内存、RTC 低速内存外的所有模块。

Deep-sleep 模式需配置唤醒源，其拥有多种唤醒源，这些唤醒源也可以组合在一起，此时任何一个唤醒源都可以触发唤醒。若不配置唤醒源进入 Deep-sleep 模式，芯片将一直处在睡眠状态，直到外部复位。具体唤醒源有 RTC 定时器、触摸传感器、外部唤醒 (ext0)、外部唤醒 (ext1)、ULP 协处理器、GPIO 唤醒等。

Deep-sleep 模式工作流程如下图所示：

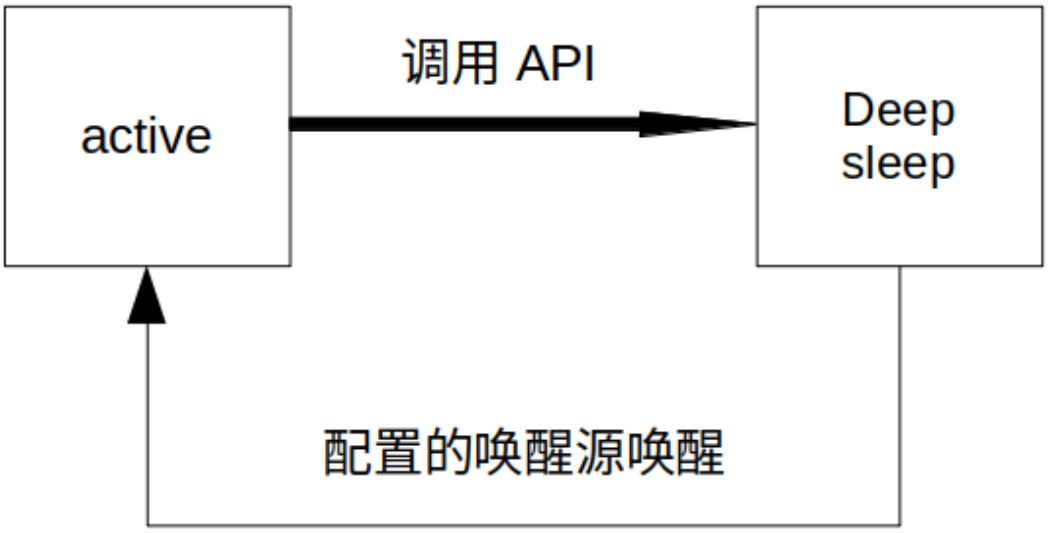


图 31: Deep-sleep 模式工作流程图

Deep-sleep 模式主要应用场景决定了系统很长时间才会苏醒一次，完成工作后又会继续进入 Deep-sleep，所以其理想电流图如下。

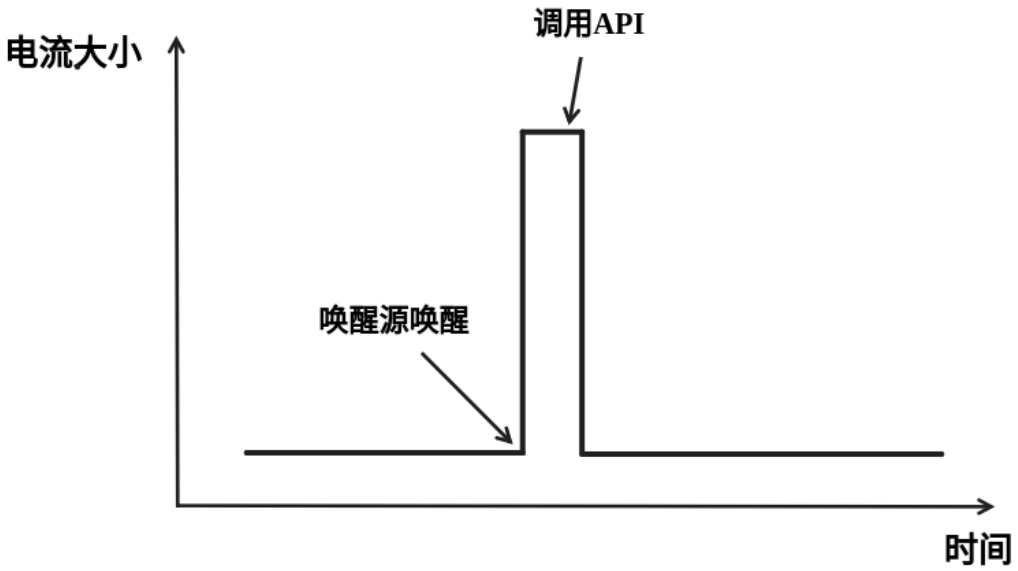


图 32: Deep-sleep 模式理想电流图

Deep-sleep 可以用于低功耗的传感器应用，或是大部分时间都不需要进行数据传输的情况，也就是通常所说的待机模式。设备可以每隔一段时间从 Deep-sleep 状态醒来测量数据并上传，之后重新进入 Deep-sleep；也可以将多个数据存储于 RTC memory，然后一次性发送出去。

如何配置纯系统下低功耗模式

介绍完纯系统下的低功耗模式后，本节将介绍公共配置选项、每种模式独有的配置选项，以及相应低功耗模式 API 的使用说明，同时给出相应模式推荐的配置。

公共配置选项

DFS 配置

DFS 有如下可配置选项：

- **max_freq_mhz** 该参数表示最大 CPU 频率 (MHz)，即 CPU 最高性能工作时候的频率，一般设置为芯片参数的最大值。
- **min_freq_mhz** 该参数表示最小 CPU 频率 (MHz)，即系统处在空闲状态时 CPU 的工作频率。该字段可设置为晶振 (XTAL) 频率值，或者 XTAL 频率值除以整数。
- **light_sleep_enable** 使能该选项，系统将在空闲状态下自动进入 Light-sleep 状态，即 Auto Light-sleep 使能，上文已经具体介绍。

具体配置方法如下：

- 1. 使能 CONFIG_PM_ENABLE
- 2. 配置 max_freq_mhz 和 min_freq_mhz，方式如下：

```
esp_pm_config_t pm_config = {
        .max_freq_mhz = CONFIG_EXAMPLE_MAX_CPU_FREQ_MHZ,
        .min_freq_mhz = CONFIG_EXAMPLE_MIN_CPU_FREQ_MHZ,
        .light_sleep_enable = false
};
ESP_ERROR_CHECK(esp_pm_configure(&pm_config));
```

推荐配置：

配置名称	设置情况
CONFIG_PM_ENABLE	ON
RTOS Tick rate (Hz)	1000
max_freq_mhz	160
min_freq_mhz	40
light_sleep_enable	false

备注： 上表中不涉及的配置均是默认。

Light-sleep 配置

本节介绍 Auto Light-sleep 的推荐配置和配置步骤。

Auto Light-sleep 有如下可配置选项：

- **Minimum step to enter sleep mode** 该参数表示系统自动进入休眠的阈值。该参数单位为 RTOS Tick，故其表示的时间与 RTOS Tick rate 相关，例该参数值为 3，RTOS Tick rate 配置为 1000 Hz 时，即当系统空闲时间大于等于 3 ms 时进入休眠。
- **Put light sleep related codes in internal RAM** 如果使能该选项，一些 light-sleep 功能将被移至 IRAM，减少代码运行时间，降低系统功耗，IRAM 使用量将增加 1.8 kB。
- **Put RTOS IDLE related codes in internal RAM** 如果使能该选项，一些 RTOS IDLE 功能将被移至 IRAM，减少代码运行时间，降低系统功耗，IRAM 使用量将增加 260 B。
- **RTC slow clock source** 该参数表表示 RTC 慢速时钟源。系统休眠时计时器模块的时钟被门控，此时使用 RTC Timer 进行计时，唤醒后使用 RTC Timer 的计数值对系统时间进行补偿。

时钟源	精度	频偏
Internal 150 kHz OSC	约 6.7 us/cycle	大
External 32 kHz XTAL	约 30.5 us/cycle	小

- **Disable all GPIO when chip at sleep** 如果使能该选项，系统将在休眠过程中禁用所有 GPIO 管脚，消除 GPIO 漏电，降低功耗，但是休眠过程中 GPIO 无法进行信号输入和输出。

唤醒源：

- RTC Timer Wakeup
- GPIO Wakeup
- UART Wakeup
- Touchpad Wakeup
- External Wakeup (ext0)
- External Wakeup (ext1)
- ULP Coprocessor Wakeup

备注： 以上仅列出可配置唤醒源，详细介绍请参考睡眠模式。

配置方法：

- 1. 配置唤醒源
- 2. 使能 CONFIG_PM_ENABLE
- 3. 使能 CONFIG_FREERTOS_USE_TICKLESS_IDLE
- 4. 配置 DFS 参数
- 5. light_sleep_enable = true，具体如下：

```
esp_pm_config_t pm_config = {
  .max_freq_mhz = CONFIG_EXAMPLE_MAX_CPU_FREQ_MHZ,
  .min_freq_mhz = CONFIG_EXAMPLE_MIN_CPU_FREQ_MHZ,
  #if CONFIG_FREERTOS_USE_TICKLESS_IDLE
  .light_sleep_enable = true
  #endif
};
ESP_ERROR_CHECK(esp_pm_configure(&pm_config));
```

- 6. 配置介绍的其余相关参数

推荐配置：

Deep-sleep 配置

对 Deep-sleep 模式来说，除了唤醒源相关配置，其余配置意义已经不大。

Deep-sleep 有如下可配置选项：

- RTC Timer wakeup
- EXT0/1 wakeup
- Touchpad wakeup
- ULP wakeup

备注： 以上仅列出可配置唤醒源，详细介绍请参考睡眠模式。

配置步骤：

- 配置唤醒源
- 调用 API，具体如下：

```
/* Enter deep sleep */
esp_deep_sleep_start();
```

用户可以通过下列配置选项，让一些特定模块在休眠时保持开启状态：

- **Power up External 40 MHz XTAL** 在一些特殊应用中，部分模块对休眠时的时钟精度及稳定度有很高要求（例如 BT）。这种情况下，可以考虑在休眠过程中打开 External 40 MHz XTAL。打开和关闭代码如下：

```
ESP_ERROR_CHECK(esp_sleep_pd_config(ESP_PD_DOMAIN_XTAL, ESP_PD_OPTION_ON));
ESP_ERROR_CHECK(esp_sleep_pd_config(ESP_PD_DOMAIN_XTAL, ESP_PD_OPTION_
↪OFF));
```

- **Power up Internal 8 MHz OSC** 在一些特殊应用中，部分模块（例如 LEDC）将 Internal 8 MHz OSC 作为时钟源，并且希望在 Light-sleep 休眠过程中也可以正常使用。这种情况下，可以考虑在休眠过程中打开 Internal 8 MHz OSC。打开和关闭代码如下：

```
ESP_ERROR_CHECK(esp_sleep_pd_config(ESP_PD_DOMAIN_RTC8M, ESP_PD_OPTION_
↪ON));
ESP_ERROR_CHECK(esp_sleep_pd_config(ESP_PD_DOMAIN_RTC8M, ESP_PD_OPTION_
↪OFF));
```

4.27 Wi-Fi Expansion

ESP32-P4 does not support Wi-Fi functionality natively, but it is possible to use the same Wi-Fi API and features using Wi-Fi expansion.

The principle of Wi-Fi Expansion operation is to add another ESP32 series, Wi-Fi capable, target connected to the ESP32-P4 in a pre-defined way. Your project could then include the esp_wifi_remote component using:

```
idf.py add-dependency esp_wifi_remote
```

Please follow the instructions described in the esp_wifi_remote documentation.

To explore the Wi-Fi Expansion functionality on ESP32-P4, you can get started with this example: examples/protocols/mqtt/tcp/README.md and choose Wi-Fi connection in the project configuration menu.

Chapter 5

迁移指南

5.1 迁移到 ESP-IDF 5.x

5.1.1 从 4.4 迁移到 5.0

迁移构建系统至 ESP-IDF v5.0

从 GNU Make 构建系统迁移至 ESP-IDF v5.0 ESP-IDF v5.0 已不再支持基于 Make 的工程，请参考从 *ESP-IDF GNU Make 构建系统迁移到 CMake 构建系统* 进行迁移。

更新片段文件语法 ESP-IDF v5.0 中将不再支持 ESP-IDF v3.x 中链接器脚本片段文件的旧式语法。在迁移的过程中需注意以下几点：

- 必须缩进，缩进不当的文件会产生解析异常；旧版本不强制缩进，但之前的文档和示例均遵循了正确的缩进语法。
- 条件改用 `if...elif...else` 结构，可以参照之前的章节。
- 映射片段和其他片段类型一样，需有名称。

明确指定组件依赖 在之前的 ESP-IDF 版本中，除了通用组件依赖项，还有一些组件总是作为公共依赖项，在构建时添加至每个组件中，如：

- `driver`
- `efuse`
- `esp_timer`
- `lwip`
- `vfs`
- `esp_wifi`
- `esp_event`
- `esp_netif`
- `esp_eth`
- `esp_phy`

这意味着可以直接包含这些组件的头文件，而无需在 `idf_component_register` 中将它们指定为依赖。此行为是由各种常见组件的传递依赖关系引起的。

在 ESP-IDF v5.0 中，此行为已修复，这些组件不再默认作为公共依赖项添加。

如果组件所依赖的某个组件不属于通用组件依赖项，则必须显式地声明此依赖关系。可以通过在组件的 CMakeLists.txt 中的 idf_component_register 调用中添加 REQUIRES <component_name> 或 PRIV_REQUIRES <component_name> 来完成。有关指定组件依赖的更多信息，请参阅组件依赖。

设置 COMPONENT_DIRS 和 EXTRA_COMPONENT_DIRS 变量 为了实现构建项目时的路径能够包含空格，ESP-IDF v5.0 做了一系列改进，其中包括改进了 CMakeLists.txt 文件中的 COMPONENT_DIRS 和 EXTRA_COMPONENT_DIRS 变量。

ESP-IDF v5.0 版本中，不再支持添加不存在的目录到变量 COMPONENT_DIRS 或 EXTRA_COMPONENT_DIRS 中，否则会出现报错。

同时，ESP-IDF v5.0 中也不再支持使用字符串拼接的方式定义 COMPONENT_DIRS 或 EXTRA_COMPONENT_DIRS 变量。这些变量应该定义为 CMake 列表。例如:

```
set(EXTRA_COMPONENT_DIRS path1 path2)
list(APPEND EXTRA_COMPONENT_DIRS path3)
```

不支持:

```
set(EXTRA_COMPONENT_DIRS "path1 path2")
set(EXTRA_COMPONENT_DIRS "${EXTRA_COMPONENT_DIRS} path3")
```

将这些变量定义为 CMake 列表的方式兼容之前的 ESP-IDF 版本。

更新 target_link_libraries 用法 ESP-IDF v5.0 修复了组件的 CMake 变量传播问题。此问题导致本应该只应用于某一组件的编译器标志和定义应用到了项目中的每个组件。

该修复也带来一定的副作用，从 ESP-IDF v5.0 开始，用户项目在使用 target_link_libraries 时必须明确指定 project_elf，同时自定义 CMake 项目必须指定 PRIVATE、PUBLIC 或 INTERFACE 参数。这是一项重大变更，不兼容以前的 ESP-IDF 版本。

例如:

```
target_link_libraries(${project_elf} PRIVATE "-Wl,--wrap=esp_panic_handler")
```

不支持:

```
target_link_libraries(${project_elf} "-Wl,--wrap=esp_panic_handler")
```

更新 CMake 版本 在 ESP-IDF v5.0 中，最低 CMake 版本已更新到 3.16，并且不再支持低于 3.16 的版本。如果你的操作系统没有安装 CMake，请运行 tools/idf_tools.py install cmake 来安装合适的版本。

该变更会影响到使用系统提供的 CMake 以及自定义 CMake 的 ESP-IDF 用户。

重新定义特定目标配置文件的应用顺序 ESP-IDF v5.0 重新安排了特定目标配置文件和 SDKCONFIG_DEFAULTS 中所有其他文件的应用顺序。现在，特定目标的配置文件将在引入它的文件之后、在 SDKCONFIG_DEFAULTS 中后续的其他文件之前应用。

例如:

```
如果 ``SDKCONFIG_DEFAULTS="sdkconfig.defaults;sdkconfig_devkit1
↪"``，且同一文件夹内有 ``sdkconfig.defaults.esp32``␣
↪文件，那么文件的应用顺序为：(1) sdkconfig.defaults (2) sdkconfig.defaults.esp32␣
↪(3) sdkconfig_devkit1
```

如果某个键在不同的特定目标配置文件中有不同的值，那么后者的值会覆盖前者。例如在以上案例中，如果某个键在 sdkconfig.defaults.esp32 和 sdkconfig_devkit1 中的值不同，则在 sdkconfig_devkit1 中的值会覆盖在 sdkconfig.defaults.esp32 中的值。

如果确实需要设置特定目标的配置值，请将其放到后应用的特定目标文件中，如 `sdkconfig_devkit1.esp32`。

GCC

GCC 版本 ESP-IDF 之前使用的 GCC 版本为 8.4.0，现已针对所有芯片目标升级至 GCC 11.2.0。若需要将代码从 GCC 8.4.0 迁移到 GCC 11.2.0，请参考以下官方 GCC 迁移指南。

- 迁移至 GCC 9
- 迁移至 GCC 10
- 迁移至 GCC 11

警告 升级至 GCC 11.2.0 后会触发新警告，或是导致原有警告内容发生变化。所有 GCC 警告的详细内容，请参考 GCC 警告选项。建议用户仔细检查代码，并设法解决这些警告。但由于某些警告的特殊性及用户代码的复杂性，有些警告可能为误报，需要进行关键修复。在这种情况下，用户可以采取多种方式来抑制这些警告。本节介绍了用户可能遇到的常见警告及如何抑制这些警告。

注意： 建议用户在抑制警告之前仔细确认该警告是否确实为误报。

`-Wstringop-overflow`、`-Wstringop-overread`、`-Wstringop-truncation` 和 `-Warray-bounds` 如果编译器不能准确判断内存或字符串的大小，使用 memory/string copy/compare 函数的用户会遇到某种 `-Wstringop` 警告。下文展示了触发这些警告的代码，并介绍了如何抑制这些警告。

```
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wstringop-overflow"
#pragma GCC diagnostic ignored "-Warray-bounds"
    memset(RTC_SLOW_MEM, 0, CONFIG_ULP_COPROC_RESERVE_MEM); // <<-- 此行触发了警告
#pragma GCC diagnostic pop
```

```
#pragma GCC diagnostic push
#if __GNUC__ >= 11
#pragma GCC diagnostic ignored "-Wstringop-overread" // <<-- 此键从 GCC 11 开始引入
#endif
#pragma GCC diagnostic ignored "-Warray-bounds" (-Warray-bounds) 。
            memcpy(backup_write_data, (void *)EFUSE_PGM_DATA0_REG, sizeof(backup_
↪write_data)); // <<-- 此行触发了警告
#pragma GCC diagnostic pop
```

`-Waddress-of-packed-member` 当访问打包 `struct` 中的某个未对齐成员时，由于非对齐内存访问会对性能产生影响，GCC 会触发 `-Waddress-of-packed-member` 警告。然而，所有基于 Xtensa 或 RISC-V 架构的 ESP 芯片都允许非对齐内存访问，并且不会产生额外的性能影响。因此，在大多数情况下，可以忽略此问题。

```
components/bt/host/bluedroid/btc/profile/std/gatt/btc_gatt_util.c: In function
↪'btc_to_bta_gatt_id':
components/bt/host/bluedroid/btc/profile/std/gatt/btc_gatt_util.c:105:21: warning:␣
↪taking address of packed member of 'struct <anonymous>' may result in an␣
↪unaligned pointer value [-Waddress-of-packed-member]
  105 |     btc_to_bta_uuid(&p_dest->uuid, &p_src->uuid);
      |                     ^~~~~~~~~~~~
```

如果该警告在多个源文件中多次出现，可以在 CMake 级别抑制该警告，如下所示。

```
set_source_files_properties(
    "host/bluedroid/bta/gatt/bta_gattc_act.c"
    "host/bluedroid/bta/gatt/bta_gattc_cache.c"
    "host/bluedroid/btc/profile/std/gatt/btc_gatt_util.c"
    "host/bluedroid/btc/profile/std/gatt/btc_gatts.c"
    PROPERTIES COMPILE_FLAGS -Wno-address-of-packed-member)
```

但如果只有一或两处警告，可以直接在源代码中进行抑制，如下所示。

```
#pragma GCC diagnostic push
#if     __GNUC__ >= 9
#pragma GCC diagnostic ignored "-Waddress-of-packed-member" <<-- 此键从 GCC 11␣
↪开始引入
#endif
    uint32_t* reg_ptr = (uint32_t*)src;
#pragma GCC diagnostic pop
```

llabs() 用于 64 位整数 stdlib.h 中的函数 abs() 需要使用 int 参数。请在计划为 64 位的类型中使用 llabs()，尤其是 time_t。

乐鑫工具链更新

Xtensa 编译器中的 int32_t 和 uint32_t 在 Xtensa 编译器中，int32_t 和 uint32_t 类型已分别从 int 和 unsigned int 更新为 long 和 unsigned long。此更新现与上游 GCC 相匹配，上游 GCC 在 Xtensa、RISC-V 和其他架构上使用 long 整数来表示 int32_t 和 uint32_t。

	2021r2 及以上版本，GCC 8	2022r1，GCC 11
Xtensa	(unsigned) int	(unsigned) long
riscv32	(unsigned) long	(unsigned) long

若代码中使用了 <inttypes.h> 提供的类型来格式化字符串，则这些代码会受到上述变化的影响。使用这些宽度固定的类型（例如 uint32_t）时，请使用 PRIi32、PRIx32 来分别替换 %i、%x 等占位符。**只有**在 <inttypes.h> 中定义的类型（例如 int）需要这种特殊格式。

在其他情况下，请注意枚举支持 int 类型。

通常，int32_t 和 int 为不同的类型。同样，uint32_t 和 unsigned int 也为不同的类型。

如果用户在其应用中没有对格式化字符串进行上述更新，程序会报错，如下所示：

```
/Users/name/esp/esp-rainmaker/components/esp-insights/components/esp_diagnostics/
↪include/esp_diagnostics.h:238:29: error: format '%u' expects argument of type
↪'unsigned int', but argument 3 has type 'uint32_t' {aka 'long unsigned int'} [-
↪Werror=format=]
238 |     esp_diag_log_event(tag, "EV (%u) %s: " format, esp_log_timestamp(), tag,
↪##__VA_ARGS__); \
    |                                ^~~~~~~~~~~~~~         ~~~~~~~~~~~~~~~~~~~
    |                                                       |
    |                                                       uint32_t {aka long␣
↪unsigned int}
                                                 uint32_t {aka long unsigned int}
```

移除构建选项 CONFIG_COMPILER_DISABLE_GCC8_WARNINGS 原有的 CONFIG_COMPILER_DISABLE_GCC8_WARNINGS 选项用于构建使用现已僵化的 GCC 5 工具链编写的陈旧代码。但由于已经过去较长时间，现在可以对警告进行修复，因此该选项已移除。

目前，在 GCC 11 中，建议用户仔细检查代码，尽量解决编译器警告。

网络

Wi-Fi

回调函数类型 esp_now_recv_cb_t 此前 esp_now_recv_cb_t 的第一个参数的类型是 const uint8_t *mac_addr，该参数只包含对端 ESP-NOW 设备的地址。

现在该函数有所更新。第一个参数的类型变更为 esp_now_recv_info_t，它包含三个成员变量 src_addr，des_addr 和 rx_ctrl。因此，需要进行如下更新:

- 重新定义的 ESP-NOW 收包回调函数。
- src_addr 可以等价替换原来的 mac_addr。
- des_addr 是 ESP-NOW 包的目的 MAC 地址，可以是单播或广播地址。使用 des_addr 可以区分单播或广播的 ESP-NOW 包，其中，即使是在加密的 ESP-NOW 配置中，广播的 ESP-NOW 包也可以是非加密的。
- rx_ctrl 是 ESP-NOW 包的 Rx control info，它包含此包的更多有用信息。

请参考 ESP-NOW 样例：wifi/espnow/main/espnow_example_main.c

以太网

esp_eth_ioctl() API 此前，*esp_eth_ioctl()* API 存在以下问题:

- 在某些情况下，第三个参数（数据类型为 void /*）可以接受 int/bool 类型实参（而非指针）作为输入。然而，文档中未描述这些情况。
- 为了将 int/bool 类型实参作为第三个参数传递，实参将强制转换为 void * 类型，以防出现如下所示的编译器警告。此等转换可能引起*esp_eth_ioctl()* 函数的滥用。

```
esp_eth_ioctl(eth_handle, ETH_CMD_S_FLOW_CTRL, (void *)true);
```

因此，我们统一了*esp_eth_ioctl()* 的用法。现在，该结构体的第三个参数在传递时必须作为指向特定数据类型的指针，表示*esp_eth_ioctl()* 读取/存储数据的位置。*esp_eth_ioctl()* 的用法如下列代码所示。

设置以太网配置的用例如下:

```
eth_duplex_t new_duplex_mode = ETH_DUPLEX_HALF;
esp_eth_ioctl(eth_handle, ETH_CMD_S_DUPLEX_MODE, &new_duplex_mode);
```

获取以太网配置的用例如下:

```
eth_duplex_t duplex_mode;
esp_eth_ioctl(eth_handle, ETH_CMD_G_DUPLEX_MODE, &duplex_mode);
```

KSZ8041/81 和 LAN8720 驱动更新 KSZ8041/81 和 LAN8720 驱动现已更新，以支持相关产品系列中的更多设备（如新一代设备）。上述驱动能够识别特定芯片编号及驱动提供的潜在支持。

更新之后，通用函数将替代特定“芯片编号”函数得以调用:

- 删除 esp_eth_phy_new_ksz8041() 以及 esp_eth_phy_new_ksz8081()，转而使用*esp_eth_phy_new_ksz80xx()*
- 删除 esp_eth_phy_new_lan8720()，转而使用*esp_eth_phy_new_lan87xx()*

ESP NETIF Glue 时间处理程序 `esp_eth_set_default_handlers()` 和 `esp_eth_clear_default_handlers()` 函数现已删除。现在可以自动处理以太网默认 IP 层处理程序的注册。如果在注册以太网/IP 事件处理程序之前，你已经按照建议，完全初始化以太网驱动和网络接口，则无需执行任何操作（除了删除受影响的函数）。否则，在注册用户事件处理程序后，应随即启动以太网驱动。

PHY 地址自动检测 以太网 PHY 地址自动检测函数 `esp_eth_detect_phy_addr()` 已重命名为*`esp_eth_phy_802_3_detect_phy_addr()`*，其声明移至 esp_eth/include/esp_eth_phy_802_3.h。

SPI 以太网模块初始化 SPI 以太网模块的初始化过程已经简化。此前，你需要在实例化 SPI 以太网 MAC 之前，使用*`spi_bus_add_device()`* 手动分配 SPI 设备。

现在，SPI 设备已在内部分配，因此无需再调用*`spi_bus_add_device()`*。`eth_dm9051_config_t`、`eth_w5500_config_t` 和 `eth_ksz8851snl_config_t` 配置结构体现已包含 SPI 设备配置成员（例如，可以微调可能依赖 PCB 设计的 SPI 时序）。`ETH_DM9051_DEFAULT_CONFIG`、`ETH_W5500_DEFAULT_CONFIG` 和 `ETH_KSZ8851SNL_DEFAULT_CONFIG` 配置初始化宏也已接受新的参数输入。了解 SPI 以太网模块初始化示例，请查看*以太网 API 参考指南*。

Ethernet 驱动 用于创建 MAC 实例的 API (*esp_eth_mac_new_*()*) 的输入参数由一个配置参数改为两个，这两个参数用于

- 供应商特定的 MAC 配置
- Ethernet 驱动 MAC 配置

该更新不仅适用于内部 Ethernet MAC *`esp_eth_mac_new_esp32()`* 也适用于外部 MAC 设备，如 `esp_eth_mac_new_ksz8851snl()`、`esp_eth_mac_new_dm9051()` 和 `esp_eth_mac_new_w5500()`。

TCP/IP 适配器 TCP/IP 适配器是在 ESP-IDF v4.1 之前使用的网络接口抽象组件。本文档概述了从 `tcpip_adapter` API 迁移至*ESP-NETIF* 的过程。

更新网络连接代码

网络软件栈初始化

- 你只需用 `esp_netif_init()` 替换 `tcpip_adapter_init()`，注意 `esp_netif_init()` 函数现将返回标准错误代码。了解详细信息，请参考*ESP-NETIF*。
- `esp_netif_deinit()` 函数用于反初始化网络软件栈。
- 你还需用 `#include "esp_netif.h"` 替换 `#include "tcpip_adapter.h"`。

创建网络接口 更新之前，TCP/IP 适配器静态定义了以下三个接口：

- Wi-Fi Station
- Wi-Fi AP
- 以太网

接口定义现已更新。网络接口的设计应严格参考*ESP-NETIF*，使其能够连接至 TCP/IP 软件栈。例如，在 TCP/IP 软件栈和事件循环初始化完成后，Wi-Fi 的初始化代码必须显示调用 `esp_netif_create_default_wifi_sta();` 或 `esp_netif_create_default_wifi_ap();`。

请参考上述三个接口的初始化代码示例：

- Wi-Fi Station：wifi/getting_started/station/main/station_example_main.c
- Wi-Fi AP：wifi/getting_started/softAP/main/softap_example_main.c
- 以太网：ethernet/basic/main/ethernet_example_main.c

其他 tcpip_adapter API 更换 所有 tcpip_adapter 函数都有对应的 esp-netif。请参考以下章节中的 esp_netif.h 部分，了解更多信息：

- Setters/Getters
- DHCP
- DNS
- IP address

TCP/IP 适配器 API tcpip_adapter_get_sta_list() 用于获取与软件接入点 (softAP) 相关联的 Wi-Fi 站点列表，现已移到 Wi-Fi 组件，并更名为 esp_wifi_ap_get_sta_list_with_ip()，它是 ESP-NETIF API *esp_netif_dhcps_get_clients_by_mac()* 一个特例。无论服务器在哪个网络接口上运行，该 API 都可更方便地提供连接到 DHCP 服务器的客户端列表。

默认事件处理程序 事件处理程序已从 tcpip_adapter 移至相应驱动程序代码。从应用程序的角度来看，这一变更不会产生任何影响，所有事件仍将以相同的方式处理。请注意，在与 IP 相关的事件处理程序中，应用程序代码通常以 esp-netif 结构体而非 LwIP 结构体的形式接收 IP 地址。两种结构体均兼容二进制格式。

打印地址的首选方式如下所示：

```
ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
```

不建议使用下述方式：

```
ESP_LOGI(TAG, "got ip:%s", ip4addr_ntoa(&event->ip_info.ip));
```

ip4addr_ntoa() 为 LwIP API，因此 esp-netif 还提供了替代函数 esp_ip4addr_ntoa()，然而总得来说仍推荐使用 IP2STR() 这一方法。

IP 地址 推荐使用 esp-netif 定义的 IP 结构。请注意，在启用默认兼容性时，LwIP 结构体仍然可以工作。

- esp-netif IP address definitions

外设

外设时钟门控 与更新之前相同，外设的时钟仍由驱动处理，用户无需对外设模块的时钟门控进行设置。

但是，如果用户想基于组件 hal 和 soc 开发自己的驱动，请注意时钟门控的头文件引用路径已由 driver/periph_ctrl.h 更新为 esp_private/periph_ctrl.h。

RTC 子系统控制 RTC 控制 API 已经从 driver/rtc_cntl.h 移动到了 esp_private/rtc_ctrl.h。

ADC

ADC 单次模式及连续模式驱动 ADC 单次模式的驱动已更新。

- 新的驱动位于组件 esp_adc 中，头文件引用路径为 esp_adc/adc_oneshot.h。
- 旧版驱动仍然可用，其头文件引用路径为 driver/adc.h。

对于 ADC 连续模式驱动，其位置已由组件 driver 更新为 esp_adc。

- 头文件引用路径由 driver/adc.h 更新为 esp_adc/adc_continuous.h。

但是，引用两种模式的旧版路径 driver/adc.h 会默认触发如下编译警告，可通过配置 Kconfig 选项*CONFIG_ADC_SUPPRESS_DEPRECATE_WARN* 关闭该警告。

```
legacy adc driver is deprecated, please migrate to use esp_adc/adc_oneshot.h and␣
↪esp_adc/adc_continuous.h for oneshot mode and continuous mode drivers␣
↪respectively
```

ADC 校准驱动 ADC 校准驱动已更新。

- 新的驱动位于组件 esp_adc 中，头文件引用路径为 esp_adc/adc_cali.h 和 esp_adc/adc_cali_scheme.h。

旧版驱动仍然可用，其头文件引用路径为 esp_adc_cal.h。如果用户要使用旧版路径，需要将组件 esp_adc 添加到文件 CMakeLists.txt 的组件需求表中。

默认情况下，引用路径 esp_adc_cal.h 会默认触发如下编译警告，可通过配置 Kconfig 选项*CONFIG_ADC_CALI_SUPPRESS_DEPRECATE_WARN* 关闭该警告。

```
legacy adc calibration driver is deprecated, please migrate to use esp_adc/adc_
↪cali.h and esp_adc/adc_cali_scheme.h
```

API 更新

- ADC 电源管理 API adc_power_acquire 和 adc_power_release 已被移至 esp_private/adc_share_hw_ctrl.h，用于内部功能。
  - 更新前，由于硬件勘误表的工作原理，这两个 API 可以被用户调用。
  - 更新后，ADC 电源管理完全由驱动在内部实现。
  - 如果用户仍需调用这个 API，可以通过引用路径 esp_private/adc_share_hw_ctrl.h 来调用它。
- 更新后，driver/adc2_wifi_private.h 已被移至 esp_private/adc_share_hw_ctrl.h。
- adc_unit_t 中的枚举 ADC_UNIT_BOTH，ADC_UNIT_ALTER 及 ADC_UNIT_MAX 已被删除。
- 由于只有部分芯片支持下列枚举的某些取值，因此将下列枚举删除。如果用户使用了不支持的取值，会造成驱动运行错误。
  - 枚举 ADC_CHANNEL_MAX
  - 枚举 ADC_ATTEN_MAX
  - 枚举 ADC_CONV_UNIT_MAX
- ESP32 中的 API hall_sensor_read 已被删除，因此 ESP32 不再支持霍尔传感器。
- API adc_set_i2s_data_source 和 adc_i2s_mode_init 已被弃用，相关的枚举 adc_i2s_source_t 也已被弃用，请使用 esp_adc/adc_continuous.h 进行迁移。
- API adc_digi_filter_reset，adc_digi_filter_set_config，adc_digi_filter_get_config 和 adc_digi_filter_enable 已被移除. 这些接口的行为不被保证。枚举 adc_digi_filter_idx_t，adc_digi_filter_mode_t 和结构体 adc_digi_iir_filter_t 已被移除。
- API esp_adc_cal_characterize 已被弃用，请迁移到 adc_cali_create_scheme_curve_fitting 或 adc_cali_create_scheme_line_fitting.
- API esp_adc_cal_raw_to_voltage 已被弃用，请迁移到 adc_cali_raw_to_voltage.
- API esp_adc_cal_get_voltage 已被弃用，请迁移到 adc_oneshot_get_calibrated_result.

GPIO

- 之前的 Kconfig 选项 *RTCIO_SUPPORT_RTC_GPIO_DESC* 已被删除，因此数组 rtc_gpio_desc 已不可用，请使用替代数组 rtc_io_desc。
- 更新后，用户回调函数无法再通过读取 GPIO 中断的状态寄存器来获取用于触发中断的 GPIO 管脚的编号。但是，用户可以通过使用回调函数变量来确定该管脚编号。
  - 更新前，GPIO 中断发生时，GPIO 中断状态寄存器调用用户回调函数之后，会被清空。因此，用户可以在回调函数中读取 GPIO 中断状态寄存器，以便确定触发中断的 GPIO 管脚。
  - 但是，在调用回调函数后清空中断状态寄存器可能会导致边沿触发的中断丢失。例如，在调用用户回调函数时，如果某个边沿触发的中断 (re) 被触发，该中断会被清除，并且其注册的用户回调函数还未被处理。

- 更新后，GPIO 的中断状态寄存器在调用用户回调函数 **之前**被清空。因此，用户无法读取 GPIO 中断状态寄存器来确定哪个管脚触发了中断。但是，用户可以通过回调函数变量来传递被触发的管脚编号。

Sigma-Delta 调制器 Sigma-Delta 调制器的驱动现已更新为*SDM*。

- 新驱动中实现了工厂模式，SDM 通道都位于内部通道池中，因此用户无需手动将 SDM 通道配置到 GPIO 管脚。
- SDM 通道会被自动分配。

尽管我们推荐用户使用新的驱动 API，旧版驱动仍然可用，位于头文件引用路径 `driver/sigmadelta.h` 中。但是，引用 `driver/sigmadelta.h` 会默认触发如下编译警告，可通过配置 Kconfig 选项*CONFIG_SDM_SUPPRESS_DEPRECATE_WARN* 关闭该警告。

```
The legacy sigma-delta driver is deprecated, please use driver/sdm.h
```

概念与使用方法上的主要更新如下所示：

主要概念更新

- SDM 通道名称已由 `sigmadelta_channel_t` 更新为*`sdm_channel_handle_t`*，后者为一个不透明指针。
- SDM 通道配置原来存放于 `sigmadelta_config_t`，现存放于*`sdm_config_t`*。
- 旧版驱动中，用户无需为 SDM 通道设置时钟源。但是在新驱动中，用户需要在*`sdm_config_t::clk_src`* 为 SDM 通道设置合适的时钟源，*`soc_periph_sdm_clk_src_t`* 中列出了可用的时钟源。
- 旧版驱动中，用户需要为通道设置 `prescale`，该参数会影响调制器输出脉冲的频率。在新的驱动中，用户需要使用*`sdm_config_t::sample_rate_hz`* 实现该功能。
- 旧版驱动中，用户通过设置 `duty` 来改变输出的模拟量，现在换成了一个更贴切的名字 `density`

主要使用方法更新

- 更新前，通道配置由通道分配在*`sdm_new_channel()`* 完成。在新驱动中，只有 `density` 可在运行时由*`sdm_channel_set_pulse_density()`* 更新。其他参数如 `gpio number`、`prescale` 只能在通道分配时进行设置。
- 在进行下一步通道操作前，用户应通过调用*`sdm_channel_enable()`* 提前 **使能**该通道。该函数有助于管理一些系统级服务，如 **电源管理**。

定时器组驱动 为统一和简化通用定时器的使用，定时器组驱动已更新为*GPTimer*。

尽管我们推荐使用新的驱动 API，旧版驱动仍然可用，其头文件引用路径为 `driver/timer.h`。但是，引用 `driver/timer.h` 会默认触发如下编译警告，可通过配置 Kconfig 选项*CONFIG_GPTIMER_SUPPRESS_DEPRECATE_WARN* 关闭该警告。

```
legacy timer group driver is deprecated, please migrate to driver/gptimer.h
```

概念和使用方法上的主要更新如下所示：

主要概念更新

- 用于识别定时器的 `timer_group_t` 和 `timer_idx_t` 已被删除。在新驱动中，定时器用参数*`gptimer_handle_t`* 表示。
- 更新后，定时器的时钟源由*`gptimer_clock_source_t`* 定义，之前的时钟源参数 `timer_src_clk_t` 不再使用。
- 更新后，定时器计数方向由*`gptimer_count_direction_t`* 定义，之前的计数方向参数 `timer_count_dir_t` 不再使用。
- 更新后，仅支持电平触发的中断，`timer_intr_t` 和 `timer_intr_mode_t` 不再使用。

- 更新后，通过设置标志位*gptimer_alarm_config_t::auto_reload_on_alarm*，可以使能自动加载。timer_autoreload_t 不再使用。

主要使用方法更新

- 更新后，通过从*gptimer_new_timer()* 创建定时器示例可以初始化定时器。用户可以在*gptimer_config_t* 进行一些基本设置，如时钟源，分辨率和计数方向。请注意，无需在驱动安装阶段进行报警事件的特殊设置。
- 更新后，报警事件在*gptimer_set_alarm_action()* 中进行设置，参数在*gptimer_alarm_config_t* 中进行设置。
- 更新后，通过*gptimer_get_raw_count()* 设置计数数值，通过*gptimer_set_raw_count()* 获取计数数值。驱动不会自动将原始数据同步到 UTC 时间戳。由于定时器的分辨率已知，用户可以自行转换数据。
- 更新后，如果*gptimer_event_callbacks_t::on_alarm* 被设置为有效的回调函数，驱动程序也会安装中断服务。在回调函数中，用户无需配置底层寄存器，如用于“清除中断状态”，“重新使能事件”的寄存器等。因此，timer_group_get_intr_status_in_isr 与 timer_group_get_auto_reload_in_isr 这些函数不再使用。
- 更新后，当报警事件发生时，为更新报警配置，用户可以在中断回调中调用*gptimer_set_alarm_action()*，这样报警事件会被重新使能。
- 更新后，如果用户将*gptimer_alarm_config_t::auto_reload_on_alarm* 设置为 true，报警事件将会一直被驱动程序使能。

UART

删除/弃用项目	替代	备注
uart_isr_register()	无	更新后，UART 中断由驱动处理。
uart_isr_free()	无	更新后，UART 中断由驱动处理。
uart_config_t 中的 use_ref_tick	*uart_config_t::source_clk*	选择时钟源。
uart_enable_pattern_det_intr()	*uart_enable_pattern_det_baud_intr()*	使能模式检测中断。

I2C

删除/弃用项目	替代	备注
i2c_isr_register()	无	更新后，I2C 中断由驱动处理。
i2c_isr_register()	无	更新后，I2C 中断由驱动处理。
i2c_opmode_t	无	更新后，该项不再在 esp-idf 中使用。

SPI

删除/弃用项目	替代	备注
spi_cal_clock()	*spi_get_actual_clock()*	获取 SPI 真实的工作频率。

- 内部头文件 spi_common_internal.h 已被移至 esp_private/spi_common_internal.h。

SDMMC

删除/弃用项目	替代	备注
sdmmc_host_pullup_en()	在*sdmmc_slot_config_t::flags* 设置标志位 SDMMC_SLOT_FLAG_INTERNAL_PULLUP	使能内部上拉。

LEDC

删除/弃用项目	替代	备注
ledc_timer_config_t 中的 bit_num	*ledc_timer_config_t::duty_resolut*	设置占空比分辨率。

脉冲计数器 (PCNT) 驱动 为统一和简化 PCNT 外设，PCNT 驱动已更新，详见*PCNT*。

尽管我们推荐使用新的驱动 API，旧版驱动仍然可用，保留在头文件引用路径 driver/pcnt.h 中。但是，引用路径 driver/pcnt.h 会默认触发如下编译警告，可通过配置 Kconfig 选项*CONFIG_PCNT_SUPPRESS_DEPRECATE_WARN* 来关闭该警告。

```
legacy pcnt driver is deprecated, please migrate to use driver/pulse_cnt.h
```

主要概念和使用方法上的更新如下所示：

主要概念更新

- 更新后，pcnt_port_t、pcnt_unit_t 和 pcnt_channel_t 这些用于识别 PCNT 单元和通道的参数已被删除。在新的驱动中，PCNT 单元由参数*pcnt_unit_handle_t* 表示，PCNT 通道由参数*pcnt_channel_handle_t* 表示，这两个参数都是不透明指针。
- 更新后，不再使用 pcnt_evt_type_t，它们由统一的 **观察点事件** 表示。在事件回调函数*pcnt_watch_cb_t* 中，通过*pcnt_watch_event_data_t* 可以分辨不同观察点。
- pcnt_count_mode_t 更新为:cpp:type:*pcnt_channel_edge_action_t*，pcnt_ctrl_mode_t 更新为*pcnt_channel_level_action_t*。

主要使用方法更新

- 更新前，PCNT 的单元配置和通道配置都通过函数 pcnt_unit_config 实现。更新后，PCNT 的单元配置通过工厂 API *pcnt_new_unit()* 完成，通道配置通过工厂 API *pcnt_new_channel()* 完成。
  - 只需配置计数范围即可初始化一个 PCNT 单元。更新后，GPIO 管脚分配通过*pcnt_new_channel()* 完成。
  - 高/低电平控制模式和上升沿/下降沿计数模式分别通过函数*pcnt_channel_set_edge_action()* 和*pcnt_channel_set_level_action()* 进行设置。
- pcnt_get_counter_value 更新为*pcnt_unit_get_count()*。
- pcnt_counter_pause 更新为*pcnt_unit_stop()*。
- pcnt_counter_resume 更新为*pcnt_unit_start()*。
- pcnt_counter_clear 更新为*pcnt_unit_clear_count()*。
- 更新后，pcnt_intr_enable 与 pcnt_intr_disable 已被删除。新的驱动中，通过注册时间回调函数*pcnt_unit_register_event_callbacks()* 来使能中断。
- 更新后，pcnt_event_enable 与 pcnt_event_disable 已被删除。新的驱动中，可通过*pcnt_unit_add_watch_point()* 和*pcnt_unit_remove_watch_point()* 来增加/删除观察点，以使能/停用 PCNT 事件。
- 更新后，pcnt_set_event_value 已被删除。新的驱动中，通过*pcnt_unit_add_watch_point()* 增加观察点时，也同时设置了事件的数值。
- 更新后，pcnt_get_event_value 与 pcnt_get_event_status 已被删除。在新的驱动中，这些信息存储在*pcnt_watch_event_data_t* 的回调函数*pcnt_watch_cb_t* 中。
- 更新后，pcnt_isr_register 与 pcnt_isr_unregister 已被删除，不允许注册 ISR 句柄。用户可以通过调用:cpp:func:*pcnt_unit_register_event_callbacks* 来注册事件回调函数。
- 更新后，pcnt_set_pin 已被删除，新的驱动不再允许在运行时切换 GPIO 管脚。如果用户想切换为其他 GPIO 管脚，可通过:cpp:func:*pcnt_del_channel* 删除当前的 PCNT 通道，然后通过:cpp:func:*pcnt_new_channel* 安装新的 GPIO 管脚。
- pcnt_filter_enable，pcnt_filter_disable 与 pcnt_set_filter_value 更新为*pcnt_unit_set_glitch_filter()*。同时，pcnt_get_filter_value 已被删除。
- pcnt_set_mode 更新为*pcnt_channel_set_edge_action()* 与*pcnt_channel_set_level_action()*。

- pcnt_isr_service_install, pcnt_isr_service_uninstall, pcnt_isr_handler_add 与 pcnt_isr_handler_remove 更新为*pcnt_unit_register_event_callbacks()*。默认的 ISR 句柄已安装在新的驱动中。

温度传感器驱动 温度传感器的驱动已更新，推荐用户使用新驱动。旧版驱动仍然可用，但是无法与新驱动同时使用。

新驱动的头文件引用路径为 driver/temperature_sensor.h。旧版驱动仍然可用，保留在引用路径 driver/temp_sensor.h 中。但是，引用路径 driver/temp_sensor.h 会默认触发如下编译警告，可通过设置 Kconfig 选项*CONFIG_TEMP_SENSOR_SUPPRESS_DEPRECATE_WARN* 来关闭该警告。

```
legacy temperature sensor driver is deprecated, please migrate to driver/
↪temperature_sensor.h
```

配置内容已更新。更新前，用户需要设置 clk_div 与 dac_offset。更新后，用户仅需设置 tsens_range。

温度传感器的使用过程也已更新。更新前，用户可通过 config->start->read_celsius 获取数据。更新后，用户需要通过 temperature_sensor_install 先安装温度传感器的驱动，测量完成后需卸载驱动，详情请参考*Temperature Sensor*。

RMT 驱动 为统一和扩展 RMT 外设的使用，RMT 驱动已更新，详见*RMT transceiver*。

尽管我们建议使用新的驱动 API，旧版驱动仍然可用，保留在头文件引用路径 driver/rmt.h``中。但是，引用路径 ``driver/rmt.h 会默认触发如下编译警告，可通过配置 Kconfig 选项*CONFIG_RMT_SUPPRESS_DEPRECATE_WARN* 来关闭该警告。

```
The legacy RMT driver is deprecated, please use driver/rmt_tx.h and/or driver/rmt_
↪rx.h
```

主要概念和使用方法更新如下所示：

主要概念更新

- 更新后，用于识别硬件通道的 rmt_channel_t 已删除。在新的驱动中，RMT 通道用参数*rmt_channel_handle_t* 表示，该通道由驱动程序动态分配，而不是由用户指定。
- rmt_item32_t 更新为*rmt_symbol_word_t*，以避免在结构体中出现嵌套的共用体。
- 更新后，rmt_mem_t 已被删除，因为我们不允许用户直接访问 RMT 内存块（即 RMTMEM）。直接访问 RMTMEM 没有意义，反而会引发错误，特别是当 RMT 通道与 DMA 通道相连时。
- 更新后，由于 rmt_mem_owner_t 由驱动控制，而不是用户，因此 rmt_mem_owner_t 已被删除。
- rmt_source_clk_t 更新为*rmt_clock_source_t*，后者不支持二进制兼容。
- 更新后，rmt_data_mode_t 已被删除，RMT 内存访问模式配置为始终使用 Non-FIFO 和 DMA 模式。
- 更新后，由于驱动有独立的发送和接收通道安装函数，因此 rmt_mode_t 已被删除。
- 更新后，rmt_idle_level_t 已被删除，在*rmt_transmit_config_t::eot_level* 中可为发送通道设置空闲状态电平。
- 更新后，rmt_carrier_level_t 已被删除，可在*rmt_carrier_config_t::polarity_active_low* 设置载流子极性。
- 更新后，rmt_channel_status_t 与 rmt_channel_status_result_t 已被删除，不再使用。
- 通过 RMT 通道发送并不需要用户提供 RMT 符号，但是用户需要提供一个 RMT 编码器用来告诉驱动如何将用户数据转换成 RMT 符号。

主要使用方法更新

- 更新后，分别通过*rmt_new_tx_channel()* 和*rmt_new_rx_channel()* 安装发送通道和接收通道。
- 更新后，rmt_set_clk_div 和 rmt_get_clk_div 已被删除。通道时钟配置只能在通道安装时完成。

- 更新后，rmt_set_rx_idle_thresh 和 rmt_get_rx_idle_thresh 已被删除。新驱动中，接收通道的空闲状态阈值定义为*rmt_receive_config_t::signal_range_max_ns*。
- 更新后，rmt_set_mem_block_num 和 rmt_get_mem_block_num 已被删除。新驱动中，内存块的数量由*rmt_tx_channel_config_t::mem_block_symbols*与*rmt_rx_channel_config_t::mem_block_symbols*决定。
- 更新后，rmt_set_tx_carrier 已被删除。新驱动使用*rmt_apply_carrier()*来设置载波动作。
- 更新后，rmt_set_mem_pd 和 rmt_get_mem_pd 已被删除，驱动程序自动调整内存的功率。
- 更新后，rmt_memory_rw_rst，rmt_tx_memory_reset 和 rmt_rx_memory_reset 已被删除，驱动程序自动进行内存重置。
- 更新后，rmt_tx_start 和 rmt_rx_start 被合并为函数*rmt_enable()*，该函数同时适用于发射通道和接收通道。
- 更新后，rmt_tx_stop 和 rmt_rx_stop 被合并为函数*rmt_disable()*，该函数同时适用于发射通道和接收通道。
- 更新后，rmt_set_memory_owner 和 rmt_get_memory_owner 已被删除，驱动程序自动添加 RMT 内存保护。
- 更新后，rmt_set_tx_loop_mode 和 rmt_get_tx_loop_mode 已被删除。新驱动中，在*rmt_transmit_config_t::loop_count*中设置循环模式。
- 更新后，rmt_set_source_clk 和 rmt_get_source_clk 已被删除。仅能在通道安装时通过*rmt_tx_channel_config_t::clk_src*和*rmt_rx_channel_config_t::clk_src*设置时钟源。
- 更新后，rmt_set_rx_filter 已被删除。新驱动中，过滤阈值定义为*rmt_receive_config_t::signal_range_min_ns*。
- 更新后，rmt_set_idle_level 和 rmt_get_idle_level 已被删除，可在*rmt_transmit_config_t::eot_level*中设置发射通道的空闲状态电平。
- 更新后，rmt_set_rx_intr_en，rmt_set_err_intr_en，rmt_set_tx_intr_en，rmt_set_tx_thr_intr_en 和 rmt_set_rx_thr_intr_en 已被删除。新驱动不允许用户在用户端开启/关闭中断，而是提供了回调函数。
- 更新后，rmt_set_gpio 和 rmt_set_pin 已被删除。新驱动不支持运行时动态切换 GPIO 管脚。
- 更新后，rmt_config 已被删除。新驱动中，基础配置在通道安装阶段完成。
- 更新后，rmt_isr_register 和 rmt_isr_deregister 已被删除，驱动程序负责分配中断。
- rmt_driver_install 更新为*rmt_new_tx_channel()*与*rmt_new_rx_channel()*。
- rmt_driver_uninstall 更新为*rmt_del_channel()*。
- 更新后，rmt_fill_tx_items，rmt_write_items 和 rmt_write_sample 已被删除。新驱动中，用户需要提供一个编码器用来将用户数据“翻译”为 RMT 符号。
- 更新后，由于用户可以通过*rmt_tx_channel_config_t::resolution_hz*配置通道的时钟分辨率，rmt_get_counter_clock 已被删除。
- rmt_wait_tx_done 更新为*rmt_tx_wait_all_done()*。
- 更新后，rmt_translator_init，rmt_translator_set_context 和 rmt_translator_get_context 已被删除。新驱动中，翻译器更新为 RMT 译码器。
- 更新后，rmt_get_ringbuf_handle 已被删除。新驱动程序不再使用 Ringbuffer 来保存 RMT 符号。输入数据会直接保存到用户提供的缓冲区中，这些缓冲区甚至可以直接被挂载到 DMA 链接内部。
- rmt_register_tx_end_callback 更新为*rmt_tx_register_event_callbacks()*，用户可以在这个参数里面注册事件回调函数*rmt_tx_event_callbacks_t::on_trans_done*。
- 更新后，rmt_set_intr_enable_mask 和 rmt_clr_intr_enable_mask 已被删除。由于驱动程序负责处理中断，因此用户无需进行处理。
- rmt_add_channel_to_group 和 rmt_remove_channel_from_group 更新为 RMT 同步管理器，详见*rmt_new_sync_manager()*。
- 更新后，rmt_set_tx_loop_count 已被删除。新驱动中，循环计数在*rmt_transmit_config_t::loop_count*进行配置。
- 更新后，rmt_enable_tx_loop_autostop 已被删除。新驱动中，发射循环自动终止一直使能，用户无法进行配置。

LCD

- LCD 面板的初始化流程也有一些更新。更新后，*esp_lcd_panel_init()*不再会自动打开显示器。用户需要调用*esp_lcd_panel_disp_on_off()*来手动打开显示器。请注意，打开显示器

与打开背光是不同的。更新后，打开屏幕前，用户可以烧录一个预定义的图案，这可以避免开机复位后屏幕上的随机噪音。
- 更新后，*esp_lcd_panel_disp_off()* 已被弃用，请使用*esp_lcd_panel_disp_on_off()* 作为替代。
- 更新后，dc_as_cmd_phase 已被删除，SPI LCD 驱动不再支持 9-bit 的 SPI LCD。请使用专用的 GPIO 管脚来控制 LCD D/C 线。
- 更新后，用于注册 RGB 面板的事件回调函数已从 esp_lcd_rgb_panel_config_t 更新为单独的 API esp_lcd_rgb_panel_register_event_callbacks()。但是，事件回调签名仍保持不变。
- 更新后，esp_lcd_rgb_panel_config_t 中的标志位 relax_on_idle 被重命名为 esp_lcd_rgb_panel_config_t::refresh_on_demand，后者虽表达了同样的含义，但是其命名更有意义。
- 更新后，如果创建 RGB LCD 时，标志位 refresh_on_demand 使能，驱动不会在*esp_lcd_panel_draw_bitmap()* 中进行刷新，用户需要调用 esp_lcd_rgb_panel_refresh() 来刷新屏幕。
- 更新后，esp_lcd_color_space_t 已被弃用，请使用*lcd_color_space_t* 来描述色彩空间，使用*lcd_rgb_element_order_t* 来描述 RGB 颜色的排列顺序。

MCPWM MCPWM 驱动已更新（详见*MCPWM*）。同时，旧版驱动已被弃用。

新驱动中，每个 MCPWM 子模块相互独立，用户可以自由进行资源连接。

尽管我们推荐使用新的驱动 API，旧版驱动仍然可用，其引用路径为 driver/mcpwm.h。但是，使用旧版驱动会默认触发如下编译警告，可以通过配置 Kconfig 选项*CONFIG_MCPWM_SUPPRESS_DEPRECATE_WARN* 来关闭该警告。

```
legacy MCPWM driver is deprecated, please migrate to the new driver (include␣
↪driver/mcpwm_prelude.h)
```

主要概念和使用方法上的更新如下所示：

主要概念更新 更新后，MCPWM 驱动是面向对象的，大多数 MCPWM 子模块都有一个与之相关的驱动对象。驱动对象是由工厂函数创建的，如*mcpwm_new_timer()*。IO 控制函数总是需要对象句柄。

旧版驱动有一个不恰当的假设，即 MCPWM 运算器应连接到不同的 MCPWM 定时器上。但是，硬件上并没有这样的限制。新驱动中，同一个 MCPWM 定时器可以连接多个运算器，这样运算器可以获得最佳的同步性能。

更新前，驱动将生成 PWM 波形的方法预设为所谓的 mcpwm_duty_type_t，但是，列出的占空比模式远远不够。类似的，旧版驱动有一些预设的 mcpwm_deadtime_type_t 也没有包含所有的使用场景。更重要的是，用户通常会被占空比模式和死区时间模式的名称所迷惑。更新后，驱动没有这些限制，但是用户必须从头开始构建发生器的行为。

在旧版驱动中，通过 GPIO 管脚，软件和其他定时器模块同步 MCPWM 定时器的方法并不统一。这增加了用户的学习成本，因此新驱动统一了同步 API。

旧版驱动混淆了“故障检测器”和“故障处理器”的概念。这让用户对 API 感到非常困惑。新驱动中，故障对象只代表一个故障源，而且我们引入了一个新概念，**制动器**，来表示故障处理器。而且，新驱动支持软件故障。

旧版驱动只为获取子模块提供了回调函数，而新驱动为 MCPWM 子模块提供多种回调函数，如停止定时器，比较匹配，故障进入，紧急停止等。

- 更新后，不再使用 mcpwm_io_signals_t 和 mcpwm_pin_config_t，GPIO 管脚配置被移至子模块的配置结构中。
- 更新后，不再使用 mcpwm_timer_t 和 mcpwm_generator_t，定时器和发生器分别用*mcpwm_timer_handle_t* 和:cpp:type:*mcpwm_gen_handle_t* 表示。
- 更新后，不再使用 mcpwm_fault_signal_t 和 mcpwm_sync_signal_t，故障和同步源分别用*mcpwm_fault_handle_t* 和:cpp:type:*mcpwm_sync_handle_t* 表示。
- 更新后，不再使用 mcpwm_capture_signal_t，获取通道用*mcpwm_cap_channel_handle_t* 表示。

主要使用方法更新

- mcpwm_gpio_init 和 mcpwm_set_pin：GPIO 管脚配置在子模块配置中完成，例如在*mcpwm_generator_config_t::gen_gpio_num* 中设置 PWM GPIO 管脚。
- mcpwm_init：为得到预期的 PWM 波形，用户需要至少分配一个 MCPWM 定时器和 MCPWM 运算器，然后通过调用*mcpwm_operator_connect_timer()* 将二者连接起来。然后，用户需要调用如:cpp:func:*mcpwm_generator_set_actions_on_timer_event*，*mcpwm_generator_set_actions_on_compare_event()* 来设置发生器对不同事件的动作。
- mcpwm_group_set_resolution：新驱动中，群组分辨率固定为最大值，通常为 80 MHz。
- mcpwm_timer_set_resolution：MCPWM 定时器的分辨率在*mcpwm_timer_config_t::resolution_hz* 中进行设置。
- mcpwm_set_frequency：PWM 频率由*mcpwm_timer_config_t::resolution_hz*，*mcpwm_timer_config_t::count_mode* 和:cpp:member:*mcpwm_timer_config_t::period_ticks* 决定。
- mcpwm_set_duty：为设置 PWM 占空比，用户应调用*mcpwm_comparator_set_compare_value()* 来改变比较器的阈值。
- mcpwm_set_duty_type：新驱动中没有预设的占空比模式，通过设置不同的发生器行为，如*mcpwm_generator_set_actions_on_timer_event()*，来配置占空比模式。
- mcpwm_set_signal_high 和 mcpwm_set_signal_low 更新为*mcpwm_generator_set_force_level()*。新驱动中，这是通过为发生器设置力作用来实现的，而不是在后台将占空比改为 0% 或 100%。
- mcpwm_start 和 mcpwm_stop 更新为*mcpwm_timer_start_stop()*。用户可以用更多的模式来启动和停止 MCPWM 定时器，详见*mcpwm_timer_start_stop_cmd_t*。
- mcpwm_carrier_init 更新为*mcpwm_operator_apply_carrier()*。
- mcpwm_carrier_enable 与 mcpwm_carrier_disable：通过检查载波设置结构*mcpwm_carrier_config_t* 是否为空集来自动使能和停用载波子模块。
- mcpwm_carrier_set_period 更新为*mcpwm_carrier_config_t::frequency_hz*。
- mcpwm_carrier_set_duty_cycle 更新为*mcpwm_carrier_config_t::duty_cycle*。
- mcpwm_carrier_oneshot_mode_enable 更新为*mcpwm_carrier_config_t::first_pulse_duration_*
- 更新后，mcpwm_carrier_oneshot_mode_disable 被删除。硬件不支持停用第一个载波脉冲（即一次性脉冲）。
- mcpwm_carrier_output_invert 更新为*mcpwm_carrier_config_t::invert_before_modulate* 和*mcpwm_carrier_config_t::invert_after_modulate*。
- mcpwm_deadtime_enable 与 mcpwm_deadtime_disable 更新为*mcpwm_generator_set_dead_time()*。
- mcpwm_fault_init 更新为*mcpwm_new_gpio_fault()*。
- mcpwm_fault_set_oneshot_mode 与 mcpwm_fault_set_cyc_mode 更新为*mcpwm_operator_set_brake_on_fault()* 与*mcpwm_generator_set_actions_on_brake_event()*。
- 由于 mcpwm_capture_enable 与 mcpwm_capture_enable_channel() 重复，因此在更新后被删除。
- 由于 mcpwm_capture_disable 与 mcpwm_capture_capture_disable_channel() 重复，因此在更新后被删除。
- mcpwm_capture_enable_channel 与 mcpwm_capture_disable_channel 更新为*mcpwm_capture_channel_enable()* 与*mcpwm_capture_channel_disable()*。
- mcpwm_capture_signal_get_value 与 mcpwm_capture_signal_get_edge：通过*mcpwm_capture_event_data_t*，获取事件回调函数中提供了计时器的数值和边缘电平。只有获取事件发生时，获取数据才有意义，提供单一的 API 来获取捕获数据是没有意义的。
- 由于 mcpwm_sync_enable 与 mcpwm_sync_configure() 重复，因此更新后被删除。
- mcpwm_sync_configure 更新为*mcpwm_timer_set_phase_on_sync()*。
- mcpwm_sync_disable 相当于将*mcpwm_timer_sync_phase_config_t::sync_src* 设置为 NULL。
- mcpwm_set_timer_sync_output 更新为*mcpwm_new_timer_sync_src()*。
- mcpwm_timer_trigger_soft_sync 更新为*mcpwm_soft_sync_activate()*。
- mcpwm_sync_invert_gpio_synchro 与设置*mcpwm_gpio_sync_src_config_t::active_neg* 功能相同。
- 更新后，mcpwm_isr_register 已被删除。用户可以注册不同的事件回调函数来替代其功能，例如，可以使用*mcpwm_capture_channel_register_event_callbacks()* 注册获取事件注册函数。

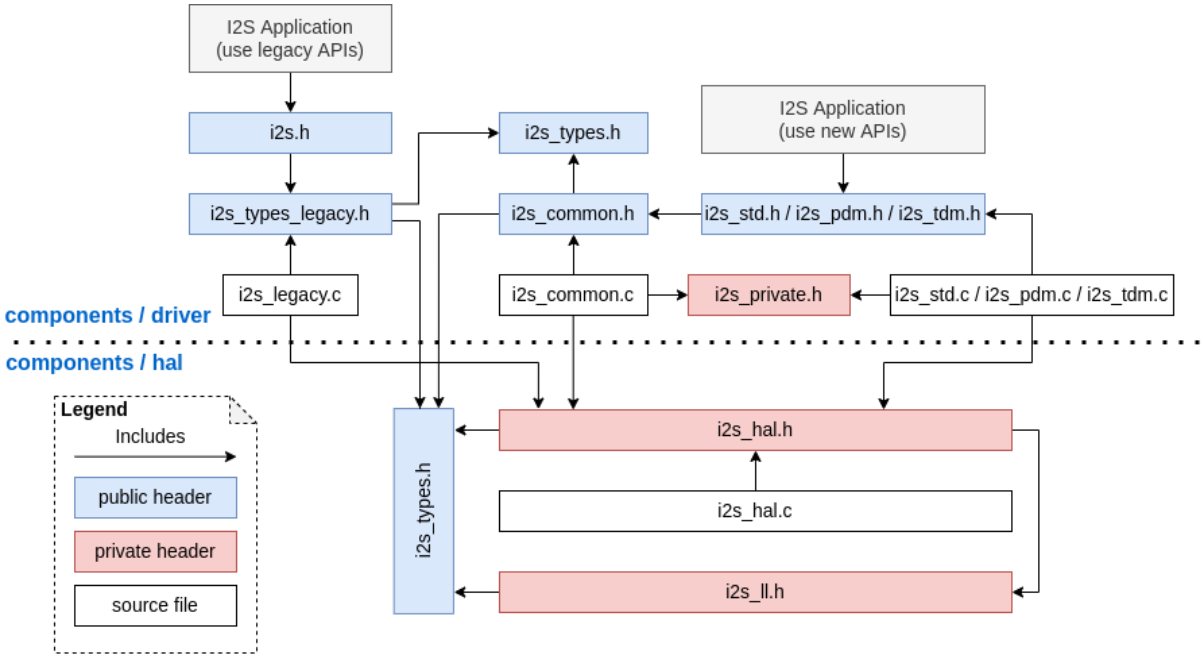
专用的 GPIO 驱动

- 更新后，所有与专用 GPIO 管脚相关的底层 (LL) 函数从 `cpu_ll.h` 中被移至 `dedic_gpio_cpu_ll.h`，并重新命名。

I2S 驱动 旧版 I2S 驱动在支持 ESP32-C3 和 ESP32-S3 新功能时暴露了很多缺点，为解决这些缺点，I2S 驱动已更新（请参考:doc:*I2S Driver <../../../api-reference/peripherals/i2s>*）。用户可以通过引用不同 I2S 模式对应的头文件来使用新版驱动的 API，如 esp_driver_i2s/include/driver/i2s_std.h，esp_driver_i2s/include/driver/i2s_pdm.h 以及 esp_driver_i2s/include/driver/i2s_tdm.h。

为保证前向兼容，旧版驱动的 API 仍然在 driver/deprecated/driver/i2s.h 中可用。但使用旧版 API 会触发编译警告，该警告可通过配置 Kconfig 选项*CONFIG_I2S_SUPPRESS_DEPRECATE_WARN* 来关闭。

以下是更新后的 I2S 文件概况。



主要概念更新

独立的发送通道和接收通道 更新后，I2S 驱动的最小控制单元是发送/接收通道，而不是整个 I2S 控制器（控制器包括多个通道）。

- 用户可以分别控制同一个 I2S 控制器的发送通道和接收通道，即可以通过配置实现分别开启和关闭发送通道和接收通道。
- `i2s_chan_handle_t` 句柄类型用于唯一地识别 I2S 通道。所有的 API 都需要该通道句柄，用户需要对这些通道句柄进行维护。
- 对于 ESP32-C3 和 ESP32-S3，同一个控制器中的发送通道和接收通道可以配置为不同的时钟或不同的模式。
- 但是对于 ESP32 和 ESP32-S2，同一个控制器中的发送通道和接收通道共享某些硬件资源。因此，配置可能会造成一个通道影响同一个控制器中的另一个通道。
- 通过将`i2s_port_t::I2S_NUM_AUTO` 设置为 I2S 端口 ID，驱动会搜索可用的发送/接收通道，之后通道会被自动注册到可用的 I2S 控制器上。但是，驱动仍然支持将通道注册到一个特定的端口上。
- 为区分发送/接收通道和声音通道，在更新后的驱动中，“通道 (channel)”一词仅代表发送/接收通道，用“声道 (slot)”来表示声音通道。

I2S 模式分类 I2S 通信模式包括以下三种模式，请注意：

- **标准模式**：标准模式通常包括两个声道，支持 Philips，MSB 和 PCM（短帧同步）格式，详见 esp_driver_i2s/include/driver/i2s_std.h。
- **PDM 模式**：PDM 模式仅支持两个声道，16 bit 数据位宽，但是 PDM TX 和 PDM RX 的配置略有不同。对于 PDM TX，采样率可通过 `i2s_pdm_tx_clk_config_t::sample_rate` 进行设置，其时钟频率取决于上采样的配置。对于 PDM RX，采样率可通过 `i2s_pdm_rx_clk_config_t::sample_rate` 进行设置，其时钟频率取决于下采样的配置，详见 esp_driver_i2s/include/driver/i2s_pdm.h。
- **TDM 模式**：TDM 模式可支持高达 16 声道，该模式可工作在 Philips，MSB，PCM（短帧同步）和 PCM（长帧同步）格式下，详见 esp_driver_i2s/include/driver/i2s_tdm.h。

在某个模式下分配新通道时，必须通过相应的函数初始化这个通道。我们强烈建议使用辅助宏来生成默认配置，以避免默认值被改动。

独立的声道配置和时钟配置 可以单独进行声道配置和时钟配置。

- 通过调用 *i2s_channel_init_std_mode()*，*i2s_channel_init_pdm_rx_mode()*，*i2s_channel_init_pdm_tx_mode()* 或:cpp:func:*i2s_channel_init_tdm_mode* 初始化声道/时钟/GPIO 管脚配置。
- 通过调用 *i2s_channel_reconfig_std_slot()*，*i2s_channel_reconfig_pdm_rx_slot()*，*i2s_channel_reconfig_pdm_tx_slot()* 或 *i2s_channel_reconfig_tdm_slot()* 可以在初始化之后改变声道配置。
- 通过调用 *i2s_channel_reconfig_std_clock()*，*i2s_channel_reconfig_pdm_rx_clock()*，*i2s_channel_reconfig_pdm_tx_clock()* 或 *i2s_channel_reconfig_tdm_clock()* 可以在初始化之后改变时钟配置。
- 通过调用 *i2s_channel_reconfig_std_gpio()*，*i2s_channel_reconfig_pdm_rx_gpio()*，*i2s_channel_reconfig_pdm_tx_gpio()* 或 *i2s_channel_reconfig_tdm_gpio()* 可以在初始化之后改变 GPIO 管脚配置。

Misc

- 更新后，I2S 驱动利用状态和状态机避免在错误状态下调用 API。
- 更新后，ADC 和 DAC 模式已被删除，只有它们各自专用的驱动及 I2S 旧版驱动还支持这两种模式。

主要使用方法更新 请参考以下步骤使用更新后的 I2S 驱动：

1. 通过调用 *i2s_new_channel()* 来获取通道句柄。我们应该在此步骤中指定外设为主机还是从机以及 I2S 端口。此外，驱动负责生成发送通道或接收通道的句柄。不需要同时输入发送通道和接收通道句柄，但需要输入至少一个句柄。输入两个句柄时，驱动会工作在双工模式。在同一端口上，发送通道和接收通道同时可用，并且共享 MCLK，BCLK 和 WS 信号。如果只输入了发送通道句柄或接收通道句柄，该通道只能工作在单工模式。
2. 通过调用 `i2s_channel_init_std_mode()`，`i2s_channel_init_pdm_rx_mode()`，`i2s_channel_init_pdm_tx_mode()` 或 `i2s_channel_init_tdm_mode()` 将通道初始化为指定模式。进行相应的声道、时钟和 GPIO 管脚配置。
3. （可选）通过调用 *i2s_channel_register_event_callback()* 注册 ISR 事件回调函数。I2S 事件由回调函数同步接收，而不是从事件队列中异步接收。
4. 通过调用 *i2s_channel_enable()* 来开启 I2S 通道的硬件资源。在更新后的驱动中，I2S 在安装后不会再自动开启，用户需要确定通道是否已经开启。
5. 分别通过 *i2s_channel_read()* 和 *i2s_channel_write()* 来读取和写入数据。当然，在 *i2s_channel_read()* 中只能输入接收通道句柄，在 *i2s_channel_write()* 中只能输入发送通道句柄。
6. （可选）通过相应的'reconfig' 函数可以更改声道、时钟和 GPIO 管脚配置，但是更改配置前必须调用 *i2s_channel_disable()*。
7. 通过调用 *i2s_channel_disable()* 可以停止使用 I2S 通道的硬件资源。
8. 不再使用某通道时，通过调用 *i2s_del_channel()* 可以删除和释放该通道资源，但是删除之前必须先停用该通道。

TWAI 驱动程序 CAN 外设驱动程序已弃用并被删除，请使用 TWAI 驱动程序代替（即在应用程序中包括 driver/twai.h）。

用于访问寄存器的宏 更新前，所有用于访问寄存器的宏都可以作为表达式来使用，所以以下命令是允许的:

```
uint32_t val = REG_SET_BITS(reg, bits, mask);
```

在 ESP-IDF v5.0 中，用于写入或读取-修改-写入寄存器的宏不能再作为表达式使用，而只能作为语句使用，这适用于以下宏：REG_WRITE，REG_SET_BIT，REG_CLR_BIT，REG_SET_BITS，REG_SET_FIELD，WRITE_PERI_REG，CLEAR_PERI_REG_MASK，SET_PERI_REG_MASK，SET_PERI_REG_BITS。

为存储要写入寄存器的值，请按以下步骤完成操作:

```
uint32_t new_val = REG_READ(reg) | mask;
REG_WRITE(reg, new_val);
```

要获得修改后的寄存器的值（该值可能与写入的值不同），要增加一个显示的读取命令:

```
REG_SET_BITS(reg, bits, mask);
uint32_t new_val = REG_READ(reg);
```

协议

Mbed TLS 在 ESP-IDF v5.0 版本中，Mbed TLS 已从 v2.x 版本更新到 v3.1.0 版本。

更多有关 Mbed TLS 从 v2.x 版本迁移到 v3.0 或更高版本的详细信息，请参考 官方指南。

重大更新（概述）

增加私有结构体字段数量

- 不再支持直接访问公共头文件中声明的结构体（struct 类型）字段。
- 当前版本下，访问公共头文件中声明的结构体字段需要使用特定的访问函数 (getter/setter)。另外，也可以用 MBEDTLS_PRIVATE 宏暂时代替，但不建议使用此种方法。
- 更多详细信息，请参考 官方指南。

SSL

- 不再支持 TLS 1.0、TLS 1.1 和 DTLS 1.0
- 不再支持 SSL 3.0

移除密码模块中的废弃函数

- 更新了与 MD、SHA、RIPEMD、RNG、HMAC 模块相关的函数 mbedtls_*_ret() 的返回值，并将其重新命名，以取代未附加 _ret 的相应函数。
- 更多详细信息，请参考 官方指南。

废弃配置选项 下列为在此次更新中废弃的重要配置选项。与以下配置有关或是依赖于下列配置的相关配置也已相应废弃。

- MBEDTLS_SSL_PROTO_SSL3：原用于支持 SSL 3.0
- MBEDTLS_SSL_PROTO_TLS1：原用于支持 TLS 1.0
- MBEDTLS_SSL_PROTO_TLS1_1：原用于支持 TLS 1.1
- MBEDTLS_SSL_PROTO_DTLS：原用于支持 DTLS 1.1（当前版本仅支持 DTLS 1.2）
- MBEDTLS_DES_C：原用于支持 3DES 密码套件
- MBEDTLS_RC4_MODE：原用于支持基于 RC4 的密码套件

备注： 上述仅列出了可通过 idf.py menuconfig 配置的主要选项。更多有关废弃选项的信息，请参考 官方指南。

其他更新

禁用 Diffie-Hellman 密码交换模式 为避免 安全风险，当前版本已默认禁用 Diffie-Hellman 密码交换模式。以下为相应的禁用配置项：

- MBEDTLS_DHM_C：原用于支持 Diffie-Hellman-Merkle 模块
- MBEDTLS_KEY_EXCHANGE_DHE_PSK：原用于支持 Diffie-Hellman 预共享密钥 (PSK) TLS 认证模式
- MBEDTLS_KEY_EXCHANGE_DHE_RSA：原用于支持带有前缀的密码套件 TLS-DHE-RSA-WITH-

---

备注： 在信号交换的初始步骤（即 client_hello）中，服务器会在客户端提供的列表中选择一个密码。由于 DHE_PSK/DHE_RSA 密码已在本次更新中禁用，服务器将退回到一个替代密码。在极个别情况中，服务器不支持任何其他的代码，此时，初始步骤将失败。若要检索服务器所支持的密码列表，需要首先在客户端使用特定的密码连接服务器，可以使用 sslscan 等工具完成连接。

从 X509 库中移除 certs 模块

- mbedtls 3.1 不再支持 mbedtls/certs.h 头文件。大多数应用程序支持从包含列表中安全删除该头文件。

对 esp_crt_bundle_set API 的重大更新

- 更新后，调用 *esp_crt_bundle_set()* API 需要一个额外的参数 bundle_size。该 API 的返回类型也从 void 变为了 *esp_err_t*。

对 esp_ds_rsa_sign API 的重大更新

- 更新后，调用 esp_ds_rsa_sign() API 无需再使用参数 mode。

HTTPS 服务器

重大更新（概述） 更新 *httpd_ssl_config_t* 结构体中持有不同证书的变量名。

- *httpd_ssl_config::servercert*：原 cacert_pem
- *httpd_ssl_config::servercert_len*：原 cacert_len
- *httpd_ssl_config::cacert_pem*：原 client_verify_cert_pem
- *httpd_ssl_config::cacert_len*：原 client_verify_cert_len

httpd_ssl_stop() API 的返回类型从 void 变为了 *esp_err_t*。

ESP HTTPS OTA

重大更新（概述）

- 函数*esp_https_ota()* 现需以指向*esp_https_ota_config_t* 的指针作为参数，而非之前的指向*esp_http_client_config_t* 的指针。

ESP-TLS

重大更新（概述）

私有化 esp_tls_t 结构体 更新后，*esp_tls_t* 已完全私有化，用户无法直接访问其内部结构。之前需要通过 ESP-TLS 句柄获得的必要数据，现在可由对应的 getter/setter 函数获取。如需特定功能的 getter/setter 函数，请在 ESP-IDF 的 Issue 板块 提出。

下列为新增的 getter/setter 函数：

- *esp_tls_get_ssl_context()*：从 ESP-TLS 句柄获取底层 ssl 栈的 ssl 上下文。

废弃函数及推荐的替代函数 下表总结了在 ESP-IDF v5.0 中废弃的函数以及相应的替代函数。

废弃函数	替代函数
esp_tls_conn_new()	*esp_tls_conn_new_sync()*
esp_tls_conn_delete()	*esp_tls_conn_destroy()*

- 函数*esp_tls_conn_http_new()* 现已废弃。请使用替代函数*esp_tls_conn_http_new_sync()*（或其异步函数*esp_tls_conn_http_new_async()*）。请注意，使用替代函数时，需要额外的参数*esp_tls_t*，此参数必须首先通过*esp_tls_init()* 函数进行初始化。

HTTP 服务器

重大更新（概述）

- esp_http_server 现不再支持 http_server.h 头文件。请使用 esp_http_server.h。

ESP HTTP 客户端

重大更新（概述）

- 函数*esp_http_client_read()* 和*esp_http_client_fetch_headers()* 现在会返回额外的返回值 -ESP_ERR_HTTP_EAGAIN 用于处理超时错误，即数据准备好前就已调用超时的情况。

TCP 传输

重大更新（概述）

- 更新后，出现连接超时的情况时，函数 esp_transport_read() 将返回 0，对其他错误则返回 < 0。请参考 esp_tcp_transport_err_t，查看所有可能的返回值。

MQTT 客户端

重大更新（概述）

- *esp_mqtt_client_config_t* 的所有字段都分组存放在子结构体中。

以下为较为常用的配置选项：

- 通过 esp_mqtt_client_config_t::broker::address::uri 配置 MQTT Broker
- 通过 esp_mqtt_client_config_t::broker::verification 配置 MQTT Broker 身份验证的相关安全问题
- 通过 esp_mqtt_client_config_t::credentials::username 配置客户端用户名
- *esp_mqtt_client_config_t* 不再支持 user_context 字段。之后注册事件处理程序，请使用*esp_mqtt_client_register_event()*；最后一个参数 event_handler_arg 可用于将用户上下文传递给处理程序。

ESP-Modbus

重大更新（概述） 本次更新从 ESP-IDF 中移除了组件 freemodbus，该组件已作为一个独立组件受到支持。可前往如下的独立仓库，查看更多有关 ESP-Modbus 的信息：

- GitHub 中的 ESP-Modbus 组件

在新版应用程序中，main 组件文件夹应包括组件管理器清单文件 idf_component.yml，如下所示：

```
dependencies:
  espressif/esp-modbus:
    version: "^1.0"
```

可以前往 组件管理器注册表 找到 ESP-Modbus 组件。更多有关如何设置组件管理器的信息，请参考 组件管理器文档。

对于使用 ESP-IDF v4.x 及以后版本的应用程序，需要通过添加组件管理器清单文件 idf_component.yml 拉取新版 ESP-Modbus 组件。同时，在编译时，应去掉已过时的 freemodbus 组件。此项操作可通过项目 CMakeLists.txt 中的以下语句实现：

```
set(EXCLUDE_COMPONENTS freemodbus)
```

配置

Protocomm *protocomm_set_security()* API 中的 pop 字段现已弃用。请使用 sec_params 字段来代替 pop。此参数应包含所使用的协议版本所要求的结构（包括安全参数）。

例如，当使用安全版本 2 时，sec_params 参数应包含指向*protocomm_security2_params_t* 类型结构的指针。

Wi-Fi 配置

- wifi_prov_mgr_start_provisioning() API 中的 pop 字段现已弃用。为了向后兼容，在使用安全版本 1 时，pop 仍可以作为字符串传递。但在使用安全版本 2 时，请使用 wifi_prov_sec_params 字段来代替 pop。此参数应包含所使用的协议版本所要求的结构（包括安全参数）。例如，当使用安全版本 2 时，wifi_prov_sec_params 参数应包含指向 wifi_prov_security2_params_t 结构体类型的指针。对于安全版本 1，该 API 的行为和使用方式保持不变。
- wifi_prov_mgr_is_provisioned() API 不再返回*ESP_ERR_INVALID_STATE* 错误。此 API 现在可以在不依赖配置管理器初始化状态的情况下工作。

ESP 本地控制 *esp_local_ctrl_proto_sec_cfg_t* API 中的 pop 字段现已弃用。请使用 sec_params 字段来代替 pop。此参数应包含所使用的协议版本所要求的结构（包括安全参数）。

例如，当使用安全版本 2 时，sec_params 字段应包含指向*esp_local_ctrl_security2_params_t* 类型结构的指针。

从 ESP-IDF 中移出或弃用的组件

移至 ESP-IDF Component Registry 的组件 以下组件已经从 ESP-IDF 中迁出至 ESP-IDF Component Registry：

- libsodium
- cbor
- jsmn
- esp_modem
- nghttp
- mdns
- esp_websocket_client
- asio
- freemodbus
- sh2lib
- expat
- coap
- esp-cryptoauthlib
- qrcode
- tjpgd
- esp_serial_slave_link
- tinyusb

备注： 请注意，http 解析功能以前属于 nghttp 组件一部分，但现在属于 http_parser 组件。

可使用 idf.py add-dependency 命令安装以上组件。

例如，要安装 X.Y 版本的 libsodium 组件，请运行：idf.py add-dependency libsodium==X.Y。

根据 semver 规则安装与 X.Y 兼容的最新版本 libsodium 组件，请运行 idf.py add-dependency libsodium~X.Y。

可前往 https://components.espressif.com 查询每个组件有哪些版本，按名称搜索该组件，组件页面上会列出所有版本。

弃用的组件 ESP-IDF v4.x 版本中已不再使用以下组件，这些组件已弃用：

- tcpip_adapter。可使用*ESP-NETIF* 组件代替，具体可参考*TCP/IP 适配器*。

备注： 不再支持 OpenSSL-API 组件。ESP-IDF Component Registry 中也没有该组件。请直接使用*ESP-TLS* 或 mbedtls API。

备注： 不再支持 esp_adc_cal 组件。新的 adc 校准驱动在 esp_adc 组件中。旧版 adc 校准驱动已被迁移进 esp_adc 组件中。要使用旧版 esp_adc_cal 驱动接口，你应该在 CMakeLists.txt 文件的组件依赖列表中增加 esp_adc。更多细节请查看*Peripherals Migration Guide*。

版本更新后无需目标组件，因此以下目标组件也已经从 ESP-IDF 中删除：

- esp32
- esp32s2
- esp32s3
- esp32c2
- esp32c3
- esp32h2

存储

分区 API 的新组件 非兼容性更新：所有的分区 API 代码都已迁移到新组件 esp_partition 中。如需查看所有受影响的函数及数据类型，请参见头文件 esp_partition.h 。

在以前，这些 API 函数和数据类型属于 spi_flash 组件。因此，在现有的应用程序中或将依赖 spi_flash，这也意味着在直接使用 esp_partition_* API/数据类型时，可能会导致构建过程失败（比如，在出现 #include "esp_partition.h" 的行中报错 fatal error: esp_partition.h: No such file or directory）。如果遇到类似问题，请按以下步骤更新项目中的 CMakeLists.txt 文件：

原有的依赖性设置：

```
idf_component_register(...
                       REQUIRES spi_flash)
```

更新后的依赖性设置：

```
idf_component_register(...
                       REQUIRES spi_flash esp_partition)
```

备注： 请根据项目的实际情况，更新相应的 REQUIRES 或是 PRIV_REQUIRES 部分。上述代码片段仅为范例。

如果问题仍未解决，请联系我们，我们将协助你进行代码迁移。

SDMMC/SDSPI 用户现可通过 sdmmc_host_t.max_freq_khz 将 SDMMC/SDSPI 接口上的 SD 卡频率配置为特定值，不再局限于之前的 SDMMC_FREQ_PROBING (400 kHz)、SDMMC_FREQ_DEFAULT (20 MHz) 或是 SDMMC_FREQ_HIGHSPEED (40 MHz)。此前，如果用户配置了上述三个给定频率之外的值，用户所选频率将自动调整为与其最为接近的给定值。

更新后，底层驱动将计算与用户配置的特定值最为接近的合适频率。相对于枚举项选择，该频率现由可用的分频器提供。不过，如果尚未更新现有的应用代码，可能会导致与 SD 卡的通信过程出现问题。如发现上述问题，请继续尝试配置与期望值接近的不同频率，直到找到合适的频率。如需查看底层驱动的计算结果以及实际应用的频率，请使用 void sdmmc_card_print_info(FILE* stream, const sdmmc_card_t* card) 函数。

FatFs FatFs 已更新至 v0.14，f_mkfs() 函数签名也已变更。新签名为 FRESULT f_mkfs (const TCHAR* path, const MKFS_PARM* opt, void* work, UINT len);，使用 MKFS_PARM 结构体作为第二个实参。

分区表 分区表生成器不再支持未对齐的分区。生成分区表时，ESP-IDF 将只接受偏移量与 4 KB 对齐的分区。此变更仅影响新生成的分区表，不影响读写现有分区。

VFS esp_vfs_semihost_register() 函数签名有所更改：

- 新签名为 esp_err_t esp_vfs_semihost_register(const char* base_path);
- 旧签名的 host_path 参数不再存在，请使用 OpenOCD 命令 ESP_SEMIHOST_BASEDIR 设置主机上的完整路径。

函数签名更改 以下函数现将返回 esp_err_t，而非 void 或 nvs_iterator_t。此前，当参数无效或内部出现问题时，这些函数将 assert() 或返回 nullptr。通过返回 esp_err_t，你将获得更加实用的错误报告。

- *nvs_entry_find()*
- *nvs_entry_next()*
- *nvs_entry_info()*

由于 esp_err_t 返回类型的更改，nvs_entry_find() 和 nvs_entry_next() 的使用模式也发生了变化。上述函数现均通过参数修改迭代器，而非返回一个迭代器。

迭代 NVS 分区的旧编程模式如下所示：

```
nvs_iterator_t it = nvs_entry_find(<nvs_partition_name>, <namespace>, NVS_TYPE_
↪ANY);
while (it != NULL) {
        nvs_entry_info_t info;
        nvs_entry_info(it, &info);
        it = nvs_entry_next(it);
        printf("key '%s', type '%d'", info.key, info.type);
};
```

现在，迭代 NVS 分区的编程模式已更新为：

```
nvs_iterator_t it = nullptr;
esp_err_t res = nvs_entry_find(<nvs_partition_name>, <namespace>, NVS_TYPE_ANY, &
↪it);
while(res == ESP_OK) {
    nvs_entry_info_t info;
    nvs_entry_info(it, &info); // Can omit error check if parameters are␣
↪guaranteed to be non-NULL
    printf("key '%s', type '%d'", info.key, info.type);
    res = nvs_entry_next(&it);
}
nvs_release_iterator(it);
```

迭代器有效性 请注意，由于函数签名的改动，如果存在参数错误，则可能从 nvs_entry_find() 获得无效迭代器。因此，请务必在使用 nvs_entry_find() 之前将迭代器初始化为 NULL，以免在调用 nvs_release_iterator() 之前进行复杂的错误检查。上述编程模式便是一个很好的例子。

删除 SDSPI 弃用的 API 结构体 sdspi_slot_config_t 和函数 sdspi_host_init_slot() 现已删除，并由结构体 sdspi_device_config_t 和函数 sdspi_host_init_device() 替代。

ROM SPI flash 在 v5.0 之前的版本中，ROM SPI flash 函数一般通过 esp32**/rom/spi_flash.h 得以体现。因此，为支持不同 ESP 芯片而编写的代码可能会填充不同目标的 ROM 头文件。此外，并非所有 API 都可以在全部的 ESP 芯片上使用。

现在，常用 API 已提取至 esp_rom_spiflash.h。尽管这不能算作重大变更，我们强烈建议仅使用此头文件中的函数（即以 esp_rom_spiflash 为前缀并包含在 esp_rom_spiflash.h 中），以获得不同 ESP 芯片之间更佳的交叉兼容性。

为了提高 ROM SPI flash API 的可读性，以下函数也进行了重命名：

- esp_rom_spiflash_lock() 更名为 esp_rom_spiflash_set_bp()
- esp_rom_spiflash_unlock() 更名为 esp_rom_spiflash_clear_bp()

SPI flash 驱动 esp_flash_speed_t enum 类型现已弃用。现在，可以直接将实际时钟频率值传递给 flash 配置结构。下为配置 80 MHz flash 频率的示例：

```
esp_flash_spi_device_config_t dev_cfg = {
    // Other members
    .freq_mhz = 80,
    // Other members
};
```

旧版 SPI flash 驱动 为了使 SPI flash 驱动更为稳定，v5.0 已经删除旧版 SPI flash 驱动。旧版 SPI flash 驱动程序是指自 v3.0 以来的默认 SPI flash 驱动程序，以及自 v4.0 以来启用配置选项 `CONFIG_SPI_FLASH_USE_LEGACY_IMPL` 的 SPI flash 驱动。从 v5.0 开始，我们将不再支持旧版 SPI flash 驱动程序。因此，旧版驱动 API 和 `CONFIG_SPI_FLASH_USE_LEGACY_IMPL` 配置选项均已删除，请改用新 SPI flash 驱动的 API。

删除项目	替代项目
`spi_flash_erase_sector()`	`esp_flash_erase_region()`
`spi_flash_erase_range()`	`esp_flash_erase_region()`
`spi_flash_write()`	`esp_flash_write()`
`spi_flash_read()`	`esp_flash_read()`
`spi_flash_write_encrypted()`	`esp_flash_write_encrypted()`
`spi_flash_read_encrypted()`	`esp_flash_read_encrypted()`

备注： 带有前缀 `esp_flash` 的新函数接受额外的 `esp_flash_t*` 参数。你可以直接将其设置为 NULL，从而使函数运行主 flash (`esp_flash_default_chip`)。

由于系统函数不再是公共函数，`esp_spi_flash.h` 头文件已停止使用。若要使用 flash 映射 API，请使用 `spi_flash_mmap.h`。

系统

跨核执行 跨核执行 (Inter-Processor Call, IPC) 不再是一个独立组件，现已包含至 `esp_system`。

因此，若项目的 `CMakeLists.txt` 文件中出现 `PRIV_REQUIRES esp_ipc` 或 `REQUIRES esp_ipc`，应删除这些选项，因为项目中已默认包含 `esp_system` 组件。

ESP 时钟 ESP 时钟 API（即以 `esp_clk` 为前缀的函数、类型或宏）已更新为私有 API。因此，原先的包含路径 `#include "ESP32-P4/clk.h"` 和 `#include "esp_clk.h"` 已移除。如仍需使用 ESP 时钟 API（并不推荐），请使用 `#include "esp_private/esp_clk.h"` 来包含。

注意： 私有 API 不属于稳定的 API，不会遵循 ESP-IDF 的版本演进规则，因此不推荐用户在应用中使用。

缓存错误中断 缓存错误中断 API（即以 `esp_cache_err` 为前缀的函数、类型或宏）已更新为私有 API。因此，原先的包含路径 `#include "ESP32-P4/cache_err_int.h"` 已移除。如仍需使用缓存错误中断 API（并不推荐），请使用 `#include "esp_private/cache_err_int.h"` 来包含。

bootloader_support

- 函数 `bootloader_common_get_reset_reason()` 已移除。请使用 ROM 组件中的函数 `esp_rom_get_reset_reason()`。

- 函数 `esp_secure_boot_verify_sbv2_signature_block()` 和 `esp_secure_boot_verify_rsa_signature_block()` 已移除，无新的替换函数。不推荐用户直接使用以上函数。如确需要，请在 GitHub 上对该功能提交请求，并解释需要此函数的原因。

断电 断电 API（即以 `esp_brownout` 为前缀的函数、类型或宏）已更新为私有 API。因此，原先的包含路径 `#include "brownout.h"` 已移除。如仍需使用断电 API（并不推荐），请使用 `#include "esp_private/brownout.h"` 来包含。

Trax Trax API（即以 `trax_` 为前缀的函数、类型或宏）已更新为私有 API。因此，原先的包含路径 `#include "trax.h"` 已移除。如仍需使用 Trax API（并不推荐），请使用 `#include "esp_private/trax.h"` 来包含。

ROM `components/esp32/rom/` 路径下存放的已弃用的 ROM 相关头文件已移除（原包含路径为 `rom/*.h`）。请使用新的特定目标的路径 `components/esp_rom/include/ESP32-P4/`（新的包含路径为 `ESP32-P4/rom/*.h`）。

esp_hw_support

- 头文件 `soc/cpu.h` 及弃用的 CPU util 函数都已移除。请包含 `esp_cpu.h` 来代替相同功能的函数。
- 头文件 `hal/cpu_ll.h`、`hal/cpu_hal.h`、`hal/soc_ll.h`、`hal/soc_hal.h` 和 `interrupt_controller_hal.h` 的 CPU API 函数已弃用。请包含 `esp_cpu.h` 来代替相同功能的函数。
- 头文件 `compare_set.h` 已移除。请使用 `esp_cpu.h` 中提供的 `esp_cpu_compare_and_set()` 函数来代替。
- `esp_cpu_get_ccount()`、`esp_cpu_set_ccount()` 和 `esp_cpu_in_ocd_debug_mode()` 已从 `esp_cpu.h` 中移除。请分别使用 `esp_cpu_get_cycle_count()`、`esp_cpu_set_cycle_count()` 和 `esp_cpu_dbgr_is_attached()` 代替。
- 头文件 `esp_intr.h` 已移除。请包含 `esp_intr_alloc.h` 以分配和操作中断。
- Panic API（即以 `esp_panic` 为前缀的函数、类型或宏）已更新为私有 API。因此，原先的包含路径 `#include "esp_panic.h"` 已移除。如仍需使用 Panic API（并不推荐），请使用 `#include "esp_private/panic_reason.h"` 来包含。此外，请包含 `esp_debug_helpers.h` 以使用与调试有关的任意辅助函数，如打印回溯。
- 头文件 `soc_log.h` 现更名为 `esp_hw_log.h`，并已更新为私有。建议用户使用 `esp_log.h` 头文件下的日志 API。
- 包含头文件 `spinlock.h`、`clk_ctrl_os.h` 和 `rtc_wdt.h` 时不应当使用 `soc` 前缀，如 `#include "spinlock.h"`。
- `esp_chip_info()` 命令返回芯片版本，格式为 = 100 * 主要 eFuse 版本 + 次要 eFuse 版本。因此，为适应新格式，`esp_chip_info_t` 结构体中的 `revision` 扩展为 `uint16_t`。

PSRAM

- 针对特定目标的头文件 `spiram.h` 及头文件 `esp_spiram.h` 已移除，创建新组件 `esp_psram`。对于与 PSRAM 或 SPIRAM 相关的函数，请包含 `esp_psram.h`，并在 CMakeLists.txt 项目文件中将 `esp_psram` 设置为必需组件。
- `esp_spiram_get_chip_size` 和 `esp_spiram_get_size` 已移除，请使用 `esp_psram_get_size`。

eFuse

- 函数 `esp_secure_boot_read_key_digests()` 的参数类型从 `ets_secure_boot_key_digests_t*` 更新为 `esp_secure_boot_key_digests_t*`。新类型与旧类型相同，仅移除了 `allow_key_revoke` 标志。在当前代码中，后者总是设置为 `true`，并未提供额外信息。

- 针对 eFuse 晶圆增加主要修订版本和次要修订版本。API `esp_efuse_get_chip_ver()` 与新修订不兼容，因此已移除。请使用 API `efuse_hal_get_major_chip_version()`、`efuse_hal_get_minor_chip_version()` 或 `efuse_hal_chip_revision()` 来代替原有 API。

esp_common `EXT_RAM_ATTR` 已弃用。请使用新的宏 `EXT_RAM_BSS_ATTR` 以将 `.bss` 放在 PSRAM 上。

esp_system

- 头文件 `esp_random.h`、`esp_mac.h` 和 `esp_chip_info.h` 以往都是通过头文件 `esp_system.h` 间接包含，更新后必须直接包含。已移除从 `esp_system.h` 中的间接包含功能。
- 回溯解析器 API（即以 `esp_eh_frame_` 为前缀的函数、类型或宏）已更新为私有 API。因此，原先的包含路径 `#include "eh_frame_parser.h"` 已移除。如仍需使用回溯解析器 API（并不推荐），请使用 `#include "esp_private/eh_frame_parser.h"` 来包含。
- 中断看门狗定时器 API（即以 `esp_int_wdt_` 为前缀的函数、类型或宏）已更新为私有 API。因此，原先的包含路径 `#include "esp_int_wdt.h"` 已移除。如仍需使用中断看门狗定时器 API（并不推荐），请使用 `#include "esp_private/esp_int_wdt.h"` 来包含。

SOC 依赖性

- Doxyfiles 中列出的公共 API 头文件中不会显示不稳定和非必要的 SOC 头文件，如 `soc/soc.h` 和 `soc/rtc.h`。这意味着，如果用户仍然需要这些“缺失”的头文件，就必须在代码中明确包含这些文件。
- Kconfig 选项 `LEGACY_INCLUDE_COMMON_HEADERS` 也已移除。
- 头文件 `soc/soc_memory_types.h` 已弃用。请使用 `esp_memory_utils.h`。包含 `soc/soc_memory_types.h` 将触发构建警告，如 `soc_memory_types.h is deprecated, please migrate to esp_memory_utils.h`。

应用跟踪 其中一个时间戳源已从定时器组驱动改为新的*GPTimer*。Kconfig 选项已重新命名，例如 `APPTRACE_SV_TS_SOURCE_TIMER00` 已更改为 `APPTRACE_SV_TS_SOURCE_GPTIMER`。用户已无需选择组和定时器 ID。

esp_timer 基于 FRC2 的 `esp_timer` 过去可用于 ESP32，现在已移除，更新后仅可使用更简单有效的 LAC 定时器。

ESP 镜像 ESP 镜像中关于 SPI 速度的枚举成员已重新更名：

- `ESP_IMAGE_SPI_SPEED_80M` 已重新命名为 `ESP_IMAGE_SPI_SPEED_DIV_1`。
- `ESP_IMAGE_SPI_SPEED_40M` 已重新命名为 `ESP_IMAGE_SPI_SPEED_DIV_2`。
- `ESP_IMAGE_SPI_SPEED_26M` 已重新命名为 `ESP_IMAGE_SPI_SPEED_DIV_3`。
- `ESP_IMAGE_SPI_SPEED_20M` 已重新命名为 `ESP_IMAGE_SPI_SPEED_DIV_4`。

任务看门狗定时器

- API `esp_task_wdt_init()` 更新后有如下变化：
  - 以结构体的形式传递配置。
  - 可将该函数配置为订阅空闲任务。
- 原先的配置选项 `CONFIG_ESP_TASK_WDT` 重新命名为*CONFIG_ESP_TASK_WDT_INIT* 并引入了一个新选项*CONFIG_ESP_TASK_WDT_EN*。

FreeRTOS

遗留 API 及数据类型 在以往版本中，ESP-IDF 默认设置 `configENABLE_BACKWARD_COMPATIBILITY` 选项，因此可使用 FreeRTOS v8.0.0 之前的函数名称和数据类型。该选项现在已默认禁用，因此默认情况下不再支持以往的 FreeRTOS 名称或类型。用户可以选择以下一种解决方式：

- 更新代码，删除以往的 FreeRTOS 名称或类型。
- 启用 *CONFIG_FREERTOS_ENABLE_BACKWARD_COMPATIBILITY* 以显式调用这些名称或类型。

任务快照 头文件 `task_snapshot.h` 已从 `freertos/task.h` 中移除。如需使用任务快照 API，请包含 `freertos/task_snapshot.h`。

函数 `vTaskGetSnapshot()` 现返回 `BaseType_t`，成功时返回值为 `pdTRUE`，失败则返回 `pdFALSE`。

FreeRTOS 断言 在以往版本中，FreeRTOS 断言通过 `FREERTOS_ASSERT` kconfig 选项独立配置，不同于系统的其他部分。该选项已移除，现在需要通过 `COMPILER_OPTIMIZATION_ASSERTION_LEVEL` 来完成配置。

FreeRTOS 移植相关的宏 已移除用以保证弃用 API 向后兼容性的 `portmacro_deprecated.h` 文件。建议使用下列函数来代替弃用 API。

- `portENTER_CRITICAL_NESTED()` 已移除，请使用 `portSET_INTERRUPT_MASK_FROM_ISR()` 宏。
- `portEXIT_CRITICAL_NESTED()` 已移除，请使用 `portCLEAR_INTERRUPT_MASK_FROM_ISR()` 宏。
- `vPortCPUInitializeMutex()` 已移除，请使用 `spinlock_initialize()` 函数。
- `vPortCPUAcquireMutex()` 已移除，请使用 `spinlock_acquire()` 函数。
- `vPortCPUAcquireMutexTimeout()` 已移除，请使用 `spinlock_acquire()` 函数。
- `vPortCPUReleaseMutex()` 已移除，请使用 `spinlock_release()` 函数。

应用程序更新

- 函数 *`esp_ota_get_app_description()`* 和 *`esp_ota_get_app_elf_sha256()`* 已弃用，请分别使用 *`esp_app_get_description()`* 和 *`esp_app_get_elf_sha256()`* 函数来代替。这些函数已移至新组件 esp_app_format。请参考头文件 esp_app_desc.h 。

引导加载程序支持

- *`esp_app_desc_t`* 结构体此前在 esp_app_format.h 中声明，现在在 esp_app_desc.h 中声明。
- 函数 `bootloader_common_get_partition_description()` 已更新为私有函数，请使用代替函数 *`esp_ota_get_partition_description()`*。注意，此函数的第一个参数为 *`esp_partition_t`*，而非 `esp_partition_pos_t`。

芯片版本 在应用程序开始加载时，引导加载程序会检查芯片版本。只有当版本为 >= *CONFIG_ESP32P4_REV_MIN* 和 < `CONFIG_ESP32P4_REV_MAX_FULL` 时，应用程序才能成功加载。

在 OTA 升级时，会检查应用程序头部中的版本需求和芯片版本是否符合条件。只有当版本为 >= *CONFIG_ESP32P4_REV_MIN* 和 < `CONFIG_ESP32P4_REV_MAX_FULL` 时，应用程序才能成功更新。

工具

ESP-IDF 监视器 ESP-IDF 监视器在波特率方面的改动如下：

- 目前，ESP-IDF 监视器默认遵循自定义的控制台波特率 (*CONFIG_ESP_CONSOLE_UART_BAUDRATE*)，而非 115200。
- ESP-IDF v5.0 不再支持通过 menuconfig 自定义波特率。
- 支持通过设置环境变量或在命令行中使用 `idf.py monitor -b <baud>` 命令自定义波特率。
- 注意，为了与全局波特率 `idf.py -b <baud>` 保持一致，波特率参数已从 `-B` 改名为 `-b`。请运行 `idf.py monitor --help` 获取更多信息。

废弃指令 ESP-IDF v5.0 已将 `idf.py` 子命令和 `cmake` 目标名中的下划线 (_) 统一为连字符 (-)。使用废弃的子命令及目标名将会触发警告，建议使用更新后的版本。具体改动如下：

表 1: 废弃子命令及目标名

废弃名	现用名
efuse_common_table	efuse-common-table
efuse_custom_table	efuse-custom-table
erase_flash	erase-flash
partition_table	partition-table
partition_table-flash	partition-table-flash
post_debug	post-debug
show_efuse_table	show-efuse-table
erase_otadata	erase-otadata
read_otadata	read-otadata

Esptool `CONFIG_ESPTOOLPY_FLASHSIZE_DETECT` 选项已重命名为*CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE*，且默认禁用。迁移到 ESP-IDF v5.0 的新项目和现有项目必须设置*CONFIG_ESPTOOLPY_FLASHSIZE*。若因编译时 flash 大小未知而无法设置，可启用*CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE*。但需注意的是，启用该项后，为在烧录期间使用 flash 大小更新二进制头时不会导致摘要无效，映像后将不再附加 SHA256 摘要。

Windows 环境 基于 MSYS/MinGW 的 Windows 环境支持已在 ESP-IDF v4.0 中弃用，v5.0 则完全移除了该项服务。请使用*ESP-IDF 工具安装器* 设置 Windows 兼容环境。目前支持 Windows 命令行、Power Shell 和基于 Eclipse IDE 的图形用户界面等选项。此外，还可以使用 支持的插件，设置基于 VSCode 的环境。

5.1.2 从 5.0 迁移到 5.1

GCC

GCC 版本 ESP-IDF 之前使用的 GCC 版本为 11.2.0，现已针对所有芯片目标升级至 GCC 12.2.0。若需要将代码从 GCC 11.2.0 迁移到 GCC 12.2.0，请参考以下 GCC 官方迁移指南。

- 迁移至 GCC 12

警告 升级至 GCC 12.2.0 后会触发新警告，或是导致原有警告内容发生变化。了解所有 GCC 警告的详细内容，请参考 GCC 警告选项。建议用户仔细检查代码，并尽量解决这些警告。但由于某些警告的特殊性及用户代码的复杂性，有些警告可能为误报，需要进行关键修复。在这种情况下，用户可以采取多种方式来抑制警告。本节介绍了用户可能遇到的常见警告及如何修复这些警告。

-Wuse-after-free 一般而言，此警告不会针对发布版本的代码产生误报，但是这种情况可能出现在测试用例中。以下示例为如何在 ESP-IDF 的 test_realloc.c 中修复该警告。

```
void *x = malloc(64);
void *y = realloc(x, 48);
TEST_ASSERT_EQUAL_PTR(x, y);
```

将指针转换为 int 可以避免出现 `-Wuse-after-free` 警告。

```
int x = (int) malloc(64);
int y = (int) realloc((void *) x, 48);
TEST_ASSERT_EQUAL_UINT32((uint32_t) x, (uint32_t) y);
```

-Waddress GCC 12.2.0 引入了增强版 `-Waddress` 警告选项，该选项对 if 语句中的数组指针检查更加敏感。

以下代码将触发警告：

```
char array[8];
...
if (array)
    memset(array, 0xff, sizeof(array));
```

删去不必要的检查可以消除警告。

```
char array[8];
...
memset(array, 0xff, sizeof(array));
```

在 ESP-IDF 框架之外构建 RISC-V RISC-V 的 `zicsr` 和 `zifencei` 扩展现已独立于 `I` 扩展，这一变化在 GCC 12 中也有所体现。因此，在 ESP-IDF 框架之外构建 RISC-V ESP32 芯片时，请在构建系统中指定 -march 选项时添加 `_zicsr_zifencei` 后缀。示例如下。

原为：

```
riscv32-esp-elf-gcc main.c -march=rv32imac
```

现为：

```
riscv32-esp-elf-gcc main.c -march=rv32imac_zicsr_zifencei
```

外设

GPSPI 不再支持以下函数。从 ESP-IDF v5.1 版本起，GPSPI 时钟源可配置。

- `spi_get_actual_clock` 已废弃，更新为*`spi_device_get_actual_freq()`*。

LEDC

- `soc_periph_ledc_clk_src_legacy_t::LEDC_USE_RTC8M_CLK` 已废弃，更新为 `LEDC_USE_RC_FAST_CLK`。

存储

FatFs `esp_vfs_fat_sdmmc_unmount()` 已弃用，可以使用*`esp_vfs_fat_sdcard_unmount()`* 代替。此接口在更早的 ESP-IDF 版本中已弃用，但是尚未添加弃用警告。自 ESP-IDF v5.1 起，调用 `esp_vfs_fat_sdmmc_unmount()` 接口将会产生弃用警告。

SPI_FLASH

- *`spi_flash_get_counters()`* 已弃用，请使用*`esp_flash_get_counters()`* 代替。
- *`spi_flash_dump_counters()`* 已弃用，请使用*`esp_flash_dump_counters()`* 代替。
- *`spi_flash_reset_counters()`* 已弃用，请使用*`esp_flash_reset_counters()`* 代替。

网络

SNTP SNTP 模块现在提供线程安全的 API 用于访问 lwIP 功能。建议使用*ESP_NETIF* API。了解更多信息，请参考章节*SNTP API*。

系统

FreeRTOS

动态内存分配 过去，FreeRTOS 通常使用 `malloc()` 函数来分配动态内存。因此，如果应用程序允许 `malloc()` 从外部 RAM 分配内存（通过将*CONFIG_SPIRAM_USE* 选项配置为 `CONFIG_SPIRAM_USE_MALLOC`），FreeRTOS 就有可能从外部 RAM 分配动态内存，并且具体位置由堆分配器确定。

备注： 任务的动态内存分配（通常占用大部分内存）与上述介绍的情况不同，是种例外情况。FreeRTOS 会使用单独的内存分配函数，确保为任务分配的动态内存始终位于内部 RAM 中。

允许将 FreeRTOS 对象（如队列和信号量）放置在外部 RAM 中可能会出现问题，例如如果在访问这些对象时 cache 被禁用（如在 SPI flash 写入操作期间），则会导致 cache 访问错误（详细信息请参阅*严重错误*）。

因此，FreeRTOS 已经更新为始终使用内部内存（即 DRAM）进行动态内存分配。调用 FreeRTOS 创建函数（例如*`xTaskCreate()`* 或 `xQueueCreate()`）将保证为这些任务/对象分配的内存来自内部内存（详细信息请参阅*FreeRTOS 堆*）。

警告： 如果你之前使用*CONFIG_SPIRAM_USE* 将 FreeRTOS 对象放置在外部内存中，这个更改则会导致内部内存的使用增加，因为现在 FreeRTOS 对象将被分配到内部内存中。

现在将 FreeRTOS 任务/对象放置在外部内存中，需要显示地设置，可采用以下方法之一：

- 使用 `...CreateWithCaps()` API 函数如*`xTaskCreateWithCaps()`* 或*`xQueueCreateWithCaps()`* 分配任务/对象到外部内存（详情请参阅*IDF 附加 API*）。
- 使用*`heap_caps_malloc()`* 为 FreeRTOS 对象手动分配外部内存，然后使用 `...CreateStatic()` FreeRTOS 函数从分配的内存中创建对象。

电源管理

- `esp_pm_config_esp32xx_t` 已弃用，应使用 `esp_pm_config_t` 替代。
- `esp32xx/pm.h` 已弃用，应使用 `esp_pm.h` 替代。

5.1.3 从 5.1 迁移到 5.2

GCC

GCC 版本 ESP-IDF 之前使用的 GCC 版本为 12.2.0，现已针对所有芯片目标升级至 GCC 13.2.0。若需要将代码从 GCC 12.2.0 迁移到 GCC 13.2.0，请参考以下 GCC 官方迁移指南。

- 迁移至 GCC 13

常见迁移问题和解决方法

stdio.h 不再包含 sys/types.h

问题描述 使用旧工具链的代码可能会出现编译错误，例如：

```
#include <stdio.h>
clock_t var; // error: expected specifier-qualifier-list before 'clock_t'
```

解决方法 使用正确的头文件可以解决这一问题。请按照以下方式重构代码：

```
#include <time.h>
clock_t var;
```

外设

UART

- *UART_FIFO_LEN* 已弃用，更新为*UART_HW_FIFO_LEN*。

I2C I2C 驱动已经被重新设计在*I2C*，以统一接口并扩展 I2C 外设的使用。尽管我们推荐使用新驱动的接口，但用户仍然可以通过包含路径 `driver/i2c.h` 来使用老驱动。

主要的概念上和用法上的改变如下所示：

主要概念更新

- 用于初始化整个 I2C 总线的结构体 `i2c_config_t` 已经被移除. 在新驱动中，从机和主机是分开的. 用户可以独立调用*i2c_master_bus_config_t* 和*i2c_slave_config_t*。
- `i2c_mode_t` 用来判断 I2C 控制器是工作在从模式还是主模式。该枚举器已被弃用。在新驱动程序中，用户无需自行设置，驱动程序会妥善处理。
- `i2c_rw_t` 用来判断 I2C 主控制器是在进行" 写" 还是" 读"。现在，它已被弃用。
- `i2c_addr_mode_t` 被重命名为 `i2c_addr_bit_len_t`.
- 在老驱动中，你需要将 I2C 命令通过 `I2C_LINK_RECOMMENDED_SIZE`, `i2c_cmd_link_create_static`, 等链接成链表. 在新驱动中，你不需要这么做，你只需要调用相关的接口函数即可。
- 在老驱动中，选择时钟通过例如 `I2C_SCLK_SRC_FLAG_FOR_NOMAL` 的时钟标志位. 在新驱动中，你可以直接选择时钟源。

主要用法更新

- I2C 总线初始化通过两部分完成. 第一步通过*i2c_new_master_bus()* 初始化 I2C 总线, 然后初始化 I2C 设备通过*i2c_master_bus_add_device()*。
- i2c_reset_tx_fifo 和 i2c_reset_rx_fifo 已被删除，因为用户不太需要重置 fifo。但可以通过 i2c_master_bus_reset 重置整个总线。
- 删除了 i2c_cmd_link_xxx 函数，用户不需要自己使用 link 来链接命令。
- i2c_master_write_to_device 更名为 i2c_master_transmit。
- i2c_master_read_from_device 已更名为 i2c_master_receive。
- i2c_master_write_read_device 已更名为 i2c_master_transmit_receive。
- i2c_slave_write_buffer 重命名为 i2c_slave_transmit。
- i2c_slave_read_buffer 已更名为 i2c_slave_receive。

协议

CoAP CoAP 相关示例已迁移至 idf-extra-components 仓库.

HTTP2 http2_request 相关示例已迁移至 idf-extra-components 仓库.

存储

NVS 加密

- 在集成 HMAC 外设 (SOC_HMAC_SUPPORTED) 的 SoC 上，启用*flash 加密* 时将不再自动启用*NVS 加密*。
- 因此需显式启用 NVS 加密，并按照需要选择基于 flash 加密或基于 HMAC 外设的方案。即使未启用 flash 加密，也可选择基于 HMAC 外设的方案 (*CONFIG_NVS_SEC_KEY_PROTECTION_SCHEME*)。
- 启用 flash 加密后，未集成 HMAC 外设的 SoC 仍会自动启用 NVS 加密。

系统

FreeRTOS

升级 IDF FreeRTOS IDF FreeRTOS 内核（通过 FreeRTOS 实现的双核 SMP ）已升级，现基于 Vanilla FreeRTOS v10.5.1。伴随升级，IDF FreeRTOS 的设计和实现也发生了重大变化。因此，用户应注意以下关于内核行为和 API 的变化：

- 通过*CONFIG_FREERTOS_UNICORE* 选项启用单核模式时，内核行为将与 Vanilla FreeRTOS 完全相同（详见单核模式）。
- 对于由 IDF FreeRTOS 添加的与 SMP 相关的 API，xCoreID 参数的检查将更加严格。为 xCoreID 参数提供超出范围的值现在将触发断言。
- 为了保证命名一致性，以下与 SMP 相关的 API 现已被弃用并已重新更名：
  - 弃用 xTaskGetAffinity()，更名为*xTaskGetCoreID()*。
  - 弃用 xTaskGetIdleTaskHandleForCPU(),更名为 xTaskGetIdleTaskHandleForCore()。
  - 弃用 xTaskGetCurrentTaskHandleForCPU(),更名为*xTaskGetCurrentTaskHandleForCore()*。

任务快照 由于缺乏从用户代码中使用该 API 的实际方法（必须先暂停调度器才能调用该 API），任务快照 API 已被设置为私有。

紧急处理程序行为 在配置选项*CONFIG_ESP_SYSTEM_PANIC* 中，是否可选择 `CONFIG_ESP_SYSTEM_PANIC_GDBSTUB` 取决于是否在构建中包含了 `esp_gdbstub` 组件。使用 `set(COMPONENTS main)` 来缩减这类构建的组件列表时，必须将 `esp_gdbstub` 组件添加到这个组件列表中，以使 `CONFIG_ESP_SYSTEM_PANIC_GDBSTUB` 选项可用。

5.1.4 从 5.2 迁移到 5.3

经典蓝牙

Bluedroid

以下 Bluedroid API 已被废弃：

- /bt/host/bluedroid/api/include/api/esp_bt_device.h
  - 废弃 `esp_err_t esp_bt_dev_set_device_name(const char *name)`
    - 设置设备名 API 已被替换为 `esp_err_t esp_bt_gap_set_device_name(const char *name)` 或 `esp_err_t esp_ble_gap_set_device_name(const char *name)`。原来的函数已经被废弃。
  - 废弃 `esp_err_t esp_bt_dev_get_device_name(void)`
    - 获取设备名 API 已被替换为 `esp_err_t esp_bt_gap_get_device_name(void)` 或 `esp_err_t esp_ble_gap_get_device_name(void)`。原来的函数已经被废弃。

GCC

常见的移植问题及解决方法

sys/dirent.h 不再包含一些函数原型

问题 使用旧工具链的代码可能会出现编译错误。例如：

```
#include <sys/dirent.h>
/* .... */
DIR* dir = opendir("test_dir");
/* .... */
/**
 * Compile error:
 * test.c: In function 'test_opendir':
 * test.c:100:16: error: implicit declaration of function 'opendir' [-
↪Werror=implicit-function-declaration]
 *   100 |     DIR* dir = opendir(path);
 *       |                ^~~~~~~
 */
```

解决方法 包含正确的头文件即可解决此问题。请将代码重构如下：

```
#include <dirent.h>
/* .... */
DIR* dir = opendir("test_dir");
```

外设

驱动程序 为了细粒度地控制其他组件对外设驱动的依赖，原先位于 `driver` 组件下的驱动程序被拆分到了各自独立的组件中。这些组件包括：

- *esp_driver_gptimer* - 通用定时器驱动
- *esp_driver_pcnt* - 脉冲计数器驱动
- *esp_driver_gpio* - GPIO 驱动
- *esp_driver_spi* - 通用 SPI 驱动
- *esp_driver_mcpwm* - 电机控制 PWM 驱动
- *esp_driver_sdmmc* - SDMMC 驱动
- *esp_driver_sdspi* - SDSPI 驱动
- *esp_driver_sdio* - SDIO 驱动
- *esp_driver_ana_cmpr* - 模拟比较器驱动
- *esp_driver_i2s* - I2S 驱动
- *esp_driver_dac* - DAC 驱动
- *esp_driver_rmt* - RMT 驱动
- *esp_driver_tsens* - 温度传感器驱动
- *esp_driver_sdm* - Sigma-Delta 调制器驱动
- *esp_driver_i2c* - I2C 驱动
- *esp_driver_uart* - UART 驱动
- *esp_driver_ledc* - LEDC 驱动
- *esp_driver_parlio* - 并行 IO 驱动
- *esp_driver_usb_serial_jtag* - USB_SERIAL_JTAG 驱动

为了兼容性，原来的 `driver` 组件仍然存在，并作为一个 “all-in-one” 的组件，将以上这些 *esp_driver_xyz* 组件注册成自己的公共依赖。换句话说，你无需修改既有项目的 CMake 文件，但是你现在多了一个途径去指定你项目依赖的具体的外设驱动。

原来你可能使用 **linker.lf** 指定了一些驱动函数在内存空间的链接位置，但是现在，因为驱动文件的位置发生了挪动，就需要你对 **linker.lf** 文件做出相应的改动的。假如，你的 linker.lf 文件里面有以下的条目：

```
[mapping:my_mapping_scheme]
archive: libdriver.a
entries:
    gpio (noflash)
```

现在需要修改成：

```
[mapping:my_mapping_scheme]
archive: libesp_driver_gpio.a
entries:
    gpio (noflash)
```

安全元素 ATECC608A 安全元素接口示例现已移至 GitHub 上的 esp-cryptoauthlib 仓库 中。

该示例也是组件管理器注册表中 esp-cryptoauthlib 的一部分。

I2S 回调事件 *i2s_event_data_t* 中指向 DMA 数组的二级指针 `data` 因使用过于繁琐已被弃用，请使用新增的指向 DMA 数组的一级指针 `dma_buf` 字段代替。

Protocols

ESP HTTPS OTA

Breaking Changes (Summary)

- If the image length is found in the HTTP header and *esp_https_ota_config_t::bulk_flash_erase* is set to true, then instead of erasing the entire flash, the erase operation will be performed to accommodate the size of the image length.

安全性

平台安全特性 当启用 flash 加密时，仅加密 app 分区中存在的应用程序镜像，而不是加密整个分区。这可以帮助优化在首次启动期间所需的加密时间。

这可以通过配置 `CONFIG_SECURE_FLASH_ENCRYPT_ONLY_IMAGE_LEN_IN_APP_PART` 来实现，默认情况下从 ESP-IDF v5.3 开始启用，而在所有早期版本中均禁用，以避免任何破坏性行为。

存储

VFS VFS 操作符的 UART 具体实现函数从 *vfs* 组件挪到了 *esp_driver_uart* 组件中。

所有以 *esp_vfs_dev_uart_* 前缀开头的 API 已被弃用，更新成在 *uart_vfs.h* 文件中定义的以 *uart_vfs_dev_* 为前缀的一组 API。具体来说，- `esp_vfs_dev_uart_register` 更名为 `uart_vfs_dev_register` - `esp_vfs_dev_uart_port_set_rx_line_endings` 更名为 `uart_vfs_dev_port_set_rx_line_endings` - `esp_vfs_dev_uart_port_set_tx_line_endings` 更名为 `uart_vfs_dev_port_set_tx_line_endings` - `esp_vfs_dev_uart_use_nonblocking` 更名为 `uart_vfs_dev_use_nonblocking` - `esp_vfs_dev_uart_use_driver` 更名为 `uart_vfs_dev_use_driver`

为了兼容性，*vfs* 组件依旧将 *esp_driver_uart* 注册成了其私有依赖。换句话说，你无需修改既有项目的 CMake 文件。

SPI flash 驱动 XMC-C 系列闪存 suspend 功能的支持已被移除。根据闪存制造商的反馈，在某些情况下，XMC-C 闪存需要在 resume 和下一条命令之间间隔 1 毫秒。这对于软件请求来说时间太长。基于上述原因，为了安全使用 suspend 功能，我们决定取消对 XMC-C 系列闪存挂起的支持。但是你依旧可以通过打开 *CONFIG_SPI_FLASH_FORCE_ENABLE_XMC_C_SUSPEND* 来强行使能这个功能。如果您有任何疑问，请联系 espressif 商务支持。

System

Power Management

- `esp_sleep_enable_ext1_wakeup_with_level_mask` is deprecated, use *esp_sleep_enable_ext1_wakeup_io()* and *esp_sleep_disable_ext1_wakeup_io()* instead.

Unit Testing In the past versions of Unity framework, it was possible to omit a semicolon at the end of a `TEST_ASSERT_*` macro statement. This is no longer the case in the newer version of Unity, used in IDF v5.3.

For example, the following code:

```
TEST_ASSERT(some_func() == ESP_OK)
```

will now result in a compilation error. To fix this, add a semicolon at the end of the statement:

```
TEST_ASSERT(some_func() == ESP_OK);
```

Partition Table Partition Table generation tool has been fixed to ensure that the size of partition of type `app` is having flash sector (minimum erase size) aligned size (please see *偏移地址 (Offset) 和大小 (Size) 字段*). If the partition does not have aligned size, partition table generator tool will raise an error. This fix ensures that OTA updates for a case where the file size is close or equal to the size of partition works correctly (erase operation does not go beyond the partition size).

In case you have the `app` partition size which is not a multiple of the 4 KB then please note that while migrating to ESP-IDF 5.3, you must align this size to its lower 4 KB boundary for the build to succeed. This does not impact the partition table for existing devices as such but ensures that generated firmware size remains within the OTA update capablilty limit.

Chapter 6

安全指南

6.1 概述

6.1.1 安全

本指南概述了乐鑫解决方案中可用的整体安全功能。从 **安全**角度考虑，强烈建议在使用乐鑫平台和 ESP-IDF 软件栈设计产品时参考本指南。

目标

高级安全目标包括：

1. 防止执行不受信任的代码
2. 保护存储在外部 flash 中代码的可信度和完整性
3. 保护设备身份
4. 安全存储机密数据
5. 设备身份验证与加密通信

平台安全

安全启动 安全启动功能可以确保设备仅执行通过身份认证的软件。安全启动时，会验证*应用程序的启动流程* 中所涉及的所有 **可变**软件实体，形成信任链。设备启动以及 OTA 更新过程中都会进行签名认证。

关于安全启动功能的更多详情，请参阅*Secure Boot V2*。

重要： 强烈建议在所有生产设备上启用安全启动功能。

安全启动最佳实践

- 在具备高质量熵源的系统上生成签名密钥。
- 签名密钥始终保密；签名密钥泄露会危及安全启动系统。
- 禁止第三方使用 `espsecure.py` 观察密钥生成或签名过程的相关细节，这两个过程都容易受到时序攻击或其他侧信道攻击的影响。
- 确保正确烧录所有安全性 eFuse，包括禁用调试接口以及非必需的启动介质（例如 UART 下载模式）等。

flash 加密 flash 加密功能可以加密外部 flash 中的内容，从而保护存储在 flash 中软件或数据的 **机密性**。

关于该功能的更多详情，请参阅*flash 加密*。

如果 ESP32-P4 连接了外部 SPI RAM，那么写入或读取到 SPI RAM 的内容将会分别进行加密和解密。当启用 flash 加密时，上述过程将通过 MMU 的 flash 缓存实现。以上加密和解密过程为存储在 SPI RAM 中的数据提供了额外的安全层，有助于安全地启用 `CONFIG_MBEDTLS_EXTERNAL_MEM_ALLOC` 等特定配置选项。

flash 加密最佳实践

- 建议在生产环境中使用 flash 加密的发布模式。
- 建议为每个设备生成唯一的 flash 加密密钥。
- 启用*安全启动* 作为额外保护层，防止 flash 在启动前遭受恶意攻击。

设备身份 在 ESP32-P4 中，数字签名外设借助硬件加速，通过 HMAC 算法生成 RSA 数字签名。RSA 私钥仅限设备硬件访问，软件无法获取，保证了设备上存储密钥的安全性。

DS 或 ECDSA 外设可以建立与远程终端之间的 **安全设备身份**，如基于 RSA 或 ECDSA 加密算法的 TLS 双向认证。

详情请参阅*数字签名 (DS)*。

内存保护 ESP32-P4 可以通过架构或 PMS 等特定外设实现 **内存保护**，强制执行和监控内存以及某些外设的权限属性。使用相应外设，ESP-IDF 应用程序启动代码可以配置数据内存的读取/写入权限以及指令内存的读取/执行权限。如有任何操作尝试违反这些权限属性，如写入指令内存区域，将触发违规中断，导致系统 panic。

使用该功能需启用配置选项*CONFIG_ESP_SYSTEM_MEMPROT_FEATURE*，该选项默认启用。请注意，该功能的 API 是 **私有**的，仅供 ESP-IDF 代码使用。

备注： 内存保护功能可以防止因软件漏洞导致的远程代码注入。

调试接口

JTAG

- 如果启用了任一安全功能，则 JTAG 接口将保持禁用。更多详情请参阅*JTAG 与 flash 加密和安全引导*。
- 如果不启用其他安全功能，也可以使用*eFuse API* 禁用 JTAG 接口。
- ESP32-P4 支持软禁用 JTAG 接口，并且可以通过 HMAC 烧录密钥重新启用，请参阅*HMAC 启用 JTAG 接口*。

UART 下载模式 ESP32-P4 中，如果启用了任一安全功能，则会激活安全 UART 下载模式。

- 要启用安全 UART 下载模式，也可以调用*`esp_efuse_enable_rom_secure_download_mode()`*。
- 该模式下，禁止执行通过 UART 下载模式下载的任意代码。
- 该模式将限制部分涉及更新 SPI 配置的命令，如更改波特率、基本的 flash 写入以及通过 `get_security_info` 返回当前启用的安全功能摘要。
- 要完全禁用安全 UART 下载模式，可以将*CONFIG_SECURE_UART_ROM_DL_MODE* 设置为建议选项 `Permanently disable ROM Download Mode`，或者在运行时调用*`esp_efuse_disable_rom_download_mode()`*。

重要： 安全 UART 下载模式下，仅支持使用 `--no-stub` 参数调用 `esptool.py`。

产品安全

安全 OTA 更新

- OTA 更新必须通过安全传输进行，如 HTTPS。
- ESP-IDF 为此提供了一个简化的抽象层，即*ESP HTTPS OTA 升级*。
- 如果启用了*安全启动*，则服务器应托管已签名的应用程序镜像。
- 如果启用了*flash 加密*，则服务器端不需要额外操作，在 flash 写入时，设备将自动加密。
- OTA 更新的*回滚过程* 可以在验证完应用程序的功能后，再将应用程序切换为 `active` 状态。

防回滚保护 防回滚保护功能确保设备仅执行特定版本的应用程序，即应满足设备 eFuse 存储的安全版本条件。因此，即使已由合法密钥信任和签名，应用程序可能包含已撤销的安全功能或凭据，因此设备必须拒绝执行此类应用程序。

ESP-IDF 仅支持在应用程序使用该功能，并通过二级引导加载程序管理。安全版本存储在设备 eFuse 中，并在启动时和 OTA 更新期间与应用程序镜像头进行比较。

关于启用此功能的更多详情，请参阅*防回滚*。

加密固件分发 OTA 更新期间，使用加密的固件分发，可以确保在从服务器 **传输**到设备的过程中，应用程序保持加密。OTA 更新期间，这可以作为在 TLS 通信之上的额外保护层，保护应用程序身份。

关于加密固件分发的工作示例，请参阅*使用预加密固件进行 OTA 升级*。

安全存储 安全存储指在设备上以安全方式存储应用程序的特定数据，即将数据存储在外部 flash 中。外部 flash 通常是可读写的 flash 分区，用于存储设备特定的配置数据，如 Wi-Fi 凭据。

ESP-IDF 提供了 **NVS（非易失性存储）**管理组件，允许加密数据分区。该功能与上文提到的*flash 加密* 平台功能相关。

关于该功能的工作原理和启用说明，请参阅*NVS 加密*。

重要： ESP-IDF 组件会默认将 Wi-Fi 证书等设备特定数据写入 NVS 默认分区，建议使用 **NVS 加密**功能来保护这些数据。

安全设备控制 ESP-IDF 提供了 ESP 本地控制组件，可以通过 `Wi-Fi/Ethernet + HTTP` 或 `BLE` 安全地控制 ESP 设备。

关于该功能的更多详情，请参阅*ESP 本地控制*。

安全策略

ESP-IDF GitHub 代码库内含 安全政策介绍。

公告

- 乐鑫会发布重要 安全公告 ，包括硬件和软件相关公告。
- ESP-IDF 软件组件的相关安全公告会发布在 GitHub 仓库 。

软件更新 ESP-IDF 会及时处理针对组件和第三方库的相关报告，并修复关键安全问题。修复内容会逐步同步到 ESP-IDF 的所有适用版本分支中。

ESP-IDF 的发布说明将涵盖各 ESP-IDF 组件和第三方库的相应安全问题和 CVE 编号。

重要： 为获取所有关键安全修复，建议定期更新到 ESP-IDF 的最新 Bugfix 版本。

6.2 功能

6.2.1 flash 加密

本文档旨在引导用户快速了解 ESP32-P4 的 flash 加密功能，通过应用程序代码示例向用户演示如何在开发及生产过程中测试及验证 flash 加密的相关操作。

概述

flash 加密功能用于加密与 ESP32-P4 搭载使用的片外 flash 中的内容。启用 flash 加密功能后，固件会以明文形式烧录，然后在首次启动时将数据进行加密。因此，物理读取 flash 将无法恢复大部分 flash 内容。

重要： 对于生产用途，flash 加密仅应在“发布”模式下启用。

重要： 启用 flash 加密将限制后续 ESP32-P4 更新。在使用 flash 加密功能前，请务必阅读本文档了解其影响。

加密分区

启用 flash 加密后，会默认加密以下类型的数据：

- 二级引导程序 （固件引导加载程序）
- 分区表
- *NVS* 密钥分区
- Otadata
- 所有 `app` 类型的分区

其他类型的数据将视情况进行加密：

- 分区表中标有 `encrypted` 标志的分区。如需了解详情，请参考加密分区标志。
- 如果启用了安全启动，则会对安全启动引导程序摘要进行加密（见下文）。

相关 eFuses

flash 加密操作由 ESP32-P4 上的多个 eFuse 控制。以下是这些 eFuse 列表及其描述，下表中的各 eFuse 名称也在 espefuse.py 工具中使用，为了能在 eFuse API 中使用，请在名称前加上 `ESP_EFUSE_`，如：esp_efuse_read_field_bit(ESP_EFUSE_DISABLE_DL_ENCRYPT)。

表 1: flash 加密过程中使用的 eFuses

eFuse	描述	位深
BLOCK_KEYN	AES 密钥存储，N 在 0-5 之间。	XTS_AES_128 有一个 256 位密钥块，XTS_AES_256 有两个 256 位密钥块（共 512 位）。
KEY_PURPOSE_N	控制 eFuse 块 BLOCK_KEYN 的目的，其中 N 在 0-5 之间。可能的值：2 代表 XTS_AES_256_KEY_1，3 代表 XTS_AES_256_KEY_2，4 代表 XTS_AES_128_KEY。最终 AES 密钥是基于其中一个或两个目的 eFuses 值推导。有关各种可能的组合，请参阅 *ESP32-P4 技术参考手册* > *外部内存加密和解密（XTS_AES)* [PDF]。	4
DIS_DOWNLOAD_MANUAL_ENCRY	设置后，在下载启动模式下禁用 flash 加密。	1
SPI_BOOT_CRYPT_CNT	设置 SPI 启动模式后，可启用加密和解密。如果在 eFuse 中设置了 1 或 3 个比特位，则启用该功能，否则将禁用。	3

备注:

- 上表中列出的所有 eFuse 位都提供读/写访问控制。
- 这些位的默认值是 0。

对上述 eFuse 位的读写访问由 WR_DIS 和 RD_DIS 寄存器中的相应字段控制。有关 ESP32-P4 eFuse 的详细信息，请参考*eFuse 管理器*。要使用 espefuse.py 更改 eFuse 字段的保护位，请使用以下两个命令：read_protect_efuse 和 write_protect_efuse。例如 `espefuse.py write_protect_efuse DISABLE_DL_ENCRYPT`。

flash 的加密过程

假设 eFuse 值处于默认状态，且固件的引导加载程序编译为支持 flash 加密，则 flash 加密的具体过程如下：

1. 第一次开机复位时，flash 中的所有数据都是未加密的（明文）。ROM 引导加载程序加载固件引导加载程序。
2. 固件的引导加载程序将读取 SPI_BOOT_CRYPT_CNT eFuse 值 (0b000)。因为该值为 0（偶数位），固件引导加载程序将配置并启用 flash 加密块。关于 flash 加密块的更多信息，请参考 *ESP32-P4 技术参考手册* > *eFuse 控制器 (eFuse)* > *自动加密块* [PDF]。
3. 固件引导加载程序首先检查 eFuse 中是否已经存在有效密钥（例如用 espefuse 工具烧写的密钥），如果存在，则会跳过密钥生成，并将该密钥用于 flash 加密过程。否则，固件引导加载程序使用 RNG（随机数发生器）模块生成一个 256 位或 512 位的密钥，具体位数取决于*生成的 XTS-AES 密钥的大小*，然后将其分别写入一个或两个 *BLOCK_KEYN* eFuse 中。软件也为存储密钥的块更新了 KEY_PURPOSE_N。由于上述一个或两个 BLOCK_KEYN eFuse 已设置了读保护和写保护位，因此无法通过软件访问密钥。KEY_PURPOSE_N 字段也受写保护。flash 加密操作完全在硬件中完成，无法通过软件访问密钥。
4. flash 加密块将加密 flash 的内容（固件的引导加载程序、应用程序、以及标有“加密”标志的分区）。就地加密可能会耗些时间（对于大分区最多需要一分钟）。
5. 固件引导加载程序将在 SPI_BOOT_CRYPT_CNT (0b001) 中设置第一个可用位来对已加密的 flash 内容进行标记。设置奇数位。
6. 对于*开发模式*，固件引导加载程序允许 UART 引导加载程序重新烧录加密后的二进制文件。同时，SPI_BOOT_CRYPT_CNT eFuse 位不受写入保护。此外，固件引导加载程序默认置位以下 eFuse 位：

- DIS_DOWNLOAD_ICACHE
- DIS_DOWNLOAD_DCACHE
- HARD_DIS_JTAG
- DIS_LEGACY_SPI_BOOT

7. 对于发布模式，固件引导加载程序设置所有在开发模式下设置的 eFuse 位。它还写保护 SPI_BOOT_CRYPT_CNT eFuse 位。要修改此行为，请参阅启用 *UART* 引导加载程序加密/解密。
8. 重新启动设备以开始执行加密镜像。固件引导加载程序调用 flash 解密块来解密 flash 内容，然后将解密的内容加载到 IRAM 中。

在开发阶段常需编写不同的明文 flash 镜像并测试 flash 的加密过程。这要求固件下载模式能够根据需求不断加载新的明文镜像。但是，在制造和生产过程中，出于安全考虑，固件下载模式不应有权限访问 flash 内容。

因此需要有两种不同的 flash 加密配置：一种用于开发，另一种用于生产。详情请参考*flash 加密设置* 小节。

flash 加密设置

提供以下 flash 加密模式：

- 开发模式 - 建议仅在开发过程中使用。因为在这种模式下，仍然可以将新的明文固件烧录到设备，并且引导加载程序将使用存储在硬件中的密钥对该固件进行透明加密。此操作间接允许从 flash 中读出固件明文。
- 发布模式 - 推荐用于制造和生产。因为在这种模式下，如果不知道加密密钥，则不可能将明文固件烧录到设备。

本节将详细介绍上述 flash 加密模式，并且逐步说明如何使用它们。

开发模式 在开发过程中，可使用 ESP32-P4 内部生成的密钥或外部主机生成的密钥进行 flash 加密。

使用 ESP32-P4 生成的密钥 开发模式允许用户使用固件下载模式下载多个明文镜像。

测试 flash 加密过程需完成以下步骤：

1. 确保你的 ESP32-P4 设备有相关 *eFuses* 中所示的 flash 加密 eFuse 的默认设置。

   请参考如何检查*ESP32-P4 flash 加密状态*。

2. 在项目配置菜单，执行以下操作：

   - 启动时使能 *flash* 加密。
   - 选择加密模式 （默认是 **开发模式**）。
   - 选择 *UART ROM* 下载模式 （默认是 **启用**）。
   - 设置生成的 *XTS-AES* 密钥大小。
   - 选择适当详细程度的引导加载程序日志。
   - 保存配置并退出。

启用 flash 加密将增大引导加载程序，因而可能需更新分区表偏移量。请参考引导加载程序大小。

3. 运行以下命令来构建和烧录完整的镜像。

```
idf.py flash monitor
```

备注： 这个命令不包括任何应该写入 flash 分区的用户文件。请在运行此命令前手动写入这些文件，否则在写入前应单独对这些文件进行加密。

该命令将向 flash 写入未加密的镜像：固件引导加载程序、分区表和应用程序。烧录完成后，ESP32-P4 将复位。在下一次启动时，固件引导加载程序会加密：固件引导加载程序、应用程序分区和标记为“加密”的分区，然后复位。就地加密可能需要时间，对于大分区最多需要一分钟。之后，应用程序在运行时解密并执行命令。

下面是启用 flash 加密后 ESP32-P4 首次启动时的样例输出：

```
rst:0x1 (POWERON),boot:0x30f (SPI_FAST_FLASH_BOOT)
SPI mode:DIO, clock div:1
load:0x4ff2abd0,len:0xa9c
load:0x4ff2cbd0,len:0x4898
load:0x4ff33e48,len:0x2c08
entry 0x4ff2abdc
I (29) boot: ESP-IDF v5.3-dev-3079-g96ed5ffadf 2nd stage bootloader
I (29) boot: compile time Apr  1 2024 16:35:58
I (29) boot: Multicore bootloader
I (35) boot: chip revision: v0.0
I (37) boot.esp32p4: SPI Speed      : 80MHz
I (42) boot.esp32p4: SPI Mode       : DIO
I (47) boot.esp32p4: SPI Flash Size : 2MB
I (52) boot: Enabling RNG early entropy source...
I (65) boot: Partition Table:
I (69) boot: ## Label            Usage          Type ST Offset   Length
I (76) boot:  0 nvs              WiFi data        01 02 0000e000 00006000
I (84) boot:  1 storage          Unknown data     01 ff 00014000 00001000
I (91) boot:  2 factory          factory app      00 00 00020000 00100000
I (98) boot:  3 nvs_key          NVS keys         01 04 00120000 00001000
I (106) boot:  4 custom_nvs       WiFi data        01 02 00121000 00006000
I (115) boot: End of partition table
I (118) esp_image: segment 0: paddr=00020020 vaddr=40020020 size=0abc8h ( 43976)␣
↪map
E (126) fpga_rng: Project configuration is for internal FPGA use, RNG will not work
I (145) esp_image: segment 1: paddr=0002abf0 vaddr=30100000 size=00044h (    68)␣
↪load
I (147) esp_image: segment 2: paddr=0002ac3c vaddr=4ff00000 size=053dch ( 21468)␣
↪load
I (158) esp_image: segment 3: paddr=00030020 vaddr=40000020 size=1a884h (108676)␣
↪map
I (185) esp_image: segment 4: paddr=0004a8ac vaddr=4ff053dc size=06dech ( 28140)␣
↪load
I (194) esp_image: segment 5: paddr=000516a0 vaddr=4ff0c1d0 size=013c0h (  5056)␣
↪load
I (199) boot: Loaded app from partition at offset 0x20000
I (200) boot: Checking flash encryption...
I (202) efuse: Batch mode of writing fields is enabled
I (208) flash_encrypt: Generating new flash encryption key...
E (214) fpga_rng: Project configuration is for internal FPGA use, RNG will not work
I (224) efuse: Writing EFUSE_BLK_KEY0 with purpose 4
W (228) flash_encrypt: Not disabling UART bootloader encryption
I (235) flash_encrypt: Disable UART bootloader cache...
I (241) flash_encrypt: Disable JTAG...
I (246) efuse: BURN BLOCK4
I (252) efuse: BURN BLOCK4 - OK (write block == read block)
I (255) efuse: BURN BLOCK0
I (260) efuse: BURN BLOCK0 - OK (write block == read block)
I (264) efuse: Batch mode. Prepared fields are committed
I (270) esp_image: segment 0: paddr=00002020 vaddr=4ff2abd0 size=00a9ch (  2716)
I (280) esp_image: segment 1: paddr=00002ac4 vaddr=4ff2cbd0 size=04898h ( 18584)
I (291) esp_image: segment 2: paddr=00007364 vaddr=4ff33e48 size=02c08h ( 11272)
I (840) flash_encrypt: bootloader encrypted successfully
I (904) flash_encrypt: partition table encrypted and loaded successfully
I (904) flash_encrypt: Encrypting partition 1 at offset 0x14000 (length 0x1000)...
I (982) flash_encrypt: Done encrypting
```

(下页继续)

(续上页)

```
I (982) esp_image: segment 0: paddr=00020020 vaddr=40020020 size=0abc8h ( 43976)␣
↪map
I (994) esp_image: segment 1: paddr=0002abf0 vaddr=30100000 size=00044h (    68)
I (996) esp_image: segment 2: paddr=0002ac3c vaddr=4ff00000 size=053dch ( 21468)
I (1006) esp_image: segment 3: paddr=00030020 vaddr=40000020 size=1a884h (108676)␣
↪map
I (1033) esp_image: segment 4: paddr=0004a8ac vaddr=4ff053dc size=06dech ( 28140)
I (1041) esp_image: segment 5: paddr=000516a0 vaddr=4ff0c1d0 size=013c0h (  5056)
I (1044) flash_encrypt: Encrypting partition 2 at offset 0x20000 (length 0x100000).
↪..
I (17573) flash_encrypt: Done encrypting
I (17573) flash_encrypt: Encrypting partition 3 at offset 0x120000 (length 0x1000).
↪..
I (17645) flash_encrypt: Done encrypting
I (17646) efuse: BURN BLOCK0
I (17648) efuse: BURN BLOCK0 - OK (all write block bits are set)
I (17649) flash_encrypt: Flash encryption completed
I (17655) boot: Resetting with flash encryption enabled...
```

启用 flash 加密后，在下次启动时输出将显示已启用 flash 加密，样例输出如下：

```
ESP-ROM:esp32p4-20230811
Build:Aug 11 2023
rst:0x3 (SW_SYS_RESET),boot:0x30f (SPI_FAST_FLASH_BOOT)
use legacy efuse key
SPI mode:DIO, clock div:1
load:0x4ff2abd0,len:0xa9c
load:0x4ff2cbd0,len:0x4898
load:0x4ff33e48,len:0x2c08
entry 0x4ff2abdc
I (27) boot: ESP-IDF v5.3-dev-3079-g96ed5ffadf 2nd stage bootloader
I (27) boot: compile time Apr  1 2024 16:35:58
I (28) boot: Multicore bootloader
I (33) boot: chip revision: v0.0
I (35) boot.esp32p4: SPI Speed      : 80MHz
I (40) boot.esp32p4: SPI Mode       : DIO
I (45) boot.esp32p4: SPI Flash Size : 2MB
I (50) boot: Enabling RNG early entropy source...
I (63) boot: Partition Table:
I (67) boot: ## Label            Usage          Type ST Offset   Length
I (74) boot:  0 nvs              WiFi data        01 02 0000e000 00006000
I (82) boot:  1 storage          Unknown data     01 ff 00014000 00001000
I (89) boot:  2 factory          factory app      00 00 00020000 00100000
I (97) boot:  3 nvs_key          NVS keys         01 04 00120000 00001000
I (104) boot:  4 custom_nvs       WiFi data        01 02 00121000 00006000
I (113) boot: End of partition table
I (116) esp_image: segment 0: paddr=00020020 vaddr=40020020 size=0abc8h ( 43976)␣
↪map
E (124) fpga_rng: Project configuration is for internal FPGA use, RNG will not work
I (144) esp_image: segment 1: paddr=0002abf0 vaddr=30100000 size=00044h (    68)␣
↪load
I (146) esp_image: segment 2: paddr=0002ac3c vaddr=4ff00000 size=053dch ( 21468)␣
↪load
I (156) esp_image: segment 3: paddr=00030020 vaddr=40000020 size=1a884h (108676)␣
↪map
I (183) esp_image: segment 4: paddr=0004a8ac vaddr=4ff053dc size=06dech ( 28140)␣
↪load
I (192) esp_image: segment 5: paddr=000516a0 vaddr=4ff0c1d0 size=013c0h (  5056)␣
↪load
I (198) boot: Loaded app from partition at offset 0x20000
I (198) boot: Checking flash encryption...
```

(下页继续)

(续上页)

```
I (200) flash_encrypt: flash encryption is enabled (1 plaintext flashes left)
I (208) boot: Disabling RNG early entropy source...
I (233) cpu_start: Multicore app
I (249) cpu_start: Pro cpu start user code
I (249) cpu_start: cpu freq: 360000000 Hz
I (250) app_init: Application information:
I (252) app_init: Project name:     flash_encryption
I (258) app_init: App version:      qa-test-test-dynamic-pipeline-f
I (265) app_init: Compile time:     Apr  1 2024 16:35:55
I (271) app_init: ELF file SHA256:  c71f148d8...
I (276) app_init: ESP-IDF:          v5.3-dev-3079-g96ed5ffadf
I (283) efuse_init: Min chip rev:     v0.0
I (287) efuse_init: Max chip rev:     v0.99
I (292) efuse_init: Chip rev:         v0.0
I (297) heap_init: Initializing. RAM available for dynamic allocation:
I (304) heap_init: At 4FF0ECD0 len 0002C2F0 (176 KiB): RAM
I (310) heap_init: At 4FF3AFC0 len 00004BE4 (18 KiB): RAM
I (316) heap_init: At 4FF40000 len 00060000 (384 KiB): RAM
I (323) heap_init: At 50108000 len 00007FE8 (31 KiB): RTCRAM
I (329) heap_init: At 30100044 len 00001FBC (7 KiB): TCM
I (336) spi_flash: detected chip: generic
I (340) spi_flash: flash io: dio
W (344) spi_flash: Detected size(4096k) larger than the size in the binary image␣
↪header(2048k). Using the size in the binary image header.
W (357) flash_encrypt: Flash encryption mode is DEVELOPMENT (not secure)
I (365) main_task: Started on CPU0
I (375) main_task: Calling app_main()

Example to check Flash Encryption status
This is esp32p4 chip with 2 CPU core(s), WiFi, silicon revision v0.0, 2MB external␣
↪flash
FLASH_CRYPT_CNT eFuse value is 1
Flash encryption feature is enabled in DEVELOPMENT mode
Erasing partition "storage" (0x1000 bytes)
Writing data with esp_partition_write:
I (455) example: 0x4ff112a0   00 01 02 03 04 05 06 07  08 09 0a 0b 0c 0d 0e 0f  |..
↪..............|
I (455) example: 0x4ff112b0   10 11 12 13 14 15 16 17  18 19 1a 1b 1c 1d 1e 1f  |..
↪..............|
Reading with esp_partition_read:
I (465) example: 0x4ff112c0   00 01 02 03 04 05 06 07  08 09 0a 0b 0c 0d 0e 0f  |..
↪..............|
I (475) example: 0x4ff112d0   10 11 12 13 14 15 16 17  18 19 1a 1b 1c 1d 1e 1f  |..
↪..............|
Reading with esp_flash_read:
I (485) example: 0x4ff112c0   8a 20 78 9a cc bd 90 3a  96 0b 50 25 09 3b df c9  |.␣
↪x....:..P%.;..|
I (495) example: 0x4ff112d0   d5 d7 0b fe af d2 cb 9a  3f e7 50 9f e7 b5 98 fc  |..
↪......?.P.....|
I (595) main_task: Returned from app_main()
```

在此阶段，如果用户需要更新或重新烧录二进制文件，请参考重新烧录更新后的分区。

使用主机生成的密钥 可在主机中预生成 flash 加密密钥，并将其烧录到 eFuse 密钥块中。这样，无需明文 flash 更新便可以在主机上预加密数据并将其烧录。该功能可在开发模式 和发布模式 两模式下使用。如果没有预生成的密钥，数据将以明文形式烧录，然后 ESP32-P4 对数据进行就地加密。

备注： 不建议在生产中使用该方法，除非为每个设备都单独生成一个密钥。

使用主机生成的密钥需完成以下步骤：

1. 确保你的 ESP32-P4 设备有相关 *eFuses* 中所示的 flash 加密 eFuse 的默认设置。

   请参考如何检查*ESP32-P4 flash 加密状态*。

2. 通过运行以下命令生成一个随机密钥：

   如果生成的 *XTS-AES* 密钥大小 是 AES-128（256 位密钥）：

   ```
   espsecure.py generate_flash_encryption_key my_flash_encryption_
   ↪key.bin
   ```

   如果生成的 *XTS-AES* 密钥大小 是 AES-256（512 位密钥）：

   ```
   espsecure.py generate_flash_encryption_key --keylen 512 my_flash_
   ↪encryption_key.bin
   ```

3. **在第一次加密启动前**，使用以下命令将该密钥烧录到设备上，这个操作只能执行 一**次**。

   ```
   espefuse.py --port PORT burn_key BLOCK my_flash_encryption_key.bin␣
   ↪KEYPURPOSE
   ```

   其中 BLOCK 是 BLOCK_KEY0 和 BLOCK_KEY5 之间的空闲密钥区。而 KEYPURPOSE 是 AES_256_KEY_1、XTS_AES_256_KEY_2 或 XTS_AES_128_KEY。关于密钥用途，请参考 ESP32-P4 技术参考手册。

   对于 AES-128（256 位密钥）- XTS_AES_128_KEY:

   ```
   espefuse.py --port PORT burn_key BLOCK my_flash_encryption_key.bin XTS_
   ↪AES_128_KEY
   ```

   对于 AES-256（512 位密钥）- XTS_AES_256_KEY_1 和 XTS_AES_256_KEY_2。espefuse.py 支持通过虚拟密钥用途 XTS_AES_256_KEY 将这两个密钥用途和一个 512 位密钥一起烧录到两个独立的密钥块。使用此功能时，espefuse.py 将把密钥的前 256 位烧录到指定的 BLOCK，并把相应的区块密钥用途烧录到 XTS_AES_256_KEY_1。密钥的后 256 位将被烧录到 BLOCK 后的第一个空闲密钥块，并把相应的密钥用途烧录到 XTS_AES_256_KEY_2。

   ```
   espefuse.py  --port PORT  burn_key BLOCK my_flash_encryption_key.bin XTS_
   ↪AES_256_KEY
   ```

   如果你想指定使用哪两个区块，则可以将密钥分成两个 256 位密钥，并分别使用 XTS_AES_256_KEY_1 和 XTS_AES_256_KEY_2 为密钥用途进行手动烧录：

   ```
   split -b 32 my_flash_encryption_key.bin my_flash_encryption_key.bin.
   espefuse.py  --port PORT  burn_key BLOCK my_flash_encryption_key.bin.aa␣
   ↪XTS_AES_256_KEY_1
   espefuse.py  --port PORT  burn_key BLOCK+1 my_flash_encryption_key.bin.ab␣
   ↪XTS_AES_256_KEY_2
   ```

   如果未烧录密钥并在启用 flash 加密后启动设备，ESP32-P4 将生成一个软件无法访问或修改的随机密钥。

4. 在项目配置菜单 中进行如下设置：
   - 启动时启用 *flash* 加密功能
   - 选择加密模式 （默认为 **开发模式**）
   - 选择适当详细程度的引导加载程序日志
   - 保存配置并退出

启用 flash 加密将增大引导加载程序，因而可能需更新分区表偏移量。请参考引导加载程序大小。

5. 运行以下命令来构建并烧录完整的镜像：

   ```
   idf.py flash monitor
   ```

备注： 这个命令不包括任何应该被写入 flash 上的分区的用户文件。请在运行此命令前手动写入这些文件，否则在写入前应单独对这些文件进行加密。

该命令将向 flash 写入未加密的镜像：固件引导加载程序、分区表和应用程序。烧录完成后，ESP32-P4 将复位。在下一次启动时，固件引导加载程序会加密：固件引导加载程序、应用程序分区和标记为 加密的分区，然后复位。就地加密可能需要时间，对于大的分区来说可能耗时一分钟。之后，应用程序在运行时被解密并执行。

如果使用开发模式，那么更新和重新烧录二进制文件最简单的方法是重新烧录更新后的分区。

如果使用发布模式，那么可以在主机上预先加密二进制文件，然后将其作为密文烧录。具体请参考手动加密文件。

重新烧录更新后的分区 如果用户以明文方式更新了应用程序代码并需要重新烧录，则需要在烧录前对其进行加密。请运行以下命令一次完成应用程序的加密与烧录：

```
idf.py encrypted-app-flash monitor
```

如果所有分区都需要以加密形式更新，请运行：

```
idf.py encrypted-flash monitor
```

发布模式 在发布模式下，UART 引导加载程序无法执行 flash 加密操作，**只能**使用 OTA 方案下载新的明文镜像，该方案将在写入 flash 前加密明文镜像。

使用该模式需要执行以下步骤：

1. 确保你的 ESP32-P4 设备有相关 *eFuses* 中所示的 flash 加密 eFuse 的默认设置。

   请参考如何检查*ESP32-P4 flash 加密状态*。

2. 在项目配置菜单，执行以下操作：

   - 启动时使能 *flash* 加密
   - 选择发布模式（注意一旦选择了发布模式，EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT eFuse 位将被编程为在 ROM 下载模式下禁用 flash 加密硬件。）
   - 选择 *UART ROM* 下载（推荐永久性的切换到安全模式）。这是默认且推荐使用的选项。如果不需要该模式，也可以改变此配置设置永久地禁用 UART ROM 下载模式。
   - 选择适当详细程度的引导加载程序日志
   - 保存配置并退出

   启用 flash 加密将增大引导加载程序，因而可能需更新分区表偏移量。请参考引导加载程序大小。

3. 运行以下命令来构建并烧录完整的镜像：

   ```
   idf.py flash monitor
   ```

备注： 这个命令不包括任何应该被写入 flash 分区的用户文件。请在运行此命令前手动写入这些文件，否则在写入前应单独对这些文件进行加密。

该命令将向 flash 写入未加密的镜像：固件引导加载程序、分区表和应用程序。烧录完成后，ESP32-P4 将复位。在下一次启动时，固件引导加载程序会加密：固件引导加载程序、应用程序分区和标记为 加密的分区，然后复位。就地加密可能需要时间，对于大的分区来说可能耗时一分钟。之后，应用程序在运行时被解密并执行。

一旦在发布模式下启用 flash 加密，引导加载程序将写保护 SPI_BOOT_CRYPT_CNT eFuse。

请使用*OTA 方案* 对字段中的明文进行后续更新。

备注: 如果用户已经预先生成了 flash 加密密钥并存储了一个副本，并且 UART 下载模式没有通过*CONFIG_SECURE_UART_ROM_DL_MODE* 永久禁用，那么可以通过使用 `espsecure.py encrypt_flash_data --aes_xts` 预加密文件，从而在在本地更新 flash，然后烧录密文。请参考手动加密文件。

最佳实践 在生产中使用 flash 加密时：

- 不要在多个设备之间重复使用同一个 flash 加密密钥，这样攻击者就无法从一台设备上复制加密数据后再将其转移到第二台设备上。
- 如果不需要 UART ROM 下载模式，则应完全禁用该模式，或者永久设置为“安全下载模式”。安全下载模式永久性地将可用的命令限制在更新 SPI 配置、更改波特率、基本的 flash 写入和使用 *get_security_info* 命令返回当前启用的安全功能摘要。默认在发布模式下第一次启动时设置为安全下载模式。要完全禁用下载模式，请选择*CONFIG_SECURE_UART_ROM_DL_MODE* 为“永久禁用 ROM 下载模式（推荐）”或在运行时调用`esp_efuse_disable_rom_download_mode()`。
- 启用安全启动 作为额外的保护层，防止攻击者在启动前有选择地破坏 flash 中某部分。

外部启用 flash 加密

在上述过程中，对与 flash 加密相关的 eFuse 是通过固件引导加载程序烧写的，或者，也可以借助 `espefuse` 工具烧写 eFuse。如需了解详情，请参考*Enable Flash Encryption Externally*。

可能出现的错误

一旦启用 flash 加密，`SPI_BOOT_CRYPT_CNT` 的 eFuse 值将设置为奇数位。这意味着所有标有加密标志的分区都会包含加密的密本。如果 ESP32-P4 错误地加载了明文数据，则会出现以下三种典型的错误情况：

1. 如果通过 **明文固件引导加载程序镜像**重新烧录了引导加载程序分区，则 ROM 加载器将无法加载固件引导加载程序，并会显示以下错误类型：

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
```

备注: 不同应用程序中无效头文件的值不同。

备注: 如果 flash 内容被擦除或损坏，也会出现这个错误。

2. 如果固件的引导加载程序已加密，但通过 **明文分区表镜像**重新烧录了分区表，引导加载程序将无法读取分区表，从而出现以下错误：

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
```

(下页继续)

(续上页)

```
load:0x3fff001c,len:10464
ho 0 tail 12 room 4
load:0x40078000,len:19168
load:0x40080400,len:6664
entry 0x40080764
I (60) boot: ESP-IDF v4.0-dev-763-g2c55fae6c-dirty 2nd stage bootloader
I (60) boot: compile time 19:15:54
I (62) boot: Enabling RNG early entropy source...
I (67) boot: SPI Speed      : 40MHz
I (72) boot: SPI Mode       : DIO
I (76) boot: SPI Flash Size : 4MB
E (80) flash_parts: partition 0 invalid magic number 0x94f6
E (86) boot: Failed to verify partition table
E (91) boot: load partition table error!
```

3. 如果引导加载程序和分区表已加密，但使用 **明文应用程序镜像** 重新烧录了应用程序，引导加载程序将无法加载应用程序，从而出现以下错误：

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8452
load:0x40078000,len:13616
load:0x40080400,len:6664
entry 0x40080764
I (56) boot: ESP-IDF v4.0-dev-850-gc4447462d-dirty 2nd stage bootloader
I (56) boot: compile time 15:37:14
I (58) boot: Enabling RNG early entropy source...
I (64) boot: SPI Speed      : 40MHz
I (68) boot: SPI Mode       : DIO
I (72) boot: SPI Flash Size : 4MB
I (76) boot: Partition Table:
I (79) boot: ## Label            Usage          Type ST Offset   Length
I (87) boot:  0 nvs              WiFi data        01 02 0000a000 00006000
I (94) boot:  1 phy_init         RF data          01 01 00010000 00001000
I (102) boot:  2 factory          factory app      00 00 00020000 00100000
I (109) boot: End of partition table
E (113) esp_image: image at 0x20000 has invalid magic byte
W (120) esp_image: image at 0x20000 has invalid SPI mode 108
W (126) esp_image: image at 0x20000 has invalid SPI size 11
E (132) boot: Factory app partition is not bootable
E (138) boot: No bootable app partitions in the partition table
```

ESP32-P4 flash 加密状态

1. 确保你的 ESP32-P4 设备有相关 *eFuses* 中所示的 flash 加密 eFuse 的默认设置。

要检查你的 ESP32-P4 设备上是否启用了 flash 加密，请执行以下操作之一：

- 将应用示例 security/flash_encryption 烧录到你的设备上。此应用程序会打印 `SPI_BOOT_CRYPT_CNT` eFuse 值，以及是否启用了 flash 加密。
- 查询设备所连接的串口名称，在以下命令中将 `PORT` 替换为串口名称后运行：

```
espefuse.py -p PORT summary
```

在加密的 flash 中读写数据

ESP32-P4 应用程序代码可以通过调用函数*esp_flash_encryption_enabled()* 来检查当前是否启用了 flash 加密。此外，设备可以通过调用函数*esp_get_flash_encryption_mode()* 来识别 flash 加密模式。

一旦启用 flash 加密，使用代码访问 flash 内容时要更加小心。

flash 加密范围 当 SPI_BOOT_CRYPT_CNT eFuse 设置为奇数位的值，所有通过 MMU 的 flash 缓存访问的 flash 内容都将被透明解密。包括：

- flash 中可执行的应用程序代码 (IROM)。
- 所有存储于 flash 中的只读数据 (DROM)。
- 通过函数*spi_flash_mmap()* 访问的任意数据。
- ROM 引导加载程序读取的固件引导加载程序镜像。

重要： MMU flash 缓存将无条件解密所有数据。flash 中未加密存储的数据将通过 flash 缓存“被透明解密”，并在软件中存储为随机垃圾数据。

读取加密的 flash 如果需要在不使用 flash 缓存 MMU 映射的情况下读取数据，推荐使用分区读取函数*esp_partition_read()*。该函数只会解密从加密分区读取的数据。从未加密分区读取的数据不会被解密。这样，软件便能以相同的方式访问加密和未加密的 flash。

也可以使用以下 SPI flash API 函数:

- 通过函数*esp_flash_read()* 读取不会被解密的原（加密）数据。
- 通过函数*esp_flash_read_encrypted()* 读取和解密数据。

使用非易失性存储器 (NVS) API 存储的数据始终从 flash 加密的角度进行存储和读取解密。如有需要，则由库提供加密功能。详情可参考*NVS 加密*。

写入加密的 flash 推荐使用分区写入函数*esp_partition_write()*。此函数只会在将数据写入加密分区时加密数据，而写入未加密分区的数据不会被加密。通过这种方式，软件可以以相同的方式访问加密和非加密 flash。

也可以使用函数*esp_flash_write_encrypted()* 预加密和写入数据。

此外，esp-idf 应用程序中存在但不支持以下 ROM 函数：

- esp_rom_spiflash_write_encrypted 预加密并将数据写入 flash
- SPIWrite 将未加密的数据写入 flash

由于数据是按块加密的，加密数据最小的写入大小为 16 字节，对齐也是 16 字节。

更新加密的 flash

OTA 更新 如果使用函数*esp_partition_write()*，对加密分区的 OTA 更新将自动以加密形式写入。

在为已加密设备的 OTA 更新构建应用程序镜像之前，启用项目配置菜单中的*启动时使能 flash 加密* 选项。

请参考*OTA* 获取更多关于 ESP-IDF OTA 更新的信息。

通过串口更新加密 flash 通过串行引导加载程序烧录加密设备，需要串行引导加载程序下载接口没有通过 eFuse 被永久禁用。

在开发模式下，推荐的方法是*重新烧录更新后的分区*。

在发布模式下，如果主机上有存储在 eFuse 中的相同密钥的副本，那么就可以在主机上对文件进行预加密，然后进行烧录，具体请参考*手动加密文件*。

关闭 flash 加密

如果意外启用了 flash 加密，则明文数据的 flash 会使 ESP32-P4 无法正常启动。设备将不断重启，并报错 `flash read err, 1000` 或 `invalid header: 0xXXXXXX`。

对于开发模式下的 flash 加密，可以通过烧录 `SPI_BOOT_CRYPT_CNT` efuse 来关闭加密。每个芯片仅有 1 次机会，请执行以下步骤：

1. 在*项目配置菜单* 中，禁用*启动时使能 flash 加密* 选项，然后保存并退出。
2. 再次打开项目配置菜单，再次检查你是否已经禁用了该选项，如果这个选项仍被启用，引导加载程序在启动时将立即重新启用加密功能。
3. 在禁用 flash 加密后，通过运行 `idf.py flash` 来构建和烧录新的引导加载程序和应用程序。
4. 使用 `espefuse.py`（在 `components/esptool_py/esptool` 中）以关闭 `SPI_BOOT_CRYPT_CNT`，运行：

```
espefuse.py burn_efuse SPI_BOOT_CRYPT_CNT
```

重置 ESP32-P4，flash 加密应处于关闭状态，引导加载程序将正常启动。

flash 加密的要点

- 通过 ESP32-P4 的 flash 缓存映射功能，flash 可支持透明访问——任何映射到地址空间的 flash 区域在读取时都将被透明地解密。
  为便于访问，某些数据分区最好保持未加密状态，或者也可使用对已加密数据无效的 flash 友好型更新算法。由于 NVS 库无法与 flash 加密直接兼容，因此无法加密非易失性存储器的 NVS 分区。详情可参见*NVS 加密*。
- 如果以后可能需要启用 flash 加密，则编程人员在编写*使用加密 flash* 代码时需小心谨慎。
- 如果已启用安全启动，重新烧录加密设备的引导加载程序则需要“可重新烧录”的安全启动摘要（可参考*flash 加密与安全启动*）。

启用 flash 加密将增大引导加载程序，因此可能需更新分区表偏移量。请参考*引导加载程序大小*。

重要： 在首次启动加密过程中，请勿切断 ESP32-P4 的电源。如果电源被切断，flash 的内容将受到破坏，并需要重新烧录未加密数据。而这类重新烧录将不计入烧录限制次数。

---

flash 加密的局限性

flash 加密可以保护固件，防止未经授权的读取与修改。了解 flash 加密系统的局限之处亦十分重要：

- flash 加密功能与密钥同样稳固。因而，推荐在首次启动设备时，在设备上生成密钥（默认行为）。如果在设备外生成密钥，请确保遵循正确的后续步骤，不要在所有生产设备之间使用相同的密钥。
- 并非所有数据都是加密存储。因而在 flash 上存储数据时，请检查你使用的存储方式（库、API 等）是否支持 flash 加密。
- flash 加密无法防止攻击者获取 flash 的高层次布局信息。这是因为每对相邻的 16 字节 AES 块都使用相邻的 AES 密钥。当这些相邻的 16 字节块中包含相同内容时（如空白或填充区域），这些字节块将加密以产生匹配的加密块对。这让攻击者可在加密设备间进行高层次对比（例如，确认两设备是否可能运行相同的固件版本）。
- 单独使用 flash 加密可能无法防止攻击者修改本设备的固件。为防止设备上运行未经授权的固件，可搭配 flash 加密使用*安全启动*。

flash 加密与安全启动

推荐 flash 加密与安全启动搭配使用。但是，如果已启用安全启动，则重新烧录设备时会受到其他限制：

- 如果新的应用程序已使用安全启动签名密钥正确签名，则*OTA 更新* 不受限制。

flash 加密的高级功能

以下部分介绍了 flash 加密的高级功能。

加密分区标志 部分分区默认为已加密。通过在分区的标志字段中添加 “encrypted” 标志，可在分区表描述中将其他分区标记为需要加密。在这些标记分区中的数据会和应用程序分区一样视为加密数据。

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000
phy_init, data, phy,     0xf000,  0x1000
factory,  app,  factory, 0x10000, 1M
secret_data, 0x40, 0x01, 0x20000, 256K, encrypted
```

请参考分区表 获取更多关于分区表描述的具体信息。

关于分区加密，还需了解以下信息：

- 默认分区表都不包含任何加密数据分区。
- 启用 flash 加密后，”app” 分区一般都视为加密分区，因此无需标记。
- 如果未启用 flash 加密，则”encrypted” 标记无效。
- 将可选 `phy` 分区标记为”encrypted”，可以防止物理访问读取或修改 `phy_init` 数据。
- `nvs` 分区无法标记为”encrypted” 因为 NVS 库与 flash 加密不直接兼容。

启用 UART 引导加载程序加密/解密 在第一次启动时，flash 加密过程默认会烧录以下 eFuse：

- `DIS_DOWNLOAD_MANUAL_ENCRYPT` 在 UART 引导加载程序启动模式下运行时，禁止 flash 加密操作。
- `DIS_DIRECT_BOOT`` （即之前的 ``DIS_LEGACY_SPI_BOOT`）禁用传统的 SPI 启动模式。

为了能启用这些功能，可在首次启动前仅烧录部分 eFuse，并用未设置值 0 写保护其他部分。例如：

```
espefuse.py --port PORT burn_efuse DIS_DOWNLOAD_MANUAL_ENCRYPT
espefuse.py --port PORT write_protect_efuse DIS_DOWNLOAD_MANUAL_ENCRYPT
```

备注： 请注意在写保护前设置所有适当的位！

一个位可以控制三个 eFuse 的写保护，这意味着写保护一个 eFuse 位将写保护所有未设置的 eFuse 位。

由于 `esptool.py` 目前不支持读取加密 flash，所以对这些 eFuse 进行写保护从而使其保持未设置目前来说并不是很有用。

JTAG 调试 默认情况下，当启用 flash 加密（开发或发布模式）时，将通过 eFuse 禁用 JTAG 调试。引导加载程序在首次启动时执行此操作，同时启用 flash 加密。

请参考*JTAG 与 flash 加密和安全引导* 了解更多关于使用 JTAG 调试与 flash 加密的信息。

手动加密文件 手动加密或解密文件需要在 eFuse 中预烧录 flash 加密密钥（请参阅使用主机生成的密钥）并在主机上保留一份副本。如果 flash 加密配置在开发模式下，那么则不需要保留密钥的副本或遵循这些步骤，可以使用更简单的重新烧录更新后的分区 步骤。

密钥文件应该是单个原始二进制文件（例如：`key.bin`）。

例如，以下是将文件 `build/my-app.bin` 进行加密、烧录到偏移量 0x10000 的步骤。运行 `espsecure.py`，如下所示：

```
espsecure.py encrypt_flash_data --aes_xts --keyfile /path/to/key.bin --address␣
↪0x10000 --output my-app-ciphertext.bin build/my-app.bin
```

然后可以使用 `esptool.py` 将文件 `my-app-ciphertext.bin` 写入偏移量 0x10000。关于为 `esptool.py` 推荐的所有命令行选项，请查看 idf.py build 成功时打印的输出。

备注:

如果 ESP32-P4 在启动时无法识别烧录进去的密文文件，请检查密钥是否匹配以及命令行参数是否完全匹配，包括偏移量是否正确。

`espsecure.py decrypt_flash_data` 命令可以使用同样的选项（和不同的输入/输出文件）来解密 flash 密文或之前加密的文件。

片外 RAM

启用 flash 加密后，任何通过缓存从片外 SPI RAM 读取和写入的数据也将被加密/解密。这个实现的方式以及使用的密钥与 flash 加密相同。如果启用 flash 加密，则片外 SPI RAM 的加密也会被启用，无法单独控制此功能。

技术细节

以下章节将提供 flash 加密操作的相关信息。

- 有关在 Python 中实现的完整 flash 加密算法，可参见 `espsecure.py` 源代码中的函数 `_flash_encryption_operation()`。

flash 加密算法

- ESP32-P4 使用 XTS-AES 块密码模式进行 flash 加密，密钥大小为 256 位或 512 位。
- XTS-AES 是一种专门为光盘加密设计的块密码模式，它解决了其它潜在模式如 AES-CTR 在此使用情景下的不足。有关 XTS-AES 算法的详细描述，请参考 IEEE Std 1619-2007。
- flash 加密的密钥存储于一个或两个 `BLOCK_KEYN` eFuse 中，默认受保护防止进一步写入或软件读取。
- 有关在 Python 中实现的完整 flash 加密算法，可参见 `espsecure.py` 源代码中的函数 `_flash_encryption_operation()`。

6.2.2 Secure Boot V2

重要: This document is about Secure Boot V2, supported on ESP32-P4

Secure Boot V2 uses RSA-PSS or ECDSA based app and bootloader (二级引导程序) verification. This document can also be used as a reference for signing apps using the RSA-PSS or ECDSA scheme without signing the bootloader.

Background

Secure Boot protects a device from running any unauthorized (i.e., unsigned) code by checking that each piece of software that is being booted is signed. On an ESP32-P4, these pieces of software include the second stage bootloader and each application binary. Note that the first stage bootloader does not require signing as it is ROM code thus cannot be changed.

ESP32-P4 has provision to choose between a RSA-PSS or ECDSA based secure boot verification scheme.

The Secure Boot process on the ESP32-P4 involves the following steps:

1. When the first stage bootloader loads the second stage bootloader, the second stage bootloader's RSA-PSS or ECDSA signature is verified. If the verification is successful, the second stage bootloader is executed.
2. When the second stage bootloader loads a particular application image, the application's RSA-PSS or ECDSA signature is verified. If the verification is successful, the application image is executed.

Advantages

- The RSA-PSS or ECDSA public key is stored on the device. The corresponding RSA-PSS or ECDSA private key is kept at a secret place and is never accessed by the device.
- Up to three public keys can be generated and stored in the chip during manufacturing.
- ESP32-P4 provides the facility to permanently revoke individual public keys. This can be configured conservatively or aggressively.
- Conservatively - The old key is revoked after the bootloader and application have successfully migrated to a new key. Aggressively - The key is revoked as soon as verification with this key fails.
- Same image format and signature verification method is applied for applications and software bootloader.
- No secrets are stored on the device. Therefore, it is immune to passive side-channel attacks (timing or power analysis, etc.)

Secure Boot V2 Process

This is an overview of the Secure Boot V2 Process. Instructions how to enable Secure Boot are supplied in section *How To Enable Secure Boot V2*.

Secure Boot V2 verifies the bootloader image and application binary images using a dedicated *signature block*. Each image has a separately generated signature block which is appended to the end of the image.

Up to 3 signature blocks can be appended to the bootloader or application image in ESP32-P4.

Each signature block contains a signature of the preceding image as well as the corresponding RSA-3072, ECDSA-256, or ECDSA-192 public key. For more details about the format, refer to *Signature Block Format*. A digest of the RSA-3072, ECDSA-256, or ECDSA-192 public key is stored in the eFuse.

The application image is not only verified on every boot but also on each over the air (OTA) update. If the currently selected OTA app image cannot be verified, the bootloader will fall back and look for another correctly signed application image.

The Secure Boot V2 process follows these steps:

1. On startup, the ROM code checks the Secure Boot V2 bit in the eFuse. If Secure Boot is disabled, a normal boot will be executed. If Secure Boot is enabled, the boot will proceed according to the following steps.
2. The ROM code verifies the bootloader's signature block (*Verifying a Signature Block*). If this fails, the boot process will be aborted.
3. The ROM code verifies the bootloader image using the raw image data, its corresponding signature block(s), and the eFuse (*Verifying an Image*). If this fails, the boot process will be aborted.
4. The ROM code executes the bootloader.
5. The bootloader verifies the application image's signature block (*Verifying a Signature Block*). If this fails, the boot process will be aborted.
6. The bootloader verifies the application image using the raw image data, its corresponding signature blocks and the eFuse (*Verifying an Image*). If this fails, the boot process will be aborted. If the verification fails but another application image is found, the bootloader will then try to verify that other image using steps 5 to 7. This repeats until a valid image is found or no other images are found.
7. The bootloader executes the verified application image.

Signature Block Format

The signature block starts on a 4 KB aligned boundary and has a flash sector of its own. The signature is calculated over all bytes in the image including the padding bytes (*Secure Padding*).

备注： ESP32-P4 has a provision to choose between RSA scheme and ECDSA scheme. Only one scheme can be used per device.

ECDSA provides similar security strength, compared to RSA, with shorter key lengths. Current estimates are that ECDSA with curve P-256 has an approximate equivalent strength to RSA with 3072-bit keys. However, ECDSA signature verification takes considerably more amount of time as compared to RSA signature verification.

RSA is recommended for use cases where fast bootup time is required whereas ECDSA is recommended for use cases where shorter key length is required.

The content of each signature block is shown in the following table:

表 2: Content of a RSA Signature Block

Offset	Size (bytes)	Description
0	1	Magic byte
1	1	Version number byte (currently 0x02), 0x01 is for Secure Boot V1.
2	2	Padding bytes, Reserved. Should be zero.
4	32	SHA-256 hash of only the image content, not including the signature block.
36	384	RSA Public Modulus used for signature verification. (value ‘n’ in RFC8017).
420	4	RSA Public Exponent used for signature verification (value ‘e’ in RFC8017).
424	384	Pre-calculated R, derived from ‘n’.
808	4	Pre-calculated M’, derived from ‘n’
812	384	RSA-PSS Signature result (section 8.1.1 of RFC8017) of image content, computed using following PSS parameters: SHA256 hash, MGF1 function, salt length 32 bytes, default trailer field (0xBC).
1196	4	CRC32 of the preceding 1196 bytes.
1200	16	Zero padding to length 1216 bytes.

备注： R and M’ are used for hardware-assisted Montgomery Multiplication.

表 3: Content of a ECDSA Signature Block

Offset	Size (bytes)	Description
0	1	Magic byte.
1	1	Version number byte (currently 0x03).
2	2	Padding bytes, Reserved. Should be zero.
4	32	SHA-256 hash of only the image content, not including the signature block.
36	1	Curve ID (1 for NIST192p curve. 2 for NIST256p curve).
37	64	ECDSA Public key: 32 byte X coordinate followed by 32 byte Y coordinate.
101	64	ECDSA Signature result (section 5.3.2 of RFC6090) of the image content: 32 byte R component followed by 32 byte S component.
165	1031	Reserved.
1196	4	CRC32 of the preceding 1196 bytes.
1200	16	Zero padding to length 1216 bytes.

The remainder of the signature sector is erased flash (0xFF) which allows writing other signature blocks after previous signature block.

Secure Padding

In Secure Boot V2 scheme, the application image is padded to the flash MMU page size boundary to ensure that only verified contents are mapped in the internal address space. This is known as secure padding. Signature of the image is calculated after padding and then signature block (4KB) gets appended to the image.

- Default flash MMU page size is 64KB
- Secure padding is applied through the option `--secure-pad-v2` in the `elf2image` conversion using `esptool.py`

Following table explains the Secure Boot V2 signed image with secure padding and signature block appended:

表 4: Contents of a signed application

Offset	Size (KB)	Description
0	580	Unsigned application size (as an example)
580	60	Secure padding (aligned to next 64KB boundary)
640	4	Signature block

备注： Please note that the application image always starts on the next flash MMU page size boundary (default 64KB) and hence the space left over after the signature block shown above can be utilized to store any other data partitions (e.g., `nvs`).

Verifying a Signature Block

A signature block is ”valid” if the first byte is 0xe7 and a valid CRC32 is stored at offset 1196. Otherwise it is invalid.

Verifying an Image

An image is ”verified” if the public key stored in any signature block is valid for this device, and if the stored signature is valid for the image data read from flash.

1. Compare the SHA-256 hash digest of the public key embedded in the bootloader's signature block with the digest(s) saved in the eFuses. If public key's hash does not match any of the hashes from the eFuses, the verification fails.
2. Generate the application image digest and match it with the image digest in the signature block. If the digests do not match, the verification fails.
3. Use the public key to verify the signature of the bootloader image, using either RSA-PSS (section 8.1.2 of RFC8017) or ECDSA signature verification (section 5.3.3 of RFC6090) with the image digest calculated in step (2) for comparison.

Bootloader Size

Enabling Secure boot and/or flash encryption will increase the size of bootloader, which might require updating partition table offset. See 引导加载程序大小.

In the case when *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES* is disabled, the bootloader is sector padded (4KB) using the `--pad-to-size` option in `elf2image` command of `esptool`.

eFuse Usage

- SECURE_BOOT_EN - Enables Secure Boot protection on boot.
- KEY_PURPOSE_X - Set the purpose of the key block on ESP32-P4 by programming SECURE_BOOT_DIGESTX (X = 0, 1, 2) into KEY_PURPOSE_X (X = 0, 1, 2, 3, 4, 5). Example: If KEY_PURPOSE_2 is set to SECURE_BOOT_DIGEST1, then BLOCK_KEY2 will have the Secure Boot V2 public key digest. The write-protection bit must be set (this field does not have a read-protection bit).
- BLOCK_KEYX - The block contains the data corresponding to its purpose programmed in KEY_PURPOSE_X. Stores the SHA-256 digest of the public key. SHA-256 hash of public key modulus, exponent, pre-calculated R & M' values (represented as 776 bytes –offsets 36 to 812 - as per the *Signature Block Format*) is written to an eFuse key block. The write-protection bit must be set, but the read-protection bit must not.
- KEY_REVOKEX - The revocation bits corresponding to each of the 3 key block. Ex. Setting KEY_REVOKE2 revokes the key block whose key purpose is SECURE_BOOT_DIGEST2.
- SECURE_BOOT_AGGRESSIVE_REVOKE - Enables aggressive revocation of keys. The key is revoked as soon as verification with this key fails.

To ensure no trusted keys can be added later by an attacker, each unused key digest slot should be revoked (KEY_REVOKEX). It will be checked during app startup in `esp_secure_boot_init_checks()` and fixed unless *CONFIG_SECURE_BOOT_ALLOW_UNUSED_DIGEST_SLOTS* is enabled.

The key(s) must be readable in order to give software access to it. If the key(s) is read-protected then the software reads the key(s) as all zeros and the signature verification process will fail, and the boot process will be aborted.

How To Enable Secure Boot V2

1. Open the 项目配置菜单, in "Security features" set "Enable hardware Secure Boot in bootloader" to enable Secure Boot.
2. The "Secure Boot V2" option will be selected and the "App Signing Scheme" would be set to RSA by default. RSA is recommended because of faster verification time. You can choose between RSA and ECDSA scheme from the menu.
3. Specify the path to Secure Boot signing key, relative to the project directory.
4. Select the desired UART ROM download mode in "UART ROM download mode". By default, it is set to "Permanently switch to Secure mode" which is generally recommended. For production devices, the most secure option is to set it to "Permanently disabled".
5. Set other menuconfig options (as desired). Then exit menuconfig and save your configuration.
6. The first time you run `idf.py build`, if the signing key is not found then an error message will be printed with a command to generate a signing key via `espsecure.py generate_signing_key`.

重要： A signing key generated this way will use the best random number source available to the OS and its Python installation (*/dev/urandom* on OSX/Linux and *CryptGenRandom()* on Windows). If this random number source is weak, then the private key will be weak.

重要： For production environments, we recommend generating the key pair using openssl or another industry standard encryption program. See *Generating Secure Boot Signing Key* for more details.

---

7. Run `idf.py bootloader` to build a Secure Boot enabled bootloader. The build output will include a prompt for a flashing command, using `esptool.py write_flash`.
8. When you are ready to flash the bootloader, run the specified command (you have to enter it yourself, this step is not performed by the build system) and then wait for flashing to complete.
9. Run `idf.py flash` to build and flash the partition table and the just-built app image. The app image will be signed using the signing key you generated in step 6.

备注： `idf.py flash` does not flash the bootloader if Secure Boot is enabled.

10. Reset the ESP32-P4 and it will boot the software bootloader you flashed. The software bootloader will enable Secure Boot on the chip, and then it verifies the app image signature and boots the app. You should watch the serial console output from the ESP32-P4 to verify that Secure Boot is enabled and no errors have occurred due to the build configuration.

备注： Secure boot will not be enabled until after a valid partition table and app image have been flashed. This is to prevent accidents before the system is fully configured.

备注： If the ESP32-P4 is reset or powered down during the first boot, it will start the process again on the next boot.

11. On subsequent boots, the Secure Boot hardware will verify the software bootloader has not changed and the software bootloader will verify the signed app image (using the validated public key portion of its appended signature block).

Restrictions After Secure Boot Is Enabled

- Any updated bootloader or app will need to be signed with a key matching the digest already stored in eFuse.
- After Secure Boot is enabled, no further eFuses can be read protected. (If *flash 加密* is enabled then the bootloader will ensure that any flash encryption key generated on first boot will already be read protected.) If *CONFIG_SECURE_BOOT_INSECURE* is enabled then this behavior can be disabled, but this is not recommended.
- Please note that enabling Secure Boot or flash encryption disables the USB-OTG USB stack in the ROM, disallowing updates via the serial emulation or Device Firmware Update (DFU) on that port.

Generating Secure Boot Signing Key

The build system will prompt you with a command to generate a new signing key via `espsecure.py generate_signing_key`.

The `--version 2` parameter will generate the RSA 3072 private key for Secure Boot V2. Additionally `--scheme rsa3072` can be passed as well to generate RSA 3072 private key

Select the ECDSA scheme by passing `--version 2 --scheme ecdsa256` or `--version 2 --scheme ecdsa192` to generate corresponding ECDSA private key

The strength of the signing key is proportional to (a) the random number source of the system, and (b) the correctness of the algorithm used. For production devices, we recommend generating signing keys from a system with a quality entropy source, and using the best available RSA-PSS or ECDSA key generation utilities.

For example, to generate a signing key using the openssl command line:

For RSA 3072

```
` openssl genrsa -out my_secure_boot_signing_key.pem 3072 `
```

For ECC NIST192p curve

```
` openssl ecparam -name prime192v1 -genkey -noout -out
my_secure_boot_signing_key.pem `
```

For ECC NIST256p curve

```
` openssl ecparam -name prime256v1 -genkey -noout -out
my_secure_boot_signing_key.pem `
```

Remember that the strength of the Secure Boot system depends on keeping the signing key private.

Remote Signing of Images

Signing Using `espsecure.py` For production builds, it can be good practice to use a remote signing server rather than have the signing key on the build machine (which is the default esp-idf Secure Boot configuration). The espsecure.py command line program can be used to sign app images & partition table data for Secure Boot, on a remote system.

To use remote signing, disable the option *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES* and build the firmware. The private signing key does not need to be present on the build system.

After the app image and partition table are built, the build system will print signing steps using espsecure.py:

```
espsecure.py sign_data BINARY_FILE --version 2 --keyfile PRIVATE_SIGNING_KEY
```

The above command appends the image signature to the existing binary. You can use the *--output* argument to write the signed binary to a separate file:

```
espsecure.py sign_data --version 2 --keyfile PRIVATE_SIGNING_KEY --output SIGNED_
↪BINARY_FILE BINARY_FILE
```

Signing Using Pre-calculated Signatures If you have valid pre-calculated signatures generated for an image and their corresponding public keys, you can use these signatures to generate a signature sector and append it to the image. Note that the pre-calculated signature should be calculated over all bytes in the image including the secure-padding bytes.

In such cases, the firmware image should be built by disabling the option *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES*. This image will be secure-padded and to generate a signed binary use the following command:

```
espsecure.py sign_data --version 2 --pub-key PUBLIC_SIGNING_KEY --signature␣
↪SIGNATURE_FILE --output SIGNED_BINARY_FILE BINARY_FILE
```

The above command verifies the signature, generates a signature block (refer to *Signature Block Format*) and appends it to the binary file.

Signing Using an External Hardware Security Module (HSM) For security reasons, you might also use an external Hardware Security Module (HSM) to store your private signing key, which cannot be accessed directly but has an interface to generate the signature of a binary file and its corresponding public key.

In such cases, disable the option *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES* and build the firmware. This secure-padded image then can be used to supply the external HSM for generating a signature. Refer to Signing using an External HSM to generate a signed image.

备注： For all the above three remote signing workflows, the signed binary is written to the filename provided to the `--output` argument and the option `--append_signatures` allows us to append multiple signatures (up to 3) the image.

Secure Boot Best Practices

- Generate the signing key on a system with a quality source of entropy.
- Keep the signing key private at all times. A leak of this key will compromise the Secure Boot system.
- Do not allow any third party to observe any aspects of the key generation or signing process using espsecure.py. Both processes are vulnerable to timing or other side-channel attacks.
- Enable all Secure Boot options in the Secure Boot Configuration. These include flash encryption, disabling of JTAG, disabling BASIC ROM interpreter, and disabling the UART bootloader encrypted flash access.
- Use Secure Boot in combination with *flash 加密* to prevent local readout of the flash contents.

Key Management

- Between 1 and 3 RSA-3072, ECDSA-256, or ECDSA-192 public key pairs (Keys #0, #1, #2) should be computed independently and stored separately.
- The KEY_DIGEST eFuses should be write protected after being programmed.
- The unused KEY_DIGEST slots must have their corresponding KEY_REVOKE eFuse burned to permanently disable them. This must happen before the device leaves the factory.
- The eFuses can either be written by the software bootloader during during first boot after enabling "Secure Boot V2" from menuconfig or can be done using *espefuse.py* which communicates with the serial bootloader program in ROM.
- The KEY_DIGESTs should be numbered sequentially beginning at key digest #0. (i.e., if key digest #1 is used, key digest #0 should be used. If key digest #2 is used, key digest #0 & #1 must be used.)
- The software bootloader (non OTA upgradeable) is signed using at least one, possibly all three, private keys and flashed in the factory.
- Apps should only be signed with a single private key (the others being stored securely elsewhere), however they may be signed with multiple private keys if some are being revoked (see Key Revocation, below).

Multiple Keys

- The bootloader should be signed with all the private key(s) that are needed for the life of the device, before it is flashed.
- The build system can sign with at most one private key, user has to run manual commands to append more signatures if necessary.
- **You can use the append functionality of `espsecure.py`, this command would also printed at the end of the Secure B**

  espsecure.py sign_data -k secure_boot_signing_key2.pem -v 2 --append_signatures -o signed_bootloader.bin build/bootloader/bootloader.bin
- While signing with multiple private keys, it is recommended that the private keys be signed independently, if possible on different servers and stored separately.
- **You can check the signatures attached to a binary using -** espsecure.py signature_info_v2 datafile.bin

Key Revocation

- Keys are processed in a linear order. (key #0, key #1, key #2).
- Applications should be signed with only one key at a time, to minimize the exposure of unused private keys.
- The bootloader can be signed with multiple keys from the factory.

备注: Note that enabling the config *CONFIG_SECURE_BOOT_ALLOW_UNUSED_DIGEST_SLOTS* only makes sure that the **app** does not revoke the unused digest slots. But if you plan to enable secure boot during the fist boot up, the bootloader will intentionally revoke the unused digest slots while enabling secure boot, even if the above config is enabled because keeping the unused key slots un-revoked would a security hazard. In case for any development workflow if you need to avoid this revocation, you should enable secure boot externally (*Enable Secure Boot V2 Externally*) rather than enabling it during the boot up, so that the bootloader would not need to enable secure boot and thus you could avoid its revocation strategy.

Conservative Approach: Assuming a trusted private key (N-1) has been compromised, to update to new key pair (N).

1. Server sends an OTA update with an application signed with the new private key (#N).
2. The new OTA update is written to an unused OTA app partition.
3. The new application's signature block is validated. The public keys are checked against the digests programmed in the eFuse & the application is verified using the verified public key.
4. The active partition is set to the new OTA application's partition.
5. Device resets, loads the bootloader (verified with key #N-1 and #N) which then boots new app (verified with key #N).

6. The new app verifies bootloader and application with key #N (as a final check) and then runs code to revoke key #N-1 (sets KEY_REVOKE eFuse bit).
7. The API *esp_ota_revoke_secure_boot_public_key()* can be used to revoke the key #N-1.

- A similar approach can also be used to physically re-flash with a new key. For physical re-flashing, the bootloader content can also be changed at the same time.

Aggressive Approach: ROM code has an additional feature of revoking a public key digest if the signature verification fails.

To enable this feature, you need to burn SECURE_BOOT_AGGRESSIVE_REVOKE efuse or enable *CONFIG_SECURE_BOOT_ENABLE_AGGRESSIVE_KEY_REVOKE*

Key revocation is not applicable unless secure boot is successfully enabled. Also, a key is not revoked in case of invalid signature block or invalid image digest, it is only revoked in case the signature verification fails, i.e., revoke key only if failure in step 3 of *Verifying an Image*

Once a key is revoked, it can never be used for verifying a signature of an image. This feature provides strong resistance against physical attacks on the device. However, this could also brick the device permanently if all the keys are revoked because of signature verification failure.

Technical Details

The following sections contain low-level reference descriptions of various Secure Boot elements:

Manual Commands Secure boot is integrated into the esp-idf build system, so `idf.py build` will sign an app image and `idf.py bootloader` will produce a signed bootloader if secure signed binaries on build is enabled.

However, it is possible to use the `espsecure.py` tool to make standalone signatures and digests.

To sign a binary image:

```
espsecure.py sign_data --version 2 --keyfile ./my_signing_key.pem --output ./image_
↪signed.bin image-unsigned.bin
```

Keyfile is the PEM file containing an RSA-3072, ECDSA-256, or ECDSA-192 private signing key.

Secure Boot & Flash Encryption

If Secure Boot is used without *flash 加密*, it is possible to launch "time-of-check to time-of-use" attack, where flash contents are swapped after the image is verified and running. Therefore, it is recommended to use both the features together.

Signed App Verification Without Hardware Secure Boot

The Secure Boot V2 signature of apps can be checked on OTA update, without enabling the hardware Secure Boot option. This option uses the same app signature scheme as Secure Boot V2, but unlike hardware Secure Boot it does not prevent an attacker who can write to flash from bypassing the signature protection.

This may be desirable in cases where the delay of Secure Boot verification on startup is unacceptable, and/or where the threat model does not include physical access or attackers writing to bootloader or app partitions in flash.

In this mode, the public key which is present in the signature block of the currently running app will be used to verify the signature of a newly updated app. (The signature on the running app is not verified during the update process, it is assumed to be valid.) In this way the system creates a chain of trust from the running app to the newly updated app.

For this reason, it is essential that the initial app flashed to the device is also signed. A check is run on app startup and the app will abort if no signatures are found. This is to try and prevent a situation where no update is possible. The

app should have only one valid signature block in the first position. Note again that, unlike hardware Secure Boot V2, the signature of the running app is not verified on boot. The system only verifies a signature block in the first position and ignores any other appended signatures.

Although multiple trusted keys are supported when using hardware Secure Boot, only the first public key in the signature block is used to verify updates if signature checking without Secure Boot is configured. If multiple trusted public keys are required, it is necessary to enable the full Secure Boot feature instead.

备注： In general, it is recommended to use full hardware Secure Boot unless certain that this option is sufficient for application security needs.

How To Enable Signed App Verification

1. Open 项目配置菜单 -> Security features
2. Choose *App Signing Scheme*. Either *RSA* or *ECDSA (V2)*
3. Enable *CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT*
4. By default, "Sign binaries during build" will be enabled on selecting "Require signed app images" option, which will sign binary files as a part of build process. The file named in "Secure boot private signing key" will be used to sign the image.
5. If you disable "Sign binaries during build" option then all app binaries must be manually signed by following instructions in *Remote Signing of Images*.

警告： It is very important that all apps flashed have been signed, either during the build or after the build.

Advanced Features

JTAG Debugging By default, when Secure Boot is enabled then JTAG debugging is disabled via eFuse. The bootloader does this on first boot, at the same time it enables Secure Boot.

See *JTAG 与 flash 加密和安全引导* for more information about using JTAG Debugging with either Secure Boot or signed app verification enabled.

6.3 流程

6.3.1 Host-Based Security Workflows

Introduction

It is recommended to have an uninterrupted power supply while enabling security features on ESP32 SoCs. Power failures during the secure manufacturing process could cause issues that are hard to debug and, in some cases, may cause permanent boot-up failures.

This guide highlights an approach where security features are enabled with the assistance of an external host machine. Security workflows are broken down into various stages and key material is generated on the host machine; thus, allowing greater recovery chances in case of power or other failures. It also offers better timings for secure manufacturing, e.g., in the case of encryption of firmware on the host machine vs. on the device.

Goals

1. Simplify the traditional workflow with stepwise instructions.
2. Design a more flexible workflow as compared to the traditional firmware-based workflow.
3. Improve reliability by dividing the workflow into small operations.

4. Eliminate dependency on 二级引导程序 (firmware bootloader).

Pre-requisite

- `esptool`: Please make sure the `esptool` has been installed. It can be installed by running:

```
pip install esptool
```

Scope

- *Enable Flash Encryption and Secure Boot V2 Externally*
- *Enable Flash Encryption Externally*
- *Enable Secure Boot V2 Externally*
- *Enable NVS Encryption Externally*

Security Workflows

Enable Flash Encryption and Secure Boot V2 Externally

重要： It is recommended to enable both Flash Encryption and Secure Boot V2 for a production use case.

When enabling the Flash Encryption and Secure Boot V2 together we need to enable them in the following order:

1. Enable the Flash Encryption feature by following the steps listed in *Enable Flash Encryption Externally*.
2. Enable the Secure Boot V2 feature by following the steps listed in *Enable Secure Boot V2 Externally*.

The reason for this order is as follows:

备注： To enable the Secure Boot (SB) V2, it is necessary to keep the SB V2 key readable. To protect the key's readability, the write protection for `RD_DIS` (`ESP_EFUSE_WR_DIS_RD_DIS`) is applied. However, this action poses a challenge when attempting to enable Flash Encryption, as the Flash Encryption (FE) key needs to remain unreadable. This conflict arises because the `RD_DIS` is already write-protected, making it impossible to read protect the FE key.

Enable Flash Encryption Externally

In this case, all the eFuses related to Flash Encryption are written with help of the espefuse tool. More details about flash encryption can be found in the *Flash Encryption Guide*

1. Check device status

   Ensure that you have an ESP32-P4 device with default Flash Encryption eFuse settings as shown in 相关 *eFuses*.

   See how to check *ESP32-P4 flash* 加密状态.

   At this point, the Flash Encryption must not be already enabled on the chip. Additionally, the flash on the chip needs to be erased, which can be done by running:

   ```
   esptool.py --port PORT erase_flash
   ```

2. Generate a Flash Encryption key

   A random Flash Encryption key can be generated by running:

   If *Size of generated AES-XTS key* is AES-128 (256-bit key):

   ```
   espsecure.py generate_flash_encryption_key my_flash_encryption_key.bin
   ```

   else if *Size of generated AES-XTS key* is AES-256 (512-bit key):

```
espsecure.py generate_flash_encryption_key --keylen 512 my_flash_
↪encryption_key.bin
```

3. Burn the Flash Encryption key into eFuse

   This action **cannot be reverted**. It can be done by running:

```
espefuse.py --port PORT burn_key BLOCK my_flash_encryption_key.bin␣
↪KEYPURPOSE
```

   where `BLOCK` is a free keyblock between `BLOCK_KEY0` and `BLOCK_KEY5`. And `KEYPURPOSE` is either `XTS_AES_256_KEY_1`, `XTS_AES_256_KEY_2`, `XTS_AES_128_KEY`. See ESP32-P4 Technical Reference Manual for a description of the key purposes.

   For AES-128 (256-bit key) - `XTS_AES_128_KEY`:

```
espefuse.py --port PORT burn_key BLOCK my_flash_encryption_key.bin XTS_
↪AES_128_KEY
```

   For AES-256 (512-bit key) - `XTS_AES_256_KEY_1` and `XTS_AES_256_KEY_2`. `espefuse.py` supports burning both these two key purposes together with a 512-bit key to two separate key blocks via the virtual key purpose `XTS_AES_256_KEY`. When this is used `espefuse.py` will burn the first 256 bits of the key to the specified `BLOCK` and burn the corresponding block key purpose to `XTS_AES_256_KEY_1`. The last 256 bits of the key will be burned to the first free key block after `BLOCK` and the corresponding block key purpose to `XTS_AES_256_KEY_2`

```
espefuse.py --port PORT burn_key BLOCK my_flash_encryption_key.bin XTS_
↪AES_256_KEY
```

   If you wish to specify exactly which two blocks are used then it is possible to divide the key into two 256-bit keys, and manually burn each half with `XTS_AES_256_KEY_1` and `XTS_AES_256_KEY_2` as key purposes:

```
split -b 32 my_flash_encryption_key.bin my_flash_encryption_key.bin
espefuse.py --port PORT burn_key BLOCK my_flash_encryption_key.bin.aa XTS_
↪AES_256_KEY_1
espefuse.py --port PORT burn_key BLOCK+1 my_flash_encryption_key.bin.ab␣
↪XTS_AES_256_KEY_2
```

4. Burn the `SPI_BOOT_CRYPT_CNT` eFuse

   If you only want to enable Flash Encryption in **Development** mode and want to keep the ability to disable it in the future, Update the SPI_BOOT_CRYPT_CNT value in the below command from 7 to 0x1 (not recommended for production).

```
espefuse.py --port PORT --chip esp32p4 burn_efuse SPI_BOOT_CRYPT_CNT 7
```

5. Burn Flash Encryption-related security eFuses as listed below

   A) Burn security eFuses

   重要: For production use cases, it is highly recommended to burn all the eFuses listed below.

   - `DIS_DIRECT_BOOT`: Disable direct boot (legacy SPI boot mode)
   - `DIS_USB_JTAG`: Disable USB switch to JTAG
   - `DIS_PAD_JTAG`: Disable JTAG permanently
   - `DIS_DOWNLOAD_MANUAL_ENCRYPT`: Disable UART bootloader encryption access
   - `DIS_DOWNLOAD_MSPI`: Disable the MSPI access in download mode

The respective eFuses can be burned by running:

```
espefuse.py burn_efuse --port PORT EFUSE_NAME 0x1
```

备注： Please update the EFUSE_NAME with the eFuse that you need to burn. Multiple eFuses can be burned at the same time by appending them to the above command (e.g., EFUSE_NAME VAL EFUSE_NAME2 VAL2). More documentation about *espefuse.py* can be found here.

6. Configure the project

   The bootloader and the application binaries for the project must be built with Flash Encryption Release mode with default configurations.

   Flash encryption Release mode can be set in the menuconfig as follows:

   - *Enable Flash Encryption on boot*
   - *Select Release mode* (Note that once Release mode is selected, the `EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT` eFuse bit will be burned to disable Flash Encryption hardware in ROM Download Mode)
   - *Select UART ROM download mode (Permanently switch to Secure mode (recommended))*. This is the default option and is recommended. It is also possible to change this configuration setting to permanently disable UART ROM download mode, if this mode is not needed
   - *Select the appropriate bootloader log verbosity*
   - Save the configuration and exit

7. Build, Encrypt and Flash the binaries

   The binaries can be encrypted on the host machine by running:

```
espsecure.py encrypt_flash_data --aes_xts --keyfile my_flash_encryption_
↪key.bin --address 0x1000 --output bootloader-enc.bin build/bootloader/
↪bootloader.bin

espsecure.py encrypt_flash_data --aes_xts --keyfile my_flash_encryption_
↪key.bin --address 0x8000 --output partition-table-enc.bin build/
↪partition_table/partition-table.bin

espsecure.py encrypt_flash_data --aes_xts --keyfile my_flash_encryption_
↪key.bin --address 0x10000 --output my-app-enc.bin build/my-app.bin
```

   In the above command the offsets are used for a sample firmware, the actual offset for your firmware can be obtained by checking the partition table entry or by running *idf.py partition-table*. Please note that not all the binaries need to be encrypted, the encryption applies only to those generated from the partitions which are marked as `encrypted` in the partition table definition file. Other binaries are flashed unencrypted, i.e., as a plain output of the build process.

   The above files can then be flashed to their respective offset using `esptool.py`. To see all of the command line options recommended for `esptool.py`, see the output printed when `idf.py build` succeeds.

   When the application contains the following partition: `otadata`, `nvs_encryption_keys` they need to be encrypted as well. Please refer to 加密分区 for more details about encrypted partitions.

备注： If the flashed ciphertext file is not recognized by the ESP32-P4 when it boots, check that the keys match and that the command line arguments match exactly, including the correct offset. It is important to provide the correct offset as the ciphertext changes when the offset changes.

The command `espsecure.py decrypt_flash_data` can be used with the same options (and different input/output files), to decrypt ciphertext flash contents or a previously encrypted file.

8. Secure the ROM Download mode

警告： Please perform the following step at the very end. After this eFuse is burned, the espefuse tool can no longer be used to burn additional eFuses.

Disable UART ROM DL mode:

- `UART_DOWNLOAD_DIS` : Disable the UART ROM Download mode

The eFuse can be burned by running:

```
espefuse.py --port PORT burn_efuse UART_DOWNLOAD_DIS
```

Enable Security Download mode:

- `ENABLE_SECURITY_DOWNLOAD`: Enable Secure ROM download mode

The eFuse can be burned by running:

```
espefuse.py --port PORT burn_efuse ENABLE_SECURITY_DOWNLOAD
```

重要:

9. Delete Flash Encryption key on host
   Once the Flash Encryption has been enabled for the device, the key **must be deleted immediately**. This ensures that the host cannot produce encrypted binaries for the same device going forward. This step is important to reduce the vulnerability of the flash encryption key.

Flash Encryption Guidelines

- It is recommended to generate a unique Flash Encryption key for each device for production use-cases.
- It is recommended to ensure that the RNG used by host machine to generate the Flash Encryption key has good entropy.
- See *flash 加密的局限性* for more details.

Enable Secure Boot V2 Externally In this workflow, we shall use `espsecure` tool to generate signing keys and use the `espefuse` tool to burn the relevant eFuses. The details about the Secure Boot V2 process can be found at *Secure Boot V2 Guide*

1. Generate Secure Boot V2 Signing Private Key

   The Secure Boot V2 signing key for the RSA3072 scheme can be generated by running:

```
espsecure.py generate_signing_key --version 2 --scheme rsa3072 secure_
↪boot_signing_key.pem
```

   The Secure Boot V2 signing key for ECDSA scheme can be generated by running:

```
espsecure.py generate_signing_key --version 2 --scheme ecdsa256 secure_
↪boot_signing_key.pem
```

The scheme in the above command can be changed to `ecdsa192` to generate ecdsa192 private key.

A total of 3 keys can be used for Secure Boot V2 at once. These should be computed independently and stored separately. The same command with different key file names can be used to generate multiple Secure Boot V2 signing keys. It is recommended to use multiple keys in order to reduce dependency on a single key.

2. Generate Public Key Digest

   The public key digest for the private key generated in the previous step can be generated by running:

```
espsecure.py digest_sbv2_public_key --keyfile secure_boot_signing_key.pem␣
↪--output digest.bin
```

   In case of multiple digests, each digest should be kept in a separate file.

3. Burn the key digest in eFuse

   The public key digest can be burned in the eFuse by running:

```
espefuse.py --port PORT --chip esp32p4 burn_key BLOCK digest.bin SECURE_
↪BOOT_DIGEST0
```

   where `BLOCK` is a free keyblock between `BLOCK_KEY0` and `BLOCK_KEY5`.

   In case of multiple digests, the other digests can be burned sequentially by changing the key purpose to `SECURE_BOOT_DIGEST1` and `SECURE_BOOT_DIGEST2` respectively.

4. Enable Secure Boot V2

   Secure Boot V2 eFuse can be enabled by running:

```
espefuse.py --port PORT --chip esp32p4 burn_efuse SECURE_BOOT_EN
```

5. Burn relevant eFuses

   A) Burn security eFuses

   重要： For production use cases, it is highly recommended to burn all the eFuses listed below.

   - `SOFT_DIS_JTAG`: Disable software access to JTAG peripheral
   - `DIS_DIRECT_BOOT`: Disable direct boot (legacy SPI boot mode)
   - `DIS_USB_JTAG`: Disable USB switch to JTAG
   - `DIS_PAD_JTAG`: Disable JTAG permanently
   - `SECURE_BOOT_AGGRESSIVE_REVOKE`: Aggressive revocation of key digests, see *Aggressive Approach:* for more details.

   The respective eFuses can be burned by running:

```
espefuse.py burn_efuse --port PORT EFUSE_NAME 0x1
```

   备注： Please update the EFUSE_NAME with the eFuse that you need to burn. Multiple eFuses can be burned at the same time by appending them to the above command (e.g., EFUSE_NAME VAL EFUSE_NAME2 VAL2). More documentation about *espefuse.py* can be found here

   B) Secure Boot V2-related eFuses

i) Disable the read-protection option:

The Secure Boot digest burned in the eFuse must be kept readable otherwise the Secure Boot operation would result in a failure. To prevent the accidental enabling of read protection for this key block, the following eFuse needs to be burned:

重要： After burning above-mentioned eFuse, the read protection cannot be enabled for any key. E.g., if Flash Encryption which requires read protection for its key is not enabled at this point, then it cannot be enabled afterwards. Please ensure that no eFuse keys are going to need read protection after completing this step.

```
espefuse.py -p $ESPPORT write_protect_efuse RD_DIS
```

ii) Revoke key digests:

The unused digest slots need to be revoked when we are burning the Secure Boot key. The respective slots can be revoked by running

```
espefuse.py --port PORT --chip esp32p4 burn_efuse EFUSE_REVOKE_BIT
```

The `EFUSE_REVOKE_BIT` in the above command can be `SECURE_BOOT_KEY_REVOKE0` or `SECURE_BOOT_KEY_REVOKE1` or `SECURE_BOOT_KEY_REVOKE2`. Please note that only the unused key digests must be revoked. Once revoked, the respective digest cannot be used again.

6. Configure the project

   By default, the ROM bootloader would only verify the 二级引导程序 (firmware bootloader). The firmware bootloader would verify the app partition only when the *CONFIG_SECURE_BOOT* option is enabled (and *CONFIG_SECURE_BOOT_VERSION* is set to `SECURE_BOOT_V2_ENABLED`) while building the bootloader.

   a) Open the 项目配置菜单, in "Security features" set "Enable hardware Secure Boot in bootloader" to enable Secure Boot.

   The "Secure Boot V2" option will be selected and the "App Signing Scheme" will be set to RSA by default.

   b) Disable the option *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES* for the project in the 项目配置菜单. This shall make sure that all the generated binaries are secure padded and unsigned. This step is done to avoid generating signed binaries as we are going to manually sign the binaries using `espsecure` tool.

7. Build, Sign and Flash the binaries

   After the above configurations, the bootloader and application binaries can be built with `idf.py build` command.

   The Secure Boot V2 workflow only verifies the `bootloader` and `application` binaries, hence only those binaries need to be signed. The other binaries (e.g., `partition-table.bin`) can be flashed as they are generated in the build stage.

   The `bootloader.bin` and `app.bin` binaries can be signed by running:

```
espsecure.py sign_data --version 2 --keyfile secure_boot_signing_key.pem -
↪-output bootloader-signed.bin build/bootloader/bootloader.bin

espsecure.py sign_data --version 2 --keyfile secure_boot_signing_key.pem -
↪-output my-app-signed.bin build/my-app.bin
```

   If multiple keys Secure Boot keys are to be used then the same signed binary can be appended with a signature block signed with the new key as follows:

```
espsecure.py sign_data --keyfile secure_boot_signing_key2.pem --version 2␣
↪--append_signatures -o bootloader-signed2.bin bootloader-signed.bin

espsecure.py sign_data --keyfile secure_boot_signing_key2.pem --version 2␣
↪--append_signatures -o my-app-signed2.bin my-app-signed.bin
```

The same process can be repeated for the third key. Note that the names of the input and output files must not be the same.

The signatures attached to a binary can be checked by running:

```
espsecure.py signature_info_v2 bootloader-signed.bin
```

The above files along with other binaries (e.g., partition table) can then be flashed to their respective offset using `esptool.py`. To see all of the command line options recommended for `esptool.py`, see the output printed when `idf.py build` succeeds. The flash offset for your firmware can be obtained by checking the partition table entry or by running `idf.py partition-table`.

8. Secure the ROM Download mode:

警告： Please perform the following step at the very end. After this eFuse is burned, the espefuse tool can no longer be used to burn additional eFuses.

Disable UART ROM DL mode:

- `UART_DOWNLOAD_DIS` : Disable the UART ROM Download mode

The eFuse can be burned by running:

```
espefuse.py --port PORT burn_efuse UART_DOWNLOAD_DIS
```

Enable Security Download mode:

- `ENABLE_SECURITY_DOWNLOAD`: Enable Secure ROM download mode

The eFuse can be burned by running:

```
espefuse.py --port PORT burn_efuse ENABLE_SECURITY_DOWNLOAD
```

Secure Boot V2 Guidelines

- It is recommended to store the Secure Boot key in a highly secure place. A physical or a cloud HSM may be used for secure storage of the Secure Boot private key. Please take a look at *Remote Signing of Images* for more details.
- It is recommended to use all the available digest slots to reduce dependency on a single private key.

Enable NVS Encryption Externally The details about NVS Encryption and related schemes can be found at *NVS Encryption*.

Enable NVS Encryption based on HMAC

1. Generate the HMAC key and NVS Encryption key

   In the HMAC based NVS scheme, there are two keys:

   - HMAC key - this is a 256 bit HMAC key that shall be stored in the eFuse

- NVS Encryption key - This is the NVS Encryption key that is used to encrypt the NVS partition. This key is derived at run-time using the HMAC key.

The above keys can be generated with the nvs_flash/nvs_partition_generator/nvs_partition_gen.py script with help of the following command:

```
python3 nvs_partition_gen.py generate-key --key_protect_hmac --kp_hmac_
↪keygen --kp_hmac_keyfile hmac_key.bin --keyfile nvs_encr_key.bin
```

This shall generate the respective keys in the `keys` folder.

2. Burn the HMAC key in the eFuse

   The NVS key can be burned in the eFuse of ESP32-P4 with help of following command:

```
espefuse.py --port PORT burn_key BLOCK hmac_key.bin HMAC_UP
```

   where `BLOCK` is a free keyblock between `BLOCK_KEY0` and `BLOCK_KEY5`.

3. Generate the encrypted NVS partition

   We shall generate the actual encrypted NVS partition on host. More details about generating the encryption NVS partition can be found at *生成 NVS 加密分区*. For this purpose, the contents of the NVS file shall be available in a CSV file. Please check out *CSV 文件格式* for more details.

   The encrypted NVS partition can be generated with following command:

```
python3 nvs_partition_gen.py encrypt sample_singlepage_blob.csv nvs_encr_
↪partition.bin 0x3000 --inputkey keys/nvs_encr_key.bin
```

   Some command arguments are explained below:

   - CSV file name - In this case *sample_singlepage_blob.csv* is the CSV file which contains the NVS data, Replace this with the file you wish to choose.
   - NVS partition offset - This is the offset at which that NVS partition shall be stored in the flash of ESP32-P4. The offset of your nvs-partition can be found be executing *idf.py partition-table* in the projtect directory. Please update the sample value of *0x3000* in the above-provided command to the correct offset.

4. Configure the project

- Enable *NVS Encryption* by enabling *CONFIG_NVS_ENCRYPTION*.
- Enable the HMAC based NVS Encryption by setting *CONFIG_NVS_SEC_KEY_PROTECTION_SCHEME* to `CONFIG_NVS_SEC_KEY_PROTECT_USING_HMAC`
- Set the HMAC efuse key id at *CONFIG_NVS_SEC_HMAC_EFUSE_KEY_ID* to the one in which the eFuse key was burned in Step 2.

5. Flash NVS partition

   The NVS partition (`nvs_encr_partition.bin`) generated in Step 3 can then be flashed to its respective offset using `esptool.py`. To see all of the command line options recommended for `esptool.py`, check the output printed when `idf.py build` succeeds. If Flash encryption is enabled for the chip then please encrypt the partition first before flashing. You may refer the flashing related steps of *Flash Encryption workflow*.

Enable NVS Encryption based on Flash Encryption In this case we generate NVS Encryption keys on a host. This key is then flashed on the chip and protected with help of the *Flash Encryption* feature.

1. Generate the NVS Encryption key

   For generation of respective keys, we shall use *NVS partition generator utility*. We shall generate the encryption key on host and this key key shall be stored on the flash of ESP32-P4 in encrypted state.

   The key can be generated with the nvs_flash/nvs_partition_generator/nvs_partition_gen.py script with help of the following command:

```
python3 nvs_partition_gen.py generate-key --keyfile nvs_encr_key.bin
```

This shall generate the respective key in the `keys` folder.

2. Generate the encrypted NVS partition

   We shall generate the actual encrypted NVS partition on host. More details about generating the encryption NVS partition can be found at *生成 NVS 加密分区*. For this, the contents of the NVS file shall be available in a CSV file. Please refer *CSV 文件格式* for more details.

   The encrypted NVS partition can be generated with following command:

```
python3 nvs_partition_gen.py encrypt sample_singlepage_blob.csv nvs_encr_
↪partition.bin 0x3000 --inputkey keys/nvs_encr_key.bin
```

   Some command arguments are explained below:

   - CSV file name - In this case *sample_singlepage_blob.csv* is the CSV file which contains the NVS data, Replace this with the file you wish to choose.
   - NVS partition offset - This is the offset at which that NVS partition shall be stored in the flash of ESP32-P4. The offset of your nvs-partition can be found be executing *idf.py partition-table* in the projtect directory. Please update the sample value of *0x3000* in the above-provided command to the correct offset.

3. Configure the project

- Enable *NVS Encryption* by enabling *CONFIG_NVS_ENCRYPTION*.
- Set NVS to use Flash Encryption based scheme by setting *CONFIG_NVS_SEC_KEY_PROTECTION_SCHEME* to `CONFIG_NVS_SEC_KEY_PROTECT_USING_FLASH_ENC`.

4. Flash NVS partition and NVS Encryption keys

   The NVS partition (`nvs_encr_partition.bin`) and NVS Encryption key (`nvs_encr_key.bin`) can then be flashed to their respective offset using `esptool.py`. To see all of the command line options recommended for `esptool.py`, check the output printed when `idf.py build` succeeds. If Flash encryption is enabled for the chip then please encrypt the partition first before flashing. You may refer the flashing related steps of *Flash Encryption workflow*.

Chapter 7

库与框架

7.1 云框架

ESP32-P4 可以使用构建在 ESP-IDF 之上的代理来支持多个云框架。以下是各种支持的云框架代理和示例：

7.1.1 ESP RainMaker

ESP RainMaker 是用于加速 AIoT 开发的完整解决方案。更多信息见 ESP RainMaker GitHub 仓库。

7.1.2 AWS IoT

https://github.com/espressif/esp-aws-iot 是 ESP32-P4 的开源仓库，基于亚马逊 Web 服务的 aws-iot-device-sdk-embedded-C。

7.1.3 Azure IoT

https://github.com/espressif/esp-azure 是 ESP32-P4 的开源存储库，基于微软 Azure 的 azure-iot-sdk-c SDK。

7.1.4 Google IoT Core

https://github.com/espressif/esp-google-iot 是 ESP32-P4 的开源存储库，基于谷歌的 iot-device-sdk-embedded-c SDK。

7.1.5 阿里云 IoT

https://github.com/espressif/esp-aliyun 是 ESP32-P4 的开源存储库，基于阿里云的 iotkit-embedded SDK。

7.1.6 Joylink IoT

https://github.com/espressif/esp-joylink 是 ESP32-P4 的开源存储库，基于 Joylink 的 joylink_dev_sdk SDK。

7.1.7 腾讯 IoT

https://github.com/espressif/esp-welink 是 ESP32-P4 的开源存储库，基于腾讯的 welink SDK。

7.1.8 腾讯云 IoT

https://github.com/espressif/esp-qcloud 是 ESP32-P4 的开源存储库，基于腾讯云的 qcloud-iot-sdk-embedded-c SDK。

7.1.9 百度 IoT

https://github.com/espressif/esp-baidu-iot 是 ESP32-P4 的开源存储库，基于百度的 iot-sdk-c SDK。

7.2 其他库和开发框架

本文展示了一系列乐鑫官方发布的库和框架。

7.2.1 ESP-ADF

ESP-ADF 是一个全方位的音频应用程序框架，该框架支持：

- CODEC 的 HAL
- 音乐播放器和录音机
- 音频处理
- 蓝牙扬声器
- 互联网收音机
- 免提设备
- 语音识别

该框架对应的 GitHub 仓库为 ESP-ADF。

7.2.2 ESP-CSI

ESP-CSI 是一个具有实验性的框架，它利用 Wi-Fi 信道状态信息来检测人体存在。

该框架对应的 GitHub 仓库为 ESP-CSI。

7.2.3 ESP-DSP

ESP-DSP 提供了针对数字信号处理应用优化的算法，该库支持：

- 矩阵乘法
- 点积
- 快速傅立叶变换 (FFT)
- 无限脉冲响应 (IIR)
- 有限脉冲响应 (FIR)
- 向量数学运算

该库对应的 GitHub 仓库为 ESP-DSP 库。

7.2.4 ESP-WIFI-MESH

ESP-WIFI-MESH 基于 ESP-WIFI-MESH 协议搭建，该框架支持：

- 快速网络配置
- 稳定升级
- 高效调试
- LAN 控制
- 多种应用示例

该框架对应的 GitHub 仓库为 ESP-MDF。

7.2.5 ESP-WHO

ESP-WHO 框架利用 ESP32 及摄像头实现人脸检测及识别。

该框架对应的 GitHub 仓库为 ESP-WHO。

7.2.6 ESP RainMaker

ESP RainMaker 提供了一个快速 AIoT 开发的完整解决方案。使用 ESP RainMaker，用户可以创建多种 AIoT 设备，包括固件 AIoT 以及集成了语音助手、手机应用程序和云后端的 AIoT 等。

该解决方案对应的 GitHub 仓库为 GitHub 上的 ESP RainMaker。

7.2.7 ESP-IoT-Solution

ESP-IoT-Solution 涵盖了开发 IoT 系统时常用的设备驱动程序及代码框架。在 ESP-IoT-Solution 中，设备驱动程序和代码框架以独立组件存在，可以轻松地集成到 ESP-IDF 项目中。

ESP-IoT-Solution 支持：

- 传感器、显示器、音频、GUI、输入、执行器等设备驱动程序
- 低功耗、安全、存储等框架和文档
- 从实际应用角度指导乐鑫开源解决方案

该解决方案对应的 GitHub 仓库为 GitHub 上的 ESP-IoT-Solution。

7.2.8 ESP-Protocols

ESP-Protocols 库包含 ESP-IDF 的协议组件集。ESP-Protocols 中的代码以独立组件存在，可以轻松地集成到 ESP-IDF 项目中。此外，每个组件都可以在 ESP-IDF 组件注册表 中找到。

ESP-Protocols 组件：

- esp_modem 使用 AT 命令或 PPP 协议与 GSM/LTE 调制解调器连接，详情请参阅 esp_modem 文档。
- mdns (mDNS) 是一种组播 UDP 服务，用于提供本地网络服务与主机发现，详情请参阅 mdns 文档。
- esp_websocket_client 是 ESP-IDF 的托管组件，可在 ESP32 上实现 WebSocket 协议客户端，详情请参阅 esp_websocket_client 文档。有关 WebSocket 协议客户端，请参阅 WebSocket_protocol_client。
- asio 是一个跨平台的 C++ 库，请参阅 https://think-async.com/Asio/。该库基于现代 C++ 提供一致的异步模型，请参阅 asio 文档。
- esp_wifi_remote 是一个 Wi-Fi 通信库，提供标准的 Wi-Fi API，并且能够借助指定的传输接口，帮助目标设备通过外部 ESP32 芯片实现 Wi-Fi 通信。详情请参阅*Wi-Fi Expansion*。

7.2.9 ESP-BSP

ESP-BSP 库包含了各种乐鑫和第三方开发板的板级支持包 (BSP)，可以帮助快速上手特定的开发板。它们通常包含管脚定义和辅助函数，这些函数可用于初始化特定开发板的外设。此外，BSP 还提供了一些驱动程序，可用于开发版上的外部芯片，如传感器、显示屏、音频编解码器等。

7.2.10 ESP-IDF-CXX

ESP-IDF-CXX 包含了 ESP-IDF 的部分 C++ 封装，重点在实现易用性、安全性、自动资源管理，以及将错误检查转移到编译过程中，以避免运行时失败。它还提供了 ESP 定时器、I2C、SPI、GPIO 等外设或 ESP-IDF 其他功能的 C++ 类。ESP-IDF-CXX 作为组件可以从 组件注册表 中获取。详情请参阅 README.md。

Chapter 8

贡献指南

欢迎为 ESP-IDF 项目贡献内容！

8.1 如何贡献

欢迎为 ESP-IDF 贡献内容，如修复问题、新增功能、添加文档等。你可通过 Github Pull Requests 提交你的贡献内容。

8.2 准备工作

在提交 Pull Request 前，请检查以下要点：

- 贡献内容是否完全是自己的成果，或已获得与 Apache License 2.0 兼容的开源许可？如果不是，我们不能接受该内容。了解更多信息，请见版权标头指南。
- 要提交的代码是否符合 ESP-IDF *Espressif IoT Development Framework Style Guide*?
- 是否安装了 ESP-IDF *pre-commit* 钩子?
- 代码文档是否符合编写文档 的要求?
- 代码是否注释充分，便于读者理解其结构?
- 是否为贡献的代码提供文档或示例？要写出好的示例，请参考 examples readme。
- 注释或文档是否以英语书写并表达清晰，不存在拼写或语法错误?
- 欢迎贡献新的代码示例。了解更多信息，请参考创建示例项目 。
- 如果需提交多个内容，是否将所有内容按照改动的类型（每个 pull request 对应一个主要改动）进行分组？是否有命名类似 “fixed typo” 的提交 压缩到了此前的提交中?
- 如不能确定上述任意内容，请提交 Pull Request，并向我们寻求反馈。

8.3 Pull Request 提交流程

创建 Pull Request 后，PR 评论区中可能有一些关于该请求的讨论。

Pull Request 准备好待合并时，首先会合并到我们的内部 git 系统中进行内部自动化测试。

测试流程通过后，你贡献的内容将合并到公共 GitHub 库。

8.4 法律规范

在提交贡献内容前，你需签署贡献者协议。该协议将在 Pull Request 过程中自动推送。

8.5 相关文档

8.5.1 Espressif IoT Development Framework Style Guide

About This Guide

Purpose of this style guide is to encourage use of common coding practices within the ESP-IDF.

Style guide is a set of rules which are aimed to help create readable, maintainable, and robust code. By writing code which looks the same way across the code base we help others read and comprehend the code. By using same conventions for spaces and newlines we reduce chances that future changes will produce huge unreadable diffs. By following common patterns for module structure and by using language features consistently we help others understand code behavior.

We try to keep rules simple enough, which means that they can not cover all potential cases. In some cases one has to bend these simple rules to achieve readability, maintainability, or robustness.

When doing modifications to third-party code used in ESP-IDF, follow the way that particular project is written. That will help propose useful changes for merging into upstream project.

C Code Formatting

Naming

- Any variable or function which is only used in a single source file should be declared `static`.
- Public names (non-static variables and functions) should be namespaced with a per-component or per-unit prefix, to avoid naming collisions. ie `esp_vfs_register()` or `esp_console_run()`. Starting the prefix with `esp_` for Espressif-specific names is optional, but should be consistent with any other names in the same component.
- Static variables should be prefixed with `s_` for easy identification. For example, `static bool s_invert`.
- Avoid unnecessary abbreviations (ie shortening `data` to `dat`), unless the resulting name would otherwise be very long.

Indentation Use 4 spaces for each indentation level. Do not use tabs for indentation. Configure the editor to emit 4 spaces each time you press tab key.

Vertical Space Place one empty line between functions. Do not begin or end a function with an empty line.

```
void function1()
{
    do_one_thing();
    do_another_thing();
                                    // INCORRECT, do not place empty line here
}
                                    // place empty line here
void function2()
{
                                    // INCORRECT, do not use an empty line here
    int var = 0;
    while (var < SOME_CONSTANT) {
        do_stuff(&var);
    }
}
```

The maximum line length is 120 characters as long as it does not seriously affect the readability.

Horizontal Space Always add single space after conditional and loop keywords:

```
if (condition) {     // correct
    // ...
}

switch (n) {         // correct
    case 0:
        // ...
}

for(int i = 0; i < CONST; ++i) {     // INCORRECT
    // ...
}
```

Add single space around binary operators. No space is necessary for unary operators. It is okay to drop space around multiply and divide operators:

```
const int y = y0 + (x - x0) * (y1 - y0) / (x1 - x0);     // correct

const int y = y0 + (x - x0)*(y1 - y0)/(x1 - x0);         // also okay

int y_cur = -y;                                          // correct
++y_cur;

const int y = y0+(x-x0)*(y1-y0)/(x1-x0);                 // INCORRECT
```

No space is necessary around `.` and `->` operators.

Sometimes adding horizontal space within a line can help make code more readable. For example, you can add space to align function arguments:

```
esp_rom_gpio_connect_in_signal(PIN_CAM_D6,   I2S0I_DATA_IN14_IDX, false);
esp_rom_gpio_connect_in_signal(PIN_CAM_D7,   I2S0I_DATA_IN15_IDX, false);
esp_rom_gpio_connect_in_signal(PIN_CAM_HREF, I2S0I_H_ENABLE_IDX,  false);
esp_rom_gpio_connect_in_signal(PIN_CAM_PCLK, I2S0I_DATA_IN15_IDX, false);
```

Note however that if someone goes to add new line with a longer identifier as first argument (e.g., `PIN_CAM_VSYNC`), it will not fit. So other lines would have to be realigned, adding meaningless changes to the commit.

Therefore, use horizontal alignment sparingly, especially if you expect new lines to be added to the list later.

Never use TAB characters for horizontal alignment.

Never add trailing whitespace at the end of the line.

Braces

- Function definition should have a brace on a separate line:

```
// This is correct:
void function(int arg)
{

}

// NOT like this:
void function(int arg) {

}
```

- Within a function, place opening brace on the same line with conditional and loop statements:

```
if (condition) {
    do_one();
} else if (other_condition) {
    do_two();
}
```

Comments Use `//` for single line comments. For multi-line comments it is okay to use either `//` on each line or a `/* */` block.

Although not directly related to formatting, here are a few notes about using comments effectively.

- Do not use single comments to disable some functionality:

```
void init_something()
{
    setup_dma();
    // load_resources();                // WHY is this thing commented, asks␣
↪the reader?
    start_timer();
}
```

- If some code is no longer required, remove it completely. If you need it you can always look it up in git history of this file. If you disable some call because of temporary reasons, with an intention to restore it in the future, add explanation on the adjacent line:

```
void init_something()
{
    setup_dma();
    // TODO: we should load resources here, but loader is not fully integrated␣
↪yet.
    // load_resources();
    start_timer();
}
```

- Same goes for `#if 0 ... #endif` blocks. Remove code block completely if it is not used. Otherwise, add comment explaining why the block is disabled. Do not use `#if 0 ... #endif` or comments to store code snippets which you may need in the future.
- Do not add trivial comments about authorship and change date. You can always look up who modified any given line using git. E.g., this comment adds clutter to the code without adding any useful information:

```
void init_something()
{
    setup_dma();
    // XXX add 2016-09-01
    init_dma_list();
    fill_dma_item(0);
    // end XXX add
    start_timer();
}
```

Line Endings Commits should only contain files with LF (Unix style) endings.

Windows users can configure git to check out CRLF (Windows style) endings locally and commit LF endings by setting the `core.autocrlf` setting. *Github has a document about setting this option <github-line-endings>.*

If you accidentally have some commits in your branch that add LF endings, you can convert them to Unix by running this command in an MSYS2 or Unix terminal (change directory to the IDF working directory and check the correct branch is currently checked out, beforehand):

```
git rebase --exec 'git diff-tree --no-commit-id --name-only -r HEAD | xargs␣
↪dos2unix && git commit -a --amend --no-edit --allow-empty' master
```

(Note that this line rebases on master, change the branch name at the end to rebase on another branch.)

For updating a single commit, it is possible to run `dos2unix FILENAME` and then run `git commit --amend`

Formatting Your Code ESP-IDF uses Astyle to format source code. The configuration is stored in tools/ci/astyle-rules.yml file.

Initially, all components are excluded from formatting checks. You can enable formatting checks for the component by removing it from `components_not_formatted_temporary` list. Then run:

```
pre-commit run --files <path_to_files> astyle_py
```

Alternatively, you can run `astyle_py` manually. You can install it with `pip install astyle_py==VERSION`. Make sure you have the same version installed as the one specified in .pre-commit-config.yaml file. With `astyle_py` installed, run:

```
astyle_py --rules=$IDF_PATH/tools/ci/astyle-rules.yml <path-to-file>
```

Type Definitions Should be snake_case, ending with _t suffix:

```
typedef int signed_32_bit_t;
```

Enum Enums should be defined through the *typedef* and be namespaced:

```
typedef enum
{
    MODULE_FOO_ONE,
    MODULE_FOO_TWO,
    MODULE_FOO_THREE
} module_foo_t;
```

Assertions The standard C `assert()` function, defined in `assert.h` should be used to check conditions that should be true in source code. In the default configuration, an assert condition that returns `false` or 0 will call `abort()` and trigger a *Fatal Error*.

`assert()` should only be used to detect unrecoverable errors due to a serious internal logic bug or corruption, where it is not possible for the program to continue. For recoverable errors, including errors that are possible due to invalid external input, an *error value should be returned*.

备注： When asserting a value of type `esp_err_t` is equal to `ESP_OK`, use the *ESP_ERROR_CHECK 宏* instead of an `assert()`.

It is possible to configure ESP-IDF projects with assertions disabled (see *CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL*). Therefore, functions called in an `assert()` statement should not have side-effects.

It is also necessary to use particular techniques to avoid ”variable set but not used” warnings when assertions are disabled, due to code patterns such as:

```
int res = do_something();
assert(res == 0);
```

Once the `assert` is optimized out, the `res` value is unused and the compiler will warn about this. However the function `do_something()` must still be called, even if assertions are disabled.

When the variable is declared and initialized in a single statement, a good strategy is to cast it to `void` on a new line. The compiler will not produce a warning, and the variable can still be optimized out of the final binary:

```
int res = do_something();
assert(res == 0);
(void)res;
```

If the variable is declared separately, for example if it is used for multiple assertions, then it can be declared with the GCC attribute `__attribute__((unused))`. The compiler will not produce any unused variable warnings, but the variable can still be optimized out:

```
int res __attribute__((unused));

res = do_something();
assert(res == 0);

res = do_something_else();
assert(res != 0);
```

Header File Guards

All public facing header files should have preprocessor guards. A pragma is preferred:

```
#pragma once
```

over the following pattern:

```
#ifndef FILE_NAME_H
#define FILE_NAME_H
...
#endif // FILE_NAME_H
```

In addition to guard macros, all C header files should have `extern "C"` guards to allow the header to be used from C++ code. Note that the following order should be used: `pragma once`, then any `#include` statements, then `extern "C"` guards:

```
#pragma once

#include <stdint.h>

#ifdef __cplusplus
extern "C" {
#endif

/* declarations go here */

#ifdef __cplusplus
}
#endif
```

Include Statements

When writing `#include` statements, try to maintain the following order:

- C standard library headers.
- Other POSIX standard headers and common extensions to them (such as `sys/queue.h`.)
- Common IDF headers (`esp_log.h`, `esp_system.h`, `esp_timer.h`, `esp_sleep.h`, etc.)
- Headers of other components, such as FreeRTOS.
- Public headers of the current component.
- Private headers.

Use angle brackets for C standard library headers and other POSIX headers (`#include <stdio.h>`).

Use double quotes for all other headers (`#include "esp_log.h"`).

C++ Code Formatting

The same rules as for C apply. Where they are not enough, apply the following rules.

File Naming C++ Header files have the extension `.hpp`. C++ source files have the extension `.cpp`. The latter is important for the compiler to distinguish them from normal C source files.

Naming

- **Class and struct** names shall be written in `CamelCase` with a capital letter as beginning. Member variables and methods shall be in `snake_case`. An exception from `CamelCase` is if the readability is severely decreased, e.g., in `GPIOOutput`, then an underscore _ is allowed to make it more readable: `GPIO_Output`.
- **Namespaces** shall be in lower `snake_case`.
- **Templates** are specified in the line above the function declaration.
- Interfaces in terms of Object-Oriented Programming shall be named without the suffix `...Interface`. Later, this makes it easier to extract interfaces from normal classes and vice versa without making a breaking change.

Member Order in Classes In order of precedence:

- First put the public members, then the protected, then private ones. Omit public, protected or private sections without any members.
- First put constructors/destructors, then member functions, then member variables.

For example:

```
class ForExample {
public:
    // first constructors, then default constructor, then destructor
    ForExample(double example_factor_arg);
    ForExample();
    ~ForExample();

    // then remaining pubic methods
    set_example_factor(double example_factor_arg);

    // then public member variables
    uint32_t public_data_member;

private:
    // first private methods
    void internal_method();

    // then private member variables
    double example_factor;
};
```

Spacing

- Do not indent inside namespaces.
- Put `public`, `protected` and `private` labels at the same indentation level as the corresponding `class` label.

Simple Example

```
// file spaceship.h
#ifndef SPACESHIP_H_
#define SPACESHIP_H_
#include <cstdlib>

namespace spaceships {

class SpaceShip {
public:
    SpaceShip(size_t crew);
    size_t get_crew_size() const;

private:
    const size_t crew;
};

class SpaceShuttle : public SpaceShip {
public:
    SpaceShuttle();
};

class Sojuz : public SpaceShip {
public:
    Sojuz();
};

template <typename T>
class CargoShip {
public:
    CargoShip(const T &cargo);

private:
    T cargo;
};

} // namespace spaceships

#endif // SPACESHIP_H_

// file spaceship.cpp
#include "spaceship.h"

namespace spaceships {

// Putting the curly braces in the same line for constructors is OK if it only␣
↪initializes
// values in the initializer list
SpaceShip::SpaceShip(size_t crew) : crew(crew) { }

size_t SpaceShip::get_crew_size() const
{
    return crew;
}

SpaceShuttle::SpaceShuttle() : SpaceShip(7)
{
    // doing further initialization
}

Sojuz::Sojuz() : SpaceShip(3)
{
```

(下页继续)

(续上页)

```
    // doing further initialization
}

template <typename T>
CargoShip<T>::CargoShip(const T &cargo) : cargo(cargo) { }

} // namespace spaceships
```

CMake Code Style

- Indent with four spaces.
- Maximum line length 120 characters. When splitting lines, try to focus on readability where possible (for example, by pairing up keyword/argument pairs on individual lines).
- Do not put anything in the optional parentheses after `endforeach()`, `endif()`, etc.
- Use lowercase (`with_underscores`) for command, function, and macro names.
- For locally scoped variables, use lowercase (`with_underscores`).
- For globally scoped variables, use uppercase (`WITH_UNDERSCORES`).
- Otherwise follow the defaults of the cmake-lint project.

Configuring the Code Style for a Project Using EditorConfig

EditorConfig helps developers define and maintain consistent coding styles between different editors and IDEs. The EditorConfig project consists of a file format for defining coding styles and a collection of text editor plugins that enable editors to read the file format and adhere to defined styles. EditorConfig files are easily readable and they work nicely with version control systems.

For more information, see EditorConfig Website.

Third Party Component Code Styles

ESP-IDF integrates a number of third party components where these components may have differing code styles.

FreeRTOS The code style adopted by FreeRTOS is described in the FreeRTOS style guide. Formatting of FreeRTOS source code is automated using Uncrustify, thus a copy of the FreeRTOS code style's Uncrustify configuration (`uncrustify.cfg`) is stored within ESP-IDF FreeRTOS component.

If a FreeRTOS source file is modified, the updated file can be formatted again by following the steps below:

1. Ensure that Uncrustify (v0.69.0) is installed on your system
2. Run the following command on the update FreeRTOS source file (where `source.c` is the path to the source file that requires formatting).

```
uncrustify -c $IDF_PATH/components/freertos/FreeRTOS-Kernel/uncrustify.cfg --
↪replace source.c --no-backup
```

Documenting Code

Please see the guide here: 编写文档.

Structure

To be written.

Language Features

To be written.

8.5.2 为 ESP-IDF 安装 pre-commit 钩子

环境依赖

我们向 IDF 开发人员推荐 Python 3.8.* 及以上版本。

如果你已安装了不兼容的 Python 版本，应在安装 pre-commit 工具前进行更新。

安装 pre-commit 工具

运行 `pip install pre-commit` 。

安装 pre-commit 钩子

1. 切换到 IDF 项目路径。
2. 运行 `pre-commit install --allow-missing-config -t pre-commit -t commit-msg` 命令。使用这种方式安装钩子后，即使在没有 `.pre-commit-config.yaml` 的分支中也能成功提交。
3. 在运行 `git commit` 命令时，pre-commit 钩子会自动运行。

卸载 pre-commit 钩子

运行 `pre-commit uninstall` 。

更多

更多详细使用方法，请参考 pre-commit 文档。

Windows 用户常见问题

`/usr/bin/env: python: Permission denied.`

> 如果使用 Git Bash，请运行 `which python` 检查 Python 的可执行位置。
>
> 如果该可执行文件位于 `~/AppData/Local/Microsoft/WindowsApps/`，这其实是一个到 Windows 应用商店的链接，而不是真正的可执行文件。
>
> 请手动安装 Python，并在 `PATH` 环境变量中进行更新。

你的 `USERPROFILE` 中包含非 ASCII 字符

> 如果 `pre-commit` 的缓存路径包含非 ASCII 字符，用 `pre-commit` 为特定钩子初始化环境时可能会失败。解决方案是，将 `PRE_COMMIT_HOME` 设置为一个仅包含标准字符的路径，然后再运行 pre-commit。
>
> - CMD: `set PRE_COMMIT_HOME=C:\somepath\pre-commit`
> - PowerShell: `$Env:PRE_COMMIT_HOME = "C:\somepath\pre-commit"`
> - git bash: `export PRE_COMMIT_HOME="/c/somepath/pre-commit"`

8.5.3 编写文档

本文档简要总结了 espressif/esp-idf 库中的文档风格，并介绍了如何添加新文档。

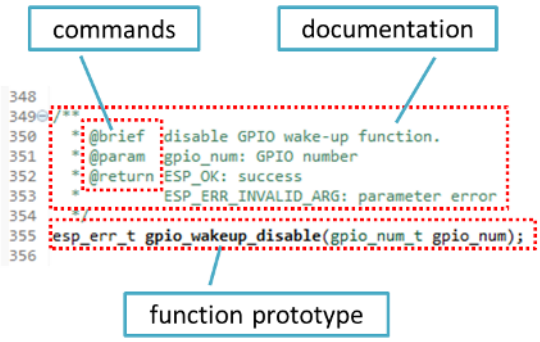
概述

为本仓库代码编写文档时，请遵循 Doxygen style，即在标准注释块中插入特殊命令，如 `@param`：

```
/**
 * @param ratio this is oxygen to air ratio
 */
```

Doxygen 能够解析代码，提取此类特殊命令及其后续文本，并基于提取的信息构建文档。

下图为一个包含函数文档的典型注释块：

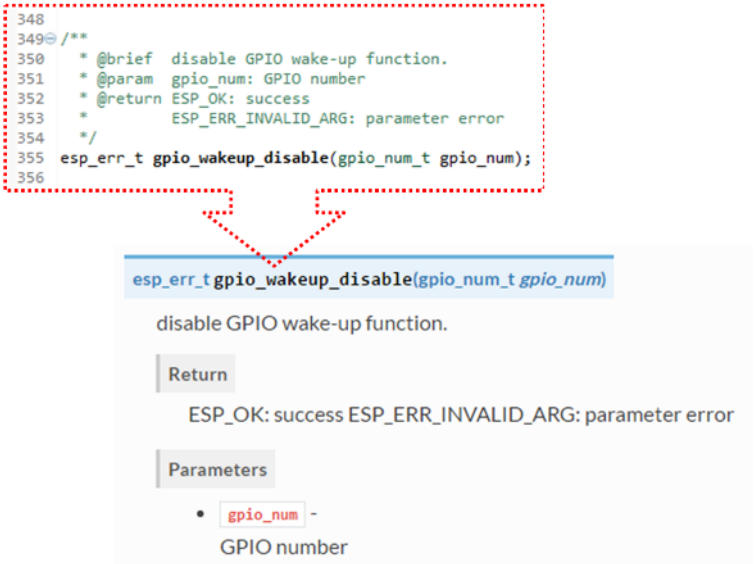


Doxygen 支持多种格式，并支持文档内部的多个详情级别，具有很强的灵活性。要了解更多可用功能，请参考 Doxygen Manual。

为什么要用 Doxygen?

我们的最终目的是保证代码文档的一致性。因此，我们可以使用 Sphinx 和 Breathe 等工具，在代码变更时自动更新 API 文档。

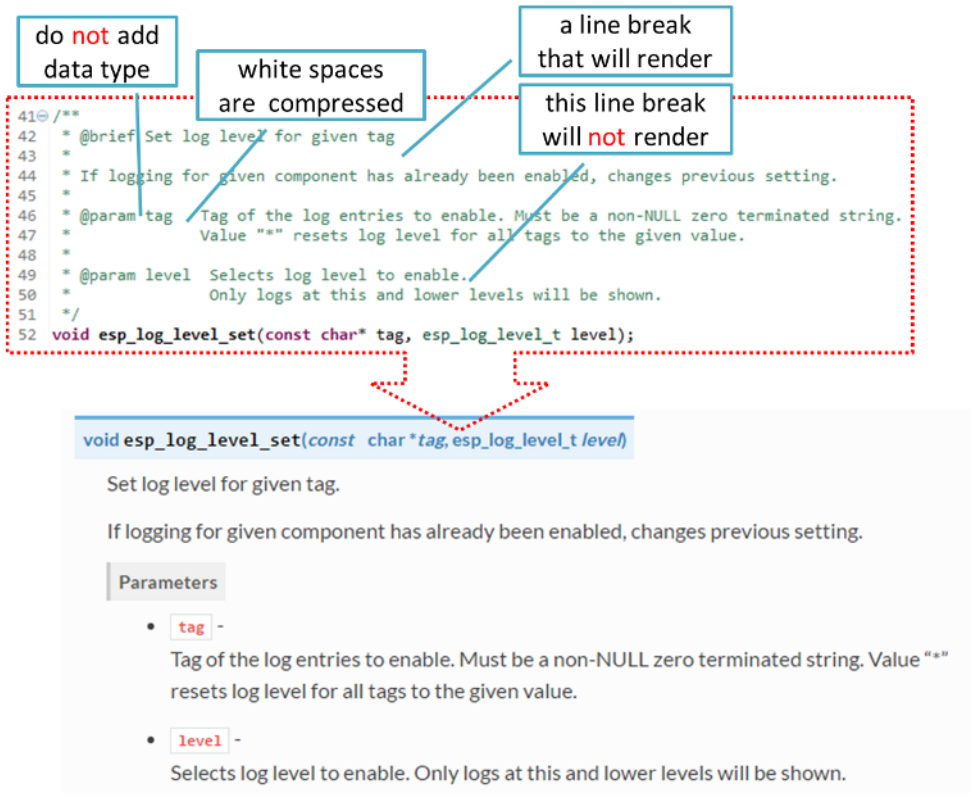
借助这些工具，以上代码的渲染效果如下：



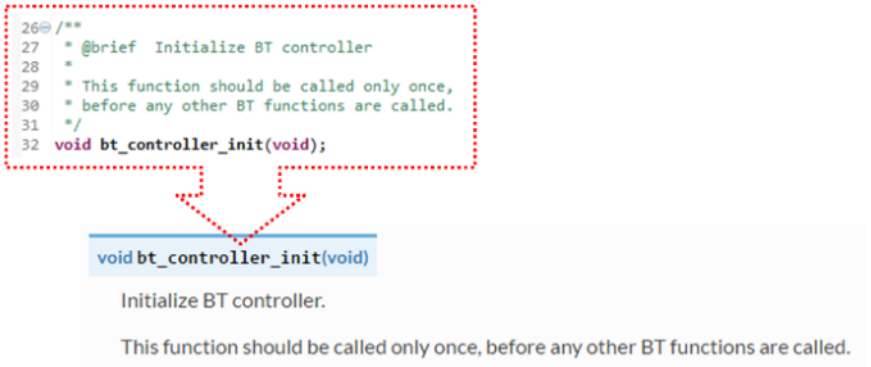
试一试！

为 ESP-IDF 库添加代码时，请遵循以下标准：

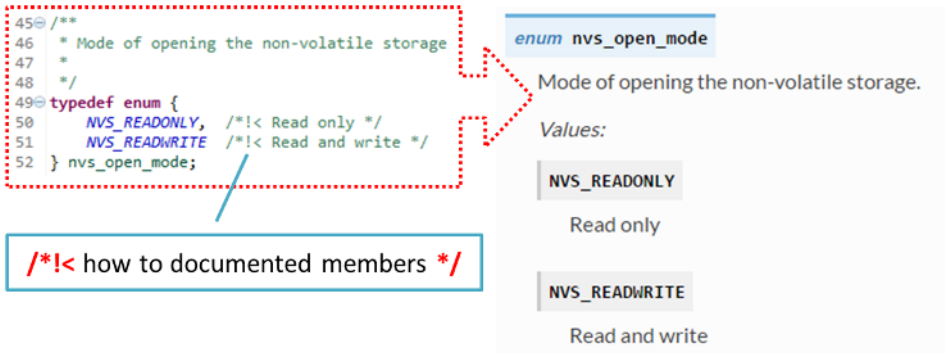
1. 为代码的所有构件提供文档说明，包括函数、结构体、类型定义、枚举、宏等。对其目的、功能和局限性进行充分介绍，提供符合读者预期的良好体验。
2. 函数文档应说明该函数的作用。如果函数接受输入参数并返回值，也需要对输入参数和返回值进行解释。
3. 请勿在参数前添加数据类型或除空格外的任何字符。所有空格和换行符都会压缩为一个空格。如需换行，请换行两次。



4. 在函数输入为空或不返回值时，无需插入 `@param` 或 `@return` 命令。



5. 为 `define` `struct` 或 `enum` 的成员撰写文档时，需遵循以下格式在每个成员后添加注释。



6. 要渲染出整齐的列表，可在命令后换行（如以下代码示例中的 `@return` 命令）。

```
*
* @return
*     - ESP_OK if erase operation was successful
*     - ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
*     - ESP_ERR_NVS_READ_ONLY if handle was opened as read only
*     - ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
*     - other error codes from the underlying storage driver
```

7. 头文件或一组文件库的功能概述需在同一路径下单独的 `README.rst` 文档中进行描述。如该路径下还有其他 API 的头文件，则应将此头文件对应的 README 文件命名为 `apiname-readme.rst`。

进阶用法

也可以使用以下进阶技巧，产出更加优质使用的文档。

在编写代码时，请遵循以下标准：

1. 使用 `@code{c}` 和 `@endcode` 命令添加代码示例片段，阐述实现过程。

```
*
* @code{c}
* // Example of using nvs_get_i32:
* int32_t max_buffer_size = 4096; // default value
* esp_err_t err = nvs_get_i32(my_handle, "max_buffer_size", &max_buffer_size);
* assert(err == ESP_OK || err == ESP_ERR_NVS_NOT_FOUND);
* // if ESP_ERR_NVS_NOT_FOUND was returned, max_buffer_size will still
* // have its default value.
* @endcode
*
```

   该代码片段应添加在其所说明的函数的注释区中。

2. 要高亮重点信息，可使用 `@attention` 或 `@note` 命令。

```
*
* @attention
*     1. This API only impact WIFI_MODE_STA or WIFI_MODE_APSTA mode
*     2. If the ESP32 is connected to an AP, call esp_wifi_disconnect to␣
↪disconnect.
*
```

   以上示例也展示了有序列表的用法。

3. 要描述一组具有相似功能的函数，可使用 `/**@{*/` 和 `/**@}*/` 标记命令。

```
/**@{*/
/**
 * @brief common description of similar functions
 *
 */
```

(下页继续)

(续上页)

```
void  first_similar_function (void);
void second_similar_function (void);
/**@}*/
```

如需更多应用示例，请参考 nvs_flash/include/nvs.h。

4. 如需进一步跳过重复定义或枚举等代码，可使用 `/** @cond */` 和 `/** @endcond */` 命令附上该代码。相关应用实例，请参考 esp_driver_gpio/include/driver/gpio.h。
5. 使用 markdown 添加标题、链接和表格等，增强文档的可读性。

```
*
* [ESP32-P4 Technical Reference Manual](https://www.espressif.com/sites/
↪default/files/documentation/esp32-p4_technical_reference_manual_en.pdf)
*
```

备注： 如果没有将代码片段、说明或链接等内容包含在一个文档对象相关联的注释块中，则文档中不会出现相关内容。

6. 为一个或多个完整代码示例提供说明。将说明写入单独的 `README.md` 文件中并放在 examples 路径下对应文件夹。

文档格式标准化

撰写代码文档文本时，请遵循以下标准，提供格式良好的 Markdown (.md) 或 reST (.rst) 文档。

1. 确保将一个段落写在同一行。通过换行加强可读性仅适用于代码，请勿在文本中以下图形式换行。为方便阅读，建议在段落之间空一行。

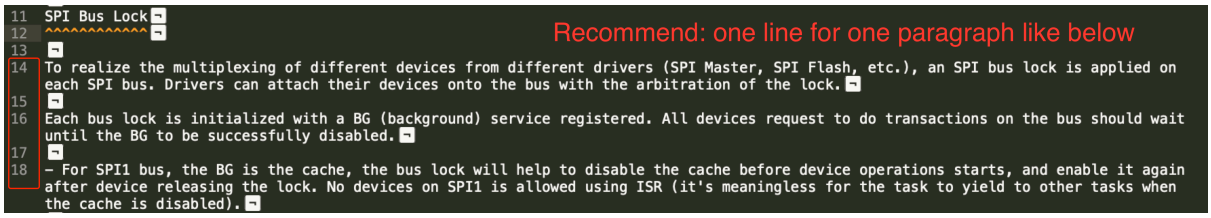


图 1: 一个段落写在同一行（点击放大）

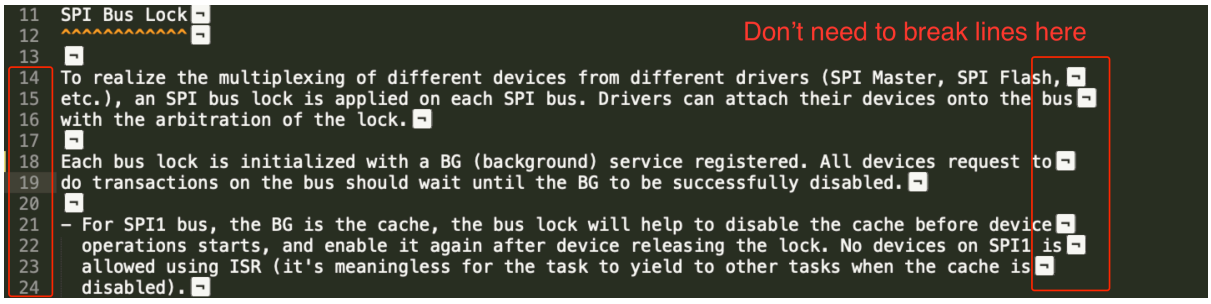


图 2: 同一段落中请勿换行（点击放大）

2. 中英文文档行号需对齐，如下所示。这样做的好处是为作者和译者节省时间。非双语作者更新文档时，仅需更新对应中文或英文文档的同一行。对译者来说，如果英文文档出现更新，可快速定位对应中文文档需更新的位置。另外，通过比较英文和中文文档的总行数，可以快速检查文档的中文版本是否落后于英文版本。

图 3: 对齐中英文文档行号（点击放大）

构建文档

文档由基于 Sphinx 的 Python 包 *esp-docs* 进行构建。

安装命令:

```
pip install esp-docs
```

安装成功后，使用如下命令在 docs 文件夹中构建文档:

```
build-docs build
```

或使用以下命令指定目标芯片和语言:

```
build-docs -t esp32 -l en build
```

如需深入了解 *esp-docs* 的功能，请参考 esp-docs。

小结

出色的代码可以实现令人赞叹的功能，精心编写的文档则让开发者们如虎添翼。

期待你的贡献！

相关文档

- *API* 文档模板

8.5.4 创建示例项目

每个 ESP-IDF 的示例都是一个完整的项目，其他人可以将示例复制至本地，并根据实际情况进行一定修改。请注意，示例项目主要是为了展示 ESP-IDF 的功能。

示例项目结构

- `main` 目录需要包含一个名为 `(something)_example_main.c` 的源文件，里面包含示例项目的主要功能。

- 如果该示例项目的子任务比较多，请根据逻辑将其拆分为 `main` 目录下的多个 C 或者 C++ 源文件，并将对应的头文件也放在同一目录下。
- 如果该示例项目具有多种功能，可以考虑在项目中增加一个 `components` 子目录，通过库功能，将示例项目的不同功能划分为不同的组件。注意，如果该组件提供的功能相对完整，且具有一定的通用性，则应该将它们添加到 ESP-IDF 的 `components` 目录中，使其成为 ESP-IDF 的一部分。
- 示例项目需要包含一个 `README.md` 文件，建议使用 示例项目 README 模板 ，并根据项目实际情况进行修改。
- 示例项目需要包含一个 `example_test.py` 文件，用于进行自动化测试。如果在 GitHub 上初次提交 Pull Request 时，可以先不包含这个脚本文件。具体细节，请见有关 Pull Request 的相关内容。

一般准则

示例代码需要遵循《乐鑫物联网开发框架风格指南》。

检查清单

提交一个新的示例项目之前，需要检查以下内容：

- 每个示例项目只能有一个主要功能。如果某个示例项目有多个主要功能，请将其拆分为两个或更多示例项目。
- 每个示例项目应包含一个 `README.md` 文件，建议使用 示例项目 README 模板 。
- 示例项目中的函数和变量的命令要遵循命名规范 中的要求。对于仅在示例项目源文件中使用的非静态变量/函数，请使用 `example` 或其他类似的前缀。
- 示例项目中的所有代码结构良好，关键代码要有详细注释。
- 示例项目中所有不必要的代码（旧的调试日志，注释掉的代码等）都必须清除掉。
- 示例项目中使用的选项（比如网络名称，地址等）不得直接硬编码，应尽可能地使用配置项，或者定义为宏或常量。
- 配置项可见 `KConfig.projbuild` 文件，该文件中包含一个名为 “Example Configuration” 的菜单。具体情况，请查看现有示例项目。
- 所有的源代码都需要在文件开头指定许可信息（表示该代码是 `in the public domain CC0`）和免责声明。或者，源代码也可以应用 `Apache License 2.0` 许可条款。请查看现有示例项目的许可信息和免责声明，并根据实际情况进行修改。
- 任何第三方代码（无论是直接使用，还是进行了一些改进）均应保留原始代码中的许可信息，且这些代码的许可必须兼容 `Apache License 2.0` 协议。

8.5.5 API 文档模板

备注： 说明

1. 使用此文件 (docs/zh_CN/api-reference/template.rst) 作为 API 参考文档模板。
2. API 参考文档需和 API 的头文件名称保持一致。
3. 使用 `..include::` 从 API 文件夹中添加相应的说明文件。
   - README.rst
   - example.rst
   - ...
4. 可选择在此文件中直接提供描述。
5. 完成后，删除所有的说明信息（类似本说明）和多余的头部信息。

概述

备注： 撰写说明

1. 提供概述，简要说明 API 的用途和使用方法。
2. 必要时提供代码片段，以说明特定函数的功能。
3. 用此 文档 中介绍的方式区分不同的章节标题：
   - # 用于设置各部分，标题上下同时标记
   - * 用于设置章标题，标题上下同时标记
   - = 用于设置节标题
   - - 用于设置小节标题
   - ^ 用于设置小小节标题
   - " 用于设置段落标题

应用示例

备注： 撰写说明

1. 准备一个或多个实际示例，展示此 API 的功能。
2. 每个示例应遵循 `esp-idf/examples/` 文件夹中项目的模式。
3. 将示例放在此文件夹中，添加 `README.md` 文件。
4. 在 `README.md` 中对展示的功能进行概述。
5. 良好的概述让读者能够充分理解示例，而无需参考源代码。
6. 按照示例的复杂程度，将代码描述分解成几个部分，并对每部分的功能进行概述。
7. 必要时，添加流程图和应用程序的输出截图。
8. 最后为本章节的每个示例添加对应链接，示例文件夹应位于 `esp-idf/examples/` 中。

API 参考

备注： 撰写说明

1. ESP-IDF 仓库通过用 *Doxygen 从头文件中检索代码标记* 的方式自动更新 API 参考文档。
2. 通过调用 Sphinx 扩展工具 `esp_extensions/run_doxygen.py`，对 docs/doxygen/Doxyfile 中 `INPUT` 语句列出的所有头文件进行更新。
3. `INPUT` 语句的每一行（以 `##` 开头的注释除外）都包含一个到头文件 `*.h` 的路径，用于生成相应的 `*.inc` 文件：

```
##
## Wi-Fi - API Reference
##
../components/esp32/include/esp_wifi.h \
../components/esp32/include/esp_smartconfig.h \
```

4. 头文件被展开时，`sdkconfig.h` 中默认定义的宏以及在 SOC 特定 `include/soc/*_caps.h` 头文件中定义的宏都会被展开。这样，头文件就可以根据 `IDF_TARGET` 的值来添加或排除相关内容。
5. `*.inc` 文件中包含每次文档构建时自动生成的 API 成员格式化参考。所有 `*.inc` 文件都位于 Sphinx 的 `_build` 路径下。如需查看生成指令，以 `esp_wifi.h` 为例，运行 `python gen-dxd.py esp32/include/esp_wifi.h`。
6. 用以下语句在文档中显示 `*.inc` 文件的内容：

```
.. include-build-file:: inc/esp_wifi.inc
```

   参考示例：docs/en/api-reference/network/esp_wifi.rst
7. 你也可以不用 `*.inc` 文件，而使用自己的方式描述 API。参考示例：docs/en/api-reference/storage/fatfs.rst。
   以下为常见的 `.. doxygen...::` 指令：
   - 函数 - `.. doxygenfunction:: name_of_function`
   - 联合体 - `.. doxygenunion:: name_of_union`
   - 结构体 - `.. doxygenstruct:: name_of_structure` 和 `:members:`

- 宏 - `.. doxygendefine:: name_of_define`
- 类定义 - `.. doxygentypedef:: name_of_type`
- 枚举 - `.. doxygenenum:: name_of_enumeration`

如需更多信息，请参考 Breathe 文档。

使用 *link custom role* 指令添加指向头文件的链接，如下所示:

```
* :component_file:`path_to/header_file.h`
```

8. 在任何情况下，要生成 API 参考文档，应该更新文件 docs/doxygen/Doxyfile 并将其中的路径更新为正在添加文档的 `*.h` 头文件的路径。
9. 更改提交并构建文档后，可以查看文档的渲染效果。如有需要，为相应的头文件纠正注释。

8.5.6 贡献者协议

个人贡献者非独占性许可协议（包含传统专利许可选项）

感谢您以贡献者身份为托管在 GitHub 上的乐鑫项目（以下简称“乐鑫”）做出贡献。

本贡献者协议（以下简称“协议”）旨在将贡献者授予乐鑫的权利明确写入文档。要有效利用这份文档，请遵循贡献指南。

1. 定义 **贡献者**是指向乐鑫提交贡献的个人版权所有者，本文档中称“贡献者”或“您”。如果您以雇员身份提交贡献，且该贡献属于您的部分工作内容，您的雇主须已批准此协议，或作为实体签署此协议。

贡献是指您向乐鑫提交的、您拥有版权的原创作品（软件/文档），包括对既有工作进行的任何修改或补充。如果您没有该原创作品的完整版权，请在 GitHub 上向乐鑫提交评论。

版权是指在原创作品的整个存续期内（包括许可的延期），保护您所拥有或控制的原创作品的所有权利，包括版权、精神权、邻接权等，视具体情况而定。

材料是指乐鑫向第三方提供的软件或文档。当本协议涉及多个软件项目时，材料指您的贡献内容所提交到的软件或文档。在您提交贡献后，此贡献可能被包含在材料中。

提交是指向乐鑫发送的任何形式的实体、电子、或书面交流信息，包括但不限于由乐鑫管理或代表乐鑫管理的电子邮件列表、源代码控制系统和问题跟踪系统，但不包括您以显著方式标记或以其他书面形式指定为“非贡献”的交流内容。

提交日期是指您向乐鑫提交贡献的日期。

文档是指贡献中任何非软件的部分。

2. 版权许可 2.1 向乐鑫授予版权许可

根据本协议的条款和条件，对包含您贡献的版权，您在此授予乐鑫全球范围内免版权授权费、非独占、永久且不可撤销的版权许可，允许乐鑫无限次转让非独占性版权许可或向第三方转许可，以任何方式使用您的贡献，使用方式包括但不限于:

- 公开发布您的贡献
- 修改您的贡献内容，制作基于或包含贡献内容的衍生作品，以及将贡献内容并入其他软件代码中
- 复制原始或修改后的贡献内容
- 分发、公开、展示和公开演示您的原始贡献或修改后贡献

2.2 精神权在其受到承认的范围内不受影响，不受适用法免除。尽管如此，您仍可在贡献的源代码文件头中添加自己的姓名。在使用您的贡献时，乐鑫尊重这一归属。

3. 专利许可 3.1 向乐鑫授予专利许可

根据本协议的条款和条件，您在此授予乐鑫全球范围内免专利授权费、非独占、永久且不可撤销（3.2 节所述情况除外）的专利许可，允许乐鑫无限次转让非独占性专利许可或向第三方转许可，从而制作、委托制作、使用、出售、许诺出售、进口、以其他方式转让贡献以及并入材料的贡献（或此结合产物的一部分）。本许可适用于您现在或此后拥有或控制的所有专利，如未经您的许可而制作、委托制作、使用、出售、许诺出售、进口、或以其他方式转让贡献本身或并入材料中的贡献，都将构成侵权。

3.2 撤回专利许可

如果乐鑫对您的贡献提出任何不出于防卫目的的侵权索赔，您保留撤回专利许可（如 3.1 条所述）的权利。如果某实体已向乐鑫或乐鑫的任何被许可人提出、维持或威胁进行专利侵权诉讼，或已自愿参与专利侵权诉讼，则乐鑫在此情况下向该实体提出的索赔应被视为出于“防卫目的”。

4. 免责声明 贡献以“原样”提供。具体地，所有明示和默示的保证条款，包括但不限于任何默示适销性保证、特定用途适用性和非侵权性保证条款，均由您对乐鑫和乐鑫对您双方互相明确表示免责。在任何此类保证不能被免责的情况下，此类保证的适用期限为法律允许的最短期限。

5. 间接损害豁免 在所适用的法律允许的最大范围内，对于本协议引起的任何利润损失、预期存款损失、数据丢失以及间接性、特殊性、偶然性、后果性和惩戒性损害，无论索赔所依据的法律或衡平法理论（合同、侵权或其他）如何，乐鑫或您在任何情况下均不承担责任。

6. 近似免责声明和损害豁免 如果第 4 节和第 5 节中提及的免责声明和间接损害豁免声明在当地适用法律下不具有法律效力，则复审法院应采用最接近于绝对免除所有与该贡献相关的民事责任的当地法律。

7. 协议期限 7.1 本协议在您接受条款和条件后生效。

7.2 如果本协议终止，第 4、5、6、7、8 条在本协议终止后仍然完全有效。为避免疑义，在本协议终止日前获得自由开源许可的贡献，在本协议终止后仍然完全有效。

8. 其他条款 8.1 本协议以及由本协议引起、或以任何方式与本协议有关的所有争议、索赔、法律行动、诉讼或其他程序，均应受中华人民共和国法律管辖，但不包括其国际私法条款。

8.2 本协议规定了您与乐鑫之间、关于您向乐鑫提供的贡献内容的完整协议，并且此协议的优先级高于其他所有正式或非正式协议。

8.3 如果本协议的任何条款被认定为无效或不可执行条款，该条款将尽量由最接近原始条款含义且可执行的条款取代。即使在本协议的基本目的或有限的补救措施失效的情况下，本协议中规定的条款和条件应仍在法律允许的最大范围内适用。

8.4 如果您了解到任何事实或情况可能影响本协议在某方面的准确性，您都同意将此事实或情况告知乐鑫。

贡献者

日期	
姓名	
称谓	
地址	

乐鑫

日期	
姓名	
称谓	
地址	

8.5.7 版权标头指南

ESP-IDF 基于 the Apache License 2.0，并包含一些不同许可证下的第三方版权代码。要了解更多信息，请参考*Copyrights and Licenses*。

本页面介绍了如何在源代码中正确标注版权标头。ESP-IDF 使用 Software Package Data Exchange (SPDX) 格式，简短易读，能够方便自动化工具处理及进行版权检查。

如何检查版权标头

请确保你已安装了包含版权标头检查器的*pre-commit hooks*。若该检查器无法检测到格式正确的 SPDX 标头，则会提供标头建议。

检查器的建议不正确怎么办

再好的自动化检查工具都无法取代人工检查。因此，开发者有责任修改检查器提供的标头，使其符合源代码的法律规范和许可证要求。某些许可证可能并不兼容，请参考后文示例。

检查器由 `tools/ci/check_copyright_config.yaml` 进行配置。请检查该配置文件提供的选项，并根据实际情况进行更新，以正确匹配标头。

常见版权标头示例

最简单的情况是，你的代码不基于任何此前已获得许可的代码，例如你完全独立完成的代码。此类代码可以使用如下版权标头，并放在 ESP-IDF 许可证下:

```
/*
 * SPDX-FileCopyrightText: 2015-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: Apache-2.0
 */
```

ESP-IDF 限制性较小的部分 ESP-IDF 的某些部分特意采用了限制性较小的许可证，方便在商业闭源代码项目中重复使用。例如公有域下或免费知识共享 (CC0) 许可下的 ESP-IDF examples 。此类源文件可使用如下标头:

```
/*
 * SPDX-FileCopyrightText: 2015-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: Unlicense OR CC0-1.0
 */
```

如允许使用多个许可证，可采用关键字 `OR` 连接，如上例所示。这可以通过在 `tools/ci/check_copyright_config.yaml` 配置文件中定义多个许可证实现。但是，请谨慎使用该方法，并仅在 ESP-IDF 项目限制性较小的部分中选择性使用。

第三方许可证 受到不同许可证许可，经乐鑫修改，并包含在 ESP-IDF 中的代码不能使用 Apache License 2.0 进行许可，即便检查器可能提出此类建议。建议保留原有版权标头，并在前面添加 SPDX 标识。

例如，对于一个由"GNU General Public License v2.0 及以上" 许可证许可、John Doe 持有、且由 Espressif Systems 做出额外修改的代码文件，请按照如下示例提供标头:

```
/*
 * SPDX-FileCopyrightText: 1991 John Doe
 *
 * SPDX-License-Identifier: GPL-2.0-or-later
 *
 * SPDX-FileContributor: 2019-2023 Espressif Systems (Shanghai) CO LTD
 */
```

这些许可证能够得到识别，并且可以在官方 SPDX license list 中找到 SPDX 标识符。其他常见许可证还包括 GPL-2.0-only，BSD-3-Clause 和 BSD-2-Clause。

特殊情况下，如果一个许可证没有包含在 SPDX license list 中，可使用 LicenseRef-[idString] 自定义许可证标识符，如以下示例中的 `LicenseRef-Special-License`。另外，必须把完整的许可文本添加到 `Special-License` 文件夹下的 `LICENSES` 路径中:

```
/*
 * SPDX-FileCopyrightText: 2015-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: LicenseRef-Special-License
 */
```

如果自定义许可证直接包含在源文件中，可使用专门的自定义许可证标识符 `LicenseRef-Included` 进行说明:

```
/*
 * SPDX-FileCopyrightText: 2015-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: LicenseRef-Included
 *
 * <Full custom license text>
 */
```

`tools/ci/check_copyright_config.yaml` 中的配置为第三方许可证提供了许多有用的功能:

- 对第三方程序库的文件部分，可以定义一个不同的许可证。
- 可以永久禁用对选定文件集的检查。请谨慎使用该选项，并且仅在其他选项都不适用时，才可使用该选项。

8.5.8 ESP-IDF pytest 指南

ESP-IDF 有多种类型的测试需在 ESP 芯片上执行（即 **目标测试**）。目标测试通常作为 IDF 测试项目（即 **测试应用程序**）的一部分进行编译，在这个过程中，测试应用程序和其他标准 IDF 项目遵循同样的构建、烧写和监控流程。

通常，目标测试需要连接一台主机（如个人电脑），负责触发特定的测试用例、提供测试数据、检查测试结果。

ESP-IDF 在主机端使用 pytest 框架（以及一些 pytest 插件）来自动进行目标测试。本文档介绍 ESP-IDF 中的 pytest，并介绍以下内容:

1. ESP-IDF 中不同类型的测试应用程序。
2. 将 pytest 框架应用于 Python 测试脚本，进行自动化目标测试。
3. ESP-IDF CI (Continuous Integration) 板载测试流程。
4. 使用 pytest 在本地执行目标测试。
5. pytest 的使用技巧。

备注： ESP-IDF 默认使用以下插件：

- pytest-embedded 和默认服务 `esp,idf`
- pytest-rerunfailures

本文档介绍的所有概念和用法都基于 ESP-IDF 的默认配置，并非都适用于原生 pytest。

安装

所有依赖项都可以通过执行安装脚本的 `--enable-pytest` 进行安装：

```
$ install.sh --enable-pytest
```

安装过程常见问题

No Package 'dbus-1' Found

```
configure: error: Package requirements (dbus-1 >= 1.8) were not met:

No package 'dbus-1' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.
```

如果遇到上述错误信息，可能需要安装一些缺失的软件包。

如果使用 Ubuntu 系统，可能需要执行：

```
sudo apt-get install libdbus-glib-1-dev
```

或

```
sudo apt-get install libdbus-1-dev
```

如使用 Linux 其他发行版本，请在搜索引擎中查询上述错误信息，并查找对应发行版需安装哪些缺失的软件包。

Invalid command 'bdist_wheel'

```
error: invalid command 'bdist_wheel'
```

如果遇到上述错误信息，可能需要安装一些缺失的 Python 包，例如：

```
python -m pip install -U pip
```

或

```
python -m pip install wheel
```

备注： 执行 pip 命令前，请确保使用的环境为 IDF Python 虚拟环境。

测试应用程序

ESP-IDF 包含不同类型的测试应用程序，可用 pytest 自动完成。

组件测试 ESP-IDF 组件通常包含针对特定组件的测试应用程序，执行针对特定组件的单元测试。推荐通过组件测试应用程序来测试组件。所有测试应用程序都应位于 `${IDF_PATH}/components/<COMPONENT_NAME>/test_apps` 下，例如：

```
components/
└── my_component/
      ├── include/
      │   └── ...
      ├── test_apps/
      │   ├── test_app_1
      │   │   ├── main/
      │   │   │   └── ...
      │   │   ├── CMakeLists.txt
      │   │   └── pytest_my_component_app_1.py
      │   ├── test_app_2
      │   │   ├── ...
      │   │   └── pytest_my_component_app_2.py
      │   └── parent_folder
      │       ├── test_app_3
      │       │   ├── ...
      │       │   └── pytest_my_component_app_3.py
      │       └── ...
      ├── my_component.c
      └── CMakeLists.txt
```

例程测试 例程测试是为了向用户展示 ESP-IDF 的部分功能（要了解更多信息，请参考 Examples Readme）。

但是，要确保这些例程正确运行，可将例程看作测试应用，并用 pytest 自动执行。所有例程都应和已测试的例程，包括 Python 测试脚本一起放在 `${IDF_PATH}/examples` 路径下，例如：

```
examples/
└── parent_folder/
    └── example_1/
        ├── main/
        │   └── ...
        ├── CMakeLists.txt
        └── pytest_example_1.py
```

自定义测试 自定义测试是为了测试 ESP-IDF 的一些任意功能，这些测试不是为了向用户展示 ESP-IDF 的功能。

所有自定义测试应用都位于 `${IDF_PATH}/tools/test_apps` 路径下。要了解更多信息，请参考 Custom Test Readme 。

在 ESP-IDF 中使用 pytest

pytest 执行步骤

1. 引导阶段

创建会话缓存：

- 端口目标缓存
- 端口应用缓存

2. 数据获取阶段
   A. 获取所有前缀为 `pytest_` 的 Python 文件。
   B. 获取所有前缀为 `test_` 的测试函数。
   C. 应用 params，并复制测试函数。
   D. 利用 CLI 选项筛选测试用例。详细用法请参考筛选测试用例。

3. 运行阶段
    A. 创建 fixture。在 ESP-IDF 中，常见 fixture 的初始化顺序如下：
        a. `pexpect_proc`: pexpect 实例
        b. `app`: IdfApp 实例
            此阶段会解析测试应用程序的相关信息，如 sdkconfig、flash_files、partition_table 等。
        c. `serial`: IdfSerial 实例
            此阶段会自动检测目标芯片所连接的主机端口。考虑到主机可能连接了多个目标芯片，应用程序会解析测试目标芯片的类型。测试程序的二进制文件会自动烧写到测试目标芯片上。
        d. `dut`: IdfDut 实例
    B. 运行测试函数。
    C. 析构 fixture。析构顺序如下：
        a. `dut`
            i. 关闭 `serial` 端口。
            ii. （仅适用于使用了 Unity 测试框架 的应用程序）生成 Unity 测试用例的 JUnit 报告。
        b. `serial`
        c. `app`
        d. `pexpect_proc`：关闭文件描述符
    D.（仅适用于使用了 Unity 测试框架 的应用程序）
        如果调用了 `dut.expect_from_unity_output()`，那么检测到 Unity 测试失败时会触发 `AssertionError`。
4. 报告阶段
    A. 为测试函数生成 Junit 报告。
    B. 将 JUnit 报告中的测试用例名修改为 ESP-IDF 测试用例 ID 格式：`<target>.<config>.<test function name>`。
5. 完成阶段（仅适用于使用了 Unity 测试框架 的应用程序）
    如果生成了 Unity 测试用例的 JUnit 报告，这些报告会被合并。

入门示例 以下 Python 测试脚本示例来自 pytest_console_basic.py 。

```
@pytest.mark.esp32
@pytest.mark.esp32c3
@pytest.mark.generic
@pytest.mark.parametrize('config', [
    'history',
    'nohistory',
], indirect=True)
def test_console_advanced(config: str, dut: IdfDut) -> None:
    if config == 'history':
        dut.expect('Command history enabled')
    elif config == 'nohistory':
        dut.expect('Command history disabled')
```

下面的小节对这个简单的测试脚本进行了逐行讲解，以说明 pytest 在 ESP-IDF 测试脚本中的典型使用方法。

目标芯片 marker 使用 Pytest marker 可以指出某个特定测试用例应在哪个目标芯片（即 ESP 芯片）上运行。例如：

```
@pytest.mark.esp32      # <-- support esp32
@pytest.mark.esp32c3    # <-- support esp32c3
@pytest.mark.generic    # <-- test env "generic"
```

上例表明，某一测试用例可以在 ESP32 和 ESP32-C3 上运行。此外，目标芯片的类型应为 `generic`。要了解有关 `generic` 类型，运行 `pytest --markers` 以获取所有 marker 的详细信息。

备注： 如果测试用例可以在 ESP-IDF 官方支持的所有目标芯片上运行（调用 `idf.py --list-targets` 了解详情），则可以使用特殊 marker `supported_targets` 指定所有目标芯片。

参数化 marker 可使用 `pytest.mark.parametrize` 和 `config` 参数对包含不同 sdkconfig 文件的不同应用程序进行相同的测试。如需了解关于 `sdkconfig.ci.xxx` 文件的更多信息，请参考 readme 下的 Configuration Files 章节。

```
@pytest.mark.parametrize('config', [
    'history',      # <-- run with app built by sdkconfig.ci.history
    'nohistory',    # <-- run with app built by sdkconfig.ci.nohistory
], indirect=True)   # <-- `indirect=True` is required
```

总体而言，这一测试函数会复制为 4 个测试用例：

- `esp32.history.test_console_advanced`
- `esp32.nohistory.test_console_advanced`
- `esp32c3.history.test_console_advanced`
- `esp32c3.nohistory.test_console_advanced`

测试串行输出 为确保测试在目标芯片上顺利执行，测试脚本可使用 `dut.expect()` 函数来测试目标芯片上的串行输出：

```
def test_console_advanced(config: str, dut: IdfDut) -> None:  # The value of␣
↪argument ``config`` is assigned by the parameterization.
    if config == 'history':
        dut.expect('Command history enabled')
    elif config == 'nohistory':
        dut.expect('Command history disabled')
```

在执行 `dut.expect(...)` 时，首先会将预期字符串编译成正则表达式用于搜索串行输出结果，直到找到与该编译后的正则表达式匹配的结果或运行超时。

如果预期字符串中包含正则表达式关键字（如括号或方括号），则需格外注意。或者，也可以使用 `dut.expect_exact(...)`，它会尝试直接匹配字符串，而不将其转换为正则表达式。

如需了解关于 `expect` 函数类型的更多信息，请参考 pytest-embedded Expecting documentation。

进阶示例

用同一应用程序进行多个 DUT 测试 有时，一个测试可能涉及多个运行同一测试程序的目标芯片。在这种情况下，可以使用 `parameterize` 将多个 DUT 实例化，例如：

```
@pytest.mark.esp32s2
@pytest.mark.esp32s3
@pytest.mark.usb_host
@pytest.mark.parametrize('count', [
    2,
], indirect=True)
def test_usb_host(dut: Tuple[IdfDut, IdfDut]) -> None:
    device = dut[0]  # <-- assume the first dut is the device
    host = dut[1]    # <-- and the second dut is the host
    ...
```

将参数 `count` 设置为 2 后，所有 fixture 都会改为元组。

用不同应用程序进行多个 DUT 测试 有时（特别是协议测试），一个测试可能涉及多个运行不同测试程序的目标芯片（例如不同目标芯片作为主机和从机）。在这种情况下，可以使用 `parameterize` 将针对不同测试应用程序的多个 DUT 实例化。

以下代码示例来自 pytest_wifi_getting_started.py 。

```
@pytest.mark.esp32
@pytest.mark.multi_dut_generic
@pytest.mark.parametrize(
    'count, app_path', [
        (2,
         f'{os.path.join(os.path.dirname(__file__), "softAP")}|{os.path.join(os.
↪path.dirname(__file__), "station")}'),
    ], indirect=True
)
def test_wifi_getting_started(dut: Tuple[IdfDut, IdfDut]) -> None:
    softap = dut[0]
    station = dut[1]
    ...
```

以上示例中，第一个 DUT 用 softAP 应用程序烧录，第二个 DUT 用 station 应用程序烧录。

备注： 这里的 `app_path` 应设置为绝对路径。Python 中的 `__file__` 宏会返回测试脚本自身的绝对路径。

用不同应用程序和目标芯片进行多目标测试 以下代码示例来自 pytest_wifi_getting_started.py 。如注释所述，该代码目前尚未在 ESP-IDF CI 中运行。

```
@pytest.mark.parametrize(
    'count, app_path, target', [
        (2,
         f'{os.path.join(os.path.dirname(__file__), "softAP")}|{os.path.join(os.
↪path.dirname(__file__), "station")}',
         'esp32|esp32s2'),
        (2,
         f'{os.path.join(os.path.dirname(__file__), "softAP")}|{os.path.join(os.
↪path.dirname(__file__), "station")}',
         'esp32s2|esp32'),
    ],
    indirect=True,
)
def test_wifi_getting_started(dut: Tuple[IdfDut, IdfDut]) -> None:
    softap = dut[0]
    station = dut[1]
    ...
```

总体而言，此测试函数会被复制为 2 个测试用例：

- 在 ESP32 上烧录 softAP，在 ESP32-S2 上烧录 station
- 在 ESP32-S2 上烧录 softAP，在 ESP32 上烧录 station

支持对不同 sdkconfig 文件及目标芯片的组合测试 以下进阶代码示例来自 pytest_panic.py 。

```
CONFIGS = [
    pytest.param('coredump_flash_bin_crc', marks=[pytest.mark.esp32, pytest.mark.
↪esp32s2]),
    pytest.param('coredump_flash_elf_sha', marks=[pytest.mark.esp32]),  # sha256␣
↪only supported on esp32
    pytest.param('coredump_uart_bin_crc', marks=[pytest.mark.esp32, pytest.mark.
↪esp32s2]),
```

（下页继续）

(续上页)

```
    pytest.param('coredump_uart_elf_crc', marks=[pytest.mark.esp32, pytest.mark.
↪esp32s2]),
    pytest.param('gdbstub', marks=[pytest.mark.esp32, pytest.mark.esp32s2]),
    pytest.param('panic', marks=[pytest.mark.esp32, pytest.mark.esp32s2]),
]

@pytest.mark.parametrize('config', CONFIGS, indirect=True)
...
```

自定义类 通常，可能会在下列情况下编写自定义类：

1. 向一定数量的 DUT 添加更多可复用功能。
2. 为不同阶段添加自定义的前置和后置函数，请参考章节*pytest 执行步骤*。

以下代码示例来自 panic/conftest.py 。

```
class PanicTestDut(IdfDut):
    ...

@pytest.fixture(scope='module')
def monkeypatch_module(request: FixtureRequest) -> MonkeyPatch:
    mp = MonkeyPatch()
    request.addfinalizer(mp.undo)
    return mp


@pytest.fixture(scope='module', autouse=True)
def replace_dut_class(monkeypatch_module: MonkeyPatch) -> None:
    monkeypatch_module.setattr('pytest_embedded_idf.dut.IdfDut', PanicTestDut)
```

`monkeypatch_module` 提供了一个 基于模块 的 monkeypatch fixture。

`replace_dut_class` 是一个 基于模块 的 自动执行 fixture。该函数会用你的自定义类替换 `IdfDut` 类。

标记不稳定测试 某些测试用例基于以太网或 Wi-Fi。然而由于网络问题，测试可能会不稳定。此时，可以将某个测试用例标记为不稳定的测试用例。

以下代码示例来自 pytest_esp_eth.py 。

```
@pytest.mark.flaky(reruns=3, reruns_delay=5)
def test_esp_eth_ip101(dut: IdfDut) -> None:
    ...
```

这一 marker 表示，如果该测试函数失败，其测试用例会每隔 5 秒钟再运行一次，最多运行三次。

标记已知失败 有时，测试会因以下原因而持续失败：

- 测试的功能（或测试本身）存在错误。
- 测试环境不稳定（例如网络问题），导致失败率较高。

可使用 xfail marker 来标记此测试用例，并写出原因。

以下代码来自 pytest_panic.py 。

```
@pytest.mark.xfail('config.getvalue("target") == "esp32s2"', reason='raised␣
↪IllegalInstruction instead')
def test_cache_error(dut: PanicTestDut, config: str, test_func_name: str) -> None:
```

这一 marker 表示该测试在 ESP32-S2 上是一个已知失败。

标记夜间运行的测试用例 在缺少 runner 时，一些测试用例仅在夜间运行的管道中触发。

```
@pytest.mark.nightly_run
```

这一 marker 表示，此测试用例仅在环境变量为 NIGHTLY_RUN 或 INCLUDE_NIGHTLY_RUN 时运行。

标记在 CI 中暂时禁用的测试用例 在缺少 runner 时，可以在 CI 中禁用一些本地能够通过测试的测试用例。

```
@pytest.mark.temp_skip_ci(targets=['esp32', 'esp32s2'], reason='lack of runners')
```

这一 marker 表明，此测试用例仍可以在本地用 pytest --target esp32 执行，但不会在 CI 中执行。

运行 Unity 测试用例 对基于组件的单元测试应用程序，以下代码即可执行所有的单板测试用例，包括普通测试用例和多阶段测试用例：

```
def test_component_ut(dut: IdfDut):
    dut.run_all_single_board_cases()
```

此代码还会跳过所有 tag 为 [ignore] 的测试用例。

如需按组执行测试用例，可运行：

```
def test_component_ut(dut: IdfDut):
    dut.run_all_single_board_cases(group='psram')
```

此代码会触发模块包含 [psram] tag 的所有测试用例。

你可能还会看到一些包含以下语句的测试脚本，这些脚本已被弃用。请使用上述建议的方法。

```
def test_component_ut(dut: IdfDut):
    dut.expect_exact('Press ENTER to see the list of tests')
    dut.write('*')
    dut.expect_unity_test_output()
```

如需了解关于 ESP-IDF 单元测试的更多内容，请参考*ESP32-P4 中的单元测试*。

在 CI 中执行测试

CI 的工作流程很简单，即编译任务 -> 板载测试任务。

编译任务

编译任务命名

- 基于组件的单元测试：build_pytest_components_<target>
- 例程测试：build_pytest_examples_<target>
- 自定义测试：build_pytest_test_apps_<target>

编译任务命令 CI 用于创建所有相关测试的命令为：python $IDF_PATH/tools/ci/ci_build_apps.py <parent_dir> --target <target> -vv --pytest-apps

所有支持指定目标芯片的应用程序都使用 build_<target>_<config> 下支持的 sdkconfig 文件创建。

例如，如果运行 python $IDF_PATH/tools/ci/ci_build_apps.py $IDF_PATH/examples/system/console/basic --target esp32 --pytest-apps 指令，文件夹结构将如下所示：

```
basic
├── build_esp32_history/
│   └── ...
├── build_esp32_nohistory/
│   └── ...
├── main/
├── CMakeLists.txt
├── pytest_console_basic.py
└── ...
```

所有编译文件的文件夹都会上传到同一路径。

板载测试任务

板载测试任务命名

- 基于部件的单元测试：`component_ut_pytest_<target>_<test_env>`
- 例程测试：`example_test_pytest_<target>_<test_env>`
- 自定义测试：`test_app_test_pytest_<target>_<test_env>`

板载测试任务命令 CI 用于执行所有相关测试的命令为：`pytest <parent_dir> --target <target> -m <test_env_marker>`

这一命令将执行父文件夹下所有具有指定目标芯片 marker 和测试环境 marker 的测试用例。

板载测试任务中的二进制文件是从编译任务中下载的，相应文件会放在同一路径下。

本地测试

首先，你需为 ESP-IDF 安装 Python 依赖：

```
$ cd $IDF_PATH
$ bash install.sh --enable-pytest
$ . ./export.sh
```

默认情况下，pytest 脚本会按照以下顺序查找编译目录：

- `build_<target>_<sdkconfig>`
- `build_<target>`
- `build_<sdkconfig>`
- `build`

因此，运行 pytest 最简单的方式是调用 `idf.py build`。

例如，如果你要执行 `$IDF_PATH/examples/get-started/hello_world` 文件夹下的所有 ESP32 测试，你可执行：

```
$ cd examples/get-started/hello_world
$ idf.py build
$ pytest --target esp32
```

如果你的测试应用程序中有多个 sdkconfig 文件，例如那些 `sdkconfig.ci.*` 文件，仅使用 `idf.py build` 命令并不能调用这些额外的 sdkconfig 文件。下文以 `$IDF_PATH/examples/system/console/basic` 为例进行说明。

如果要用 `history` 配置测试此应用程序，并用 `idf.py build` 进行编译，你需运行：

```
$ cd examples/system/console/basic
$ idf.py -DSDKCONFIG_DEFAULTS="sdkconfig.defaults;sdkconfig.ci.history" build
$ pytest --target esp32 --sdkconfig history
```

如果你想同时编译测试所有 sdkconfig 文件，则需运行我们的 CI 脚本 (ci_build_apps.py) 作为辅助脚本：

```
$ cd examples/system/console/basic
$ python $IDF_PATH/tools/ci/ci_build_apps.py . --target esp32 -vv --pytest-apps
$ pytest --target esp32
```

包含 `sdkconfig.ci.history` 配置的应用程序会编译到 `build_esp32_history` 中，而包含 `sdkconfig.ci.nohistory` 配置的应用程序会编译到 `build_esp32_nohistory` 中。`pytest --target esp32` 命令会在这两个应用程序上运行测试。

使用技巧

筛选测试用例

- 根据目标芯片筛选：`pytest --target <target>`
  pytest 会执行所有支持指定目标芯片的测试用例。
- 根据 sdkconfig 文件筛选：`pytest --sdkconfig <sdkconfig>`
  如果 `<sdkconfig>` 为 `default`，pytest 会执行所有 sdkconfig 文件包含 `sdkconfig.defaults` 的测试用例。
  如果是其他情况，pytest 会执行所有 sdkconfig 文件包含 `sdkconfig.ci.<sdkconfig>` 的测试用例。
- 使用 `pytest -k <test-case name>` 按测试用例名称筛选，可以运行单个测试用例，例如 `pytest -k test_int_wdt_cache_disabled`。

添加新 marker 我们目前使用两种自定义 marker。target marker 是指测试用例支持此目标芯片，env marker 是指测试用例应分配到 CI 中具有相应 tag 的 runner 上。

你可以在 `${IDF_PATH}/conftest.py` 文件后添加一行新的 marker。如果该 marker 是 target marker，应将其添加到 `TARGET_MARKERS` 中。如果该 marker 指定了一类测试环境，应将其添加到 `ENV_MARKERS` 中。自定义 marker 格式：`<marker_name>: <marker_description>`。

生成 JUnit 报告 你可调用 pytest 执行 `--junitxml <filepath>` 生成 JUnit 报告。在 ESP-IDF 中，测试用例命名会统一为 `<target>.<config>.<function_name>`。

跳过自动烧录二进制文件 调试测试脚本时最好跳过自动烧录二进制文件。

调用 pytest 执行 `--skip-autoflash y` 即可实现。

记录数据 在执行测试时，你有时需要记录一些数据，例如性能测试数据。

在测试脚本中使用 record_xml_attribute fixture，数据就会记录在 JUnit 报告的属性中。

日志系统 在执行测试用例时，你有时可能需要添加一些额外的日志行。

这可通过使用 Python 日志模块 实现。

其他日志函数（作为 fixture）

log_performance

```
def test_hello_world(
    dut: IdfDut,
    log_performance: Callable[[str, object], None],
) -> None:
    log_performance('test', 1)
```

以上示例可实现用预定义格式 `[performance][test]: 1` 记录性能数据，并在指定 `--junitxml <filepath>` 的情况下将其记录在 JUnit 报告的 `properties` tag 下。相应的 JUnit 测试用例节点如下所示：

```
<testcase classname="examples.get-started.hello_world.pytest_hello_world" file=
↪"examples/get-started/hello_world/pytest_hello_world.py" line="13" name="esp32.
↪default.test_hello_world" time="8.389">
    <properties>
        <property name="test" value="1"/>
    </properties>
</testcase>
```

check_performance 我们提供了 `TEST_PERFORMANCE_LESS_THAN` 和 `TEST_PERFORMANCE_GREATER_THAN` 宏来记录性能项，并检测性能项的数值是否在有效范围内。有时 C 宏无法检测一些性能项的值，为此，我们提供了 Python 函数实现相同的目的。注意，由于该 Python 函数不能很好地识别不同的 ifdef 块下同一性能项的阈值，请尽量使用 C 宏。

```
def test_hello_world(
    dut: IdfDut,
    check_performance: Callable[[str, float, str], None],
) -> None:
    check_performance('RSA_2048KEY_PUBLIC_OP', 123, 'esp32')
    check_performance('RSA_2048KEY_PUBLIC_OP', 19001, 'esp32')
```

以上示例会首先从 components/idf_test/include/idf_performance.h 和指定目标芯片的 components/idf_test/include/esp32/idf_performance_target.h 头文件中获取性能项 `RSA_2048KEY_PUBLIC_OP` 的阈值，然后检查该值是否达到了最小值或超过了最大值。

例如，假设 `IDF_PERFORMANCE_MAX_RSA_2048KEY_PUBLIC_OP` 的值为 19000，则上例中第一行 `check_performance` 会通过测试，第二行会失败并警告：`[Performance] RSA_2048KEY_PUBLIC_OP value is 19001, doesn\'t meet pass standard 19000.0`。

扩展阅读

- pytest：https://docs.pytest.org/en/latest/contents.html
- pytest-embedded：https://docs.espressif.com/projects/pytest-embedded/en/latest/

Chapter 9

ESP-IDF 版本简介

ESP-IDF 的 GitHub 仓库时常更新，特别是用于开发新特性的 master 分支。

如有量产需求，请使用稳定版本。

9.1 发布版本

通过以下链接，可以访问各个版本的配套文档：

- 最新稳定版 ESP-IDF：https://docs.espressif.com/projects/esp-idf/zh_CN/stable/
- 最新版 ESP-IDF（即 master 分支）：https://docs.espressif.com/projects/esp-idf/zh_CN/latest/

ESP-IDF 在 GitHub 平台上的完整发布历史请见 发布说明页面。该页面支持查看各个版本的发布说明、配套文档及相应获取方式。

9.2 我该选择哪个版本？

- 如有量产需求，请使用 最新稳定版本 。稳定版本已通过人工测试，后续更新仅修复 bug，主要特性不受影响（更多详情，请见版本管理 ）。请访问 发布说明页面 界面查看每一个稳定发布版本。另外，为确保使用的 ESP-IDF 版本与需生产的芯片版本兼容，可参考 ESP-IDF 版本与乐鑫芯片版本兼容性。
- 如需尝试/测试 ESP-IDF 的最新特性，请使用 最新版本（在 master 分支上）。最新版本包含 ESP-IDF 的所有最新特性，已通过自动化测试，但尚未全部完成人工测试（因此存在一定风险）。
- 如需使用稳定版本中没有的新特性，但同时又不希望受到 master 分支更新的影响，可以按照需求选择一个最适合的稳定版本，将其更新至一个预发布版本 或更新至一个发布分支。
- 如需使用其他基于 ESP-IDF 的项目，请先查看该项目的文档，以确定其兼容的 ESP-IDF 版本。

有关如何更新 ESP-IDF 本地副本的内容，请参考更新 *ESP-IDF* 章节。

9.3 版本管理

ESP-IDF 采用了 语义版本管理方法，你可以从字面含义理解每个版本的差异。其中

- 主要版本（例 `v3.0`）代表有重大更新，包括增加新特性、改变现有特性及移除已弃用的特性。升级至一个新的主要版本（例 `v2.1` 升级至 `v3.0`）意味着你可能需要更新工程代码，并重新测试工程，具体可参考 发布说明页面 的重大变更 (Breaking Change) 部分。
- 次要版本（例 `v3.1`）代表有新增特性和 bug 修复，但现有特性不受影响，公开 API 的使用也不受影响。

升级至一个新的次要版本（例 `v3.0` 升级至 `v3.1`）意味着你可能不需要更新工程代码，但需重新测试工程，特别是 发布说明页面 中专门提到的部分。

- Bugfix 版本（例 `v3.0.1`）仅修复 bug，并不增加任何新特性。
  升级至一个新的 Bugfix 版本（例 `v3.0` 升级至 `v3.0.1`）意味着你不需要更新工程代码，仅需测试与本次发布修复 bug（列表见 发布说明页面）直接相关的特性。

9.4 支持期限

ESP-IDF 的每个主要版本和次要版本都有相应的支持期限。支持期限满后，版本停止更新维护，将不再提供支持。

支持期限政策 对此有具体描述，并介绍了每个版本的支持期限是如何界定的。

发布说明页面 界面上的每一个发布版本都提供了该版本的支持期限信息。

一般而言：

- 如果你刚开始一个新项目，建议使用最新稳定版本。
- 如果你有 GitHub 账号，请点击 发布说明页面 界面右上角的"Watch" 按键，并选中"Releases only" 选项。GitHub 将会在新版本发布的时候发送通知。当你所使用的版本有 Bugfix 版本发布时，请做好升级至该 Bugfix 版本的规划。
- 如可能，请定期（如每年一次）将项目的 IDF 版本升级至一个新的主要版本或次要版本。对于次要版本更新，更新过程应该比较简单，但对于主要版本更新，可能需要细致查看发布说明并做对应的更新规划。
- 请确保你所使用的版本停止更新维护前，已做好升级至新版本的规划。

ESP-IDF 的每个主要版本和次要版本（V4.1、V4.2 等）的支持期限为 30 个月，从最初的稳定版发布日算起。

在支持期限内意味着 ESP-IDF 团队将继续在 GitHub 的发布分支上进行 bug 修复、安全修复等，并根据需要定期发布新的 Bugfix 版本。

支持期限分为“服务期”和“维护期”：

周期	时长	是否推荐新工程使用
服务期	12 个月	是
维护期	18 个月	否

在服务期内，Bugfix 版本的发布更为频繁。某些情况下，在服务期内会增加新特性，这些特性主要是为了满足新产品特定监管要求或标准，并且回归风险非常低。

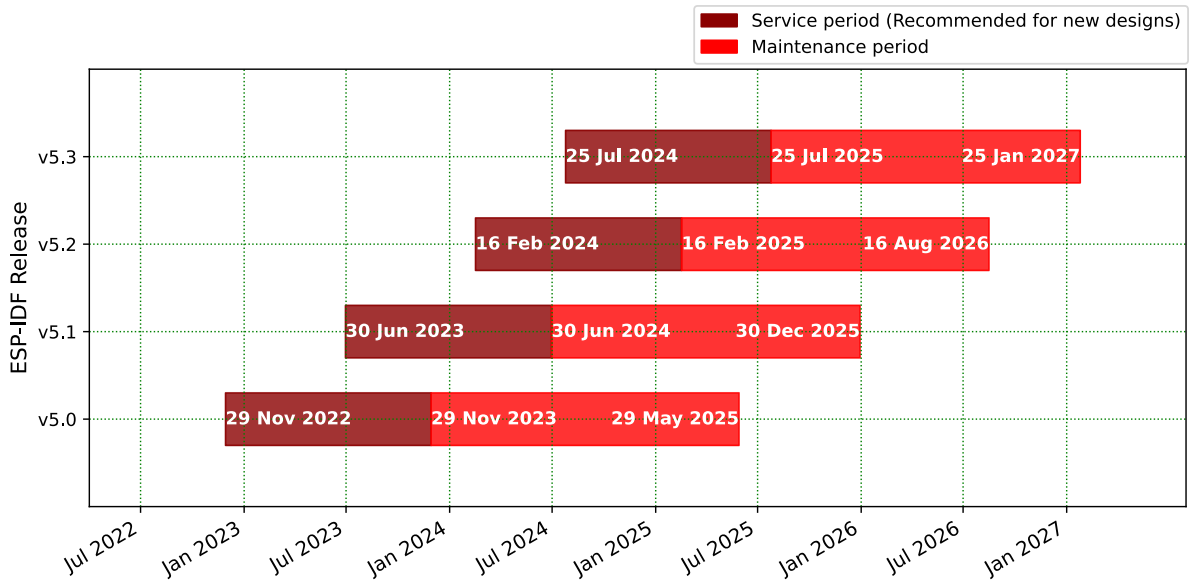
在维护期内，该版本仍受支持，但只会对严重性较高的问题或安全问题进行 bug 修复。

当开始一个新项目时，建议使用在服务期内的版本。

鼓励用户在所用版本支持期限结束之前，将所有的工程升级到最新的 ESP-IDF 版本。在版本支持期限满后，我们将不再继续进行 bug 修复。

支持期限不包括预发布版本（betas、预览版、*-rc* 和 *-dev* 版等），有时会将某个特性在发布版中标记为“预览版”，这意味着该特性也不在支持期限内。

关于 不同版本的 ESP-IDF （主要版本、次要版本、Bugfix 版本等）的更多信息，请参考 ESP-IDF 编程指南。



9.5 查看当前版本

查看 ESP-IDF 本地副本的版本，请使用 idf.py 命令:

```
idf.py --version
```

此外，由于 ESP-IDF 的版本也已编译至固件中，因此你也可以使用宏 `IDF_VER` 查看 ESP-IDF 的版本（以字符串的格式）。ESP-IDF 默认引导程序会在设备启动时打印 ESP-IDF 的版本。请注意，在 GitHub 仓库中的代码更新时，代码中的版本信息仅会在源代码重新编译或在清除编译时才会更新，因此打印出来的版本可能并不是最新的。

如果编写的代码需要支持多个 ESP-IDF 版本，可以在编译时使用*compile-time macros* 检查版本。

几个 ESP-IDF 版本的例子:

版本字符串	含义
`v3.2-dev-306-gbeb3611ca`	master 分支上的预发布版本。 - `v3.2-dev`：为 v3.2 进行的开发。 - `306`：v3.2 开发启动后的 commit 数量。 - `beb3611ca`：commit 标识符。
`v3.0.2`	稳定版本，标签为 `v3.0.2`。
`v3.1-beta1-75-g346d6b0ea`	v3.1 的 beta 测试版本（可参考更新至一个发布分支）。 - `v3.1-beta1` - 预发布标签。 - `75`：添加预发布 beta 标签后的 commit 数量。 - `346d6b0ea`：commit 标识符。
`v3.0.1-dirty`	稳定版本，标签为 `v3.0.1`。 - `dirty` 代表 ESP-IDF 的本地副本有修改。

9.6 Git 工作流

乐鑫 ESP-IDF 团队的 (Git) 开发工作流程如下：

- 新的改动总是在 master 分支（最新版本）上进行。master 分支上的 ESP-IDF 版本总带有 `-dev` 标签，表示“正在开发中”，例 `v3.1-dev`。
- 这些改动将首先在乐鑫的内部 Git 仓库进行代码审阅与测试，而后在自动化测试完成后推至 GitHub。
- 新版本一旦完成特性开发（在 `master` 分支上进行）并达到进入 beta 测试的标准，则将该版本切换至一个新分支（例 `release/v3.1`）。此外，该分支还打上预发布标签（例 `v3.1-beta1`）。你可以在 GitHub 平台上查看 ESP-IDF 的完整 分支列表 和 标签列表。Beta 预发布版本可能仍存在大量“已知问题”(Known Issue)。
- 随着对 beta 版本的不断测试，bug 修复将同时增加至该发布分支和 `master` 分支。而且，`master` 分支可能也已经开始为下个版本开发新特性了。
- 当测试快结束时，该发布分支上将增加一个 rc 标签，代表候选发布 (Release Candidate)，例 `v3.1-rc1`。此时，该分支仍属于预发布版本。
- 如果一直未发现或报告重大 bug，则该预发布版本将最终增加“主要版本”（例 `v4.0`）或“次要版本”标记（例 `v3.1`），成为正式发布版本，并体现在 发布说明页面 。
- 后续，发布版本中发现的 bug 都将在该发布分支上进行修复。
- 发布分支上会定期进行 bug 修复，人工测试完成后，该分支将增加一个 Bugfix 版本标签（例 `v3.1.1`），并体现在 发布说明页面 。

9.7 更新 ESP-IDF

请根据实际情况，对 ESP-IDF 进行更新。

- 如有量产用途，建议参考*更新至一个稳定发布版本*。
- 如需测试/研发/尝试最新特性，建议参考*更新至 master 分支* 。
- 两者折衷建议参考*更新至一个发布分支*。

备注： 在参考本指南时，请首先获得 ESP-IDF 的本地副本，具体步骤请参考入门指南 中的介绍。

9.7.1 更新至一个稳定发布版本

对于量产用户，推荐更新至一个新的 ESP-IDF 发布版本，请参考以下步骤：

- 请定期查看 发布说明页面 ，了解最新发布情况。
- 如有新发布的 Bugfix 版本（例 `v3.0.1` 或 `v3.0.2`）时，请将新的 Bugfix 版本更新至你的 ESP-IDF 目录。
- 在 Linux 或 macOS 系统中，请运行如下命令将分支更新至 vX.Y.Z：

```
cd $IDF_PATH
git fetch
git checkout vX.Y.Z
git submodule update --init --recursive
```

- 在 Windows 系统中，需要将 `cd $IDF_PATH` 替换为 `cd %IDF_PATH%`。
- 在主要版本或次要版本新发布时，请查看发布说明中的具体描述，并决定是否升级版本。具体命令与上方描述一致。

备注： 如果你在安装 ESP-IDF 时使用的是 zip 文件包，而非通过 Git 命令，则将无法使用 Git 命令进行版本升级，此属正常情况。这种情况下，请重新下载最新 zip 文件包，并替换掉之前 `IDF_PATH` 下的全部内容。

9.7.2 更新至一个预发布版本

你也可以将你的本地副本切换（命令 `git checkout`）至一个预发布版本或 rc 版本，具体方法请参考更新至一个稳定发布版本 中的描述。

预发布版本通常不体现在 发布说明页面。更多详情，请查看完整 标签列表。使用预发布版本的注意事项，请参考更新至一个发布分支 中的描述。

9.7.3 更新至 master 分支

备注： ESP-IDF 中 master 分支上的代码会时时更新，因此使用 master 分支相当在“流血的边缘试探”，存在一定风险。

如需使用 ESP-IDF 的 master 分支，请参考以下步骤：

- 在 Linux 或 macOS 系统中，使用如下命令在本地切换至 master 分支：

```
cd $IDF_PATH
git checkout master
git pull
git submodule update --init --recursive
```

- 在 Windows 系统中，需要将 `cd $IDF_PATH` 替换为 `cd %IDF_PATH%`。
- 此外，你还应在后续工作中不时使用 `git pull` 命令，将远端 master 上的更新同步到本地。注意，在更新 master 分支后，你可能需要更改工程代码，也可能遇到新的 bug。
- 如需从 master 分支切换至一个发布分支或稳定版本，请使用 `git checkout` 命令。

重要： 强烈建议定期使用 `git pull` 和 `git submodule update --init --recursive` 命令，确保本地副本得到及时更新。旧的 master 分支相当于一个“快照”，可能存在未记录的问题，且无法获得支持。对于半稳定版本，请参考更新至一个发布分支。

9.7.4 更新至一个发布分支

从稳定性来说，使用“发布分支”相当于在使用 master 分支和稳定版本之间进行折衷，包含一些 master 分支上的新特性，但也同时保证可通过 beta 测试且基本完成了 bug 修复。

更多详情，请前往 GitHub 查看完整 标签列表。

例如，在 Linux 或 macOS 系统中，可以运行以下命令更新至 ESP-IDF v3.1，随时关注该分支上的 Bugfix 版本发布（如 `v3.1.1` 等）：

```
cd $IDF_PATH
git fetch
git checkout release/v3.1
git pull
git submodule update --init --recursive
```

在 Windows 系统中，需要将 `cd $IDF_PATH` 替换为 `cd %IDF_PATH%`。

每次在该分支上使用 `git pull` 时，都相当于把最新的 Bugfix 版本发布更新至你的本地副本中。

备注： 发布分支并不会有专门的配套文档，建议使用与本分支最接近版本的文档。

Chapter 10

资源

10.1 PlatformIO

- 什么是 *PlatformIO*？
- 安装
- 配置
- 教程
- 项目示例
- 更多内容

10.1.1 什么是 PlatformIO？

PlatformIO 是一个跨平台的嵌入式开发环境，能直接支持 ESP-IDF。

由于 PlatformIO 对 ESP-IDF 的支持并非由 Espressif 团队维护，如果遇到 PlatformIO 相关的问题，请通过官方 PlatformIO 库 联系其开发人员。

要了解关于 PlatformIO 生态系统及其理念的详细概述，请参考 官方 PlatformIO 文档。

10.1.2 安装

- PlatformIO IDE 是一个可在 Windows，macOS 和 Linux 平台上使用的嵌入式 C/C++ 开发工具集。
- PlatformIO Core (CLI) 是一个命令行工具，包含多平台编译系统，平台和库管理器以及其他集成组件。它可以与各种代码开发环境一起使用，还可以集成云平台和网络服务。

10.1.3 配置

请参阅 PlatformIO 官方的 ESP-IDF 配置指南。

10.1.4 教程

- ESP-IDF and ESP32-DevKitC: debugging, unit testing, project analysis

10.1.5 项目示例

请参阅 PlatformIO 官方文档 中 ESP-IDF 相关内容。

10.1.6 更多内容

访问以下链接探索 PlatformIO 生态系统：

- 了解更多 与其他 IDE 或文本编辑器集成 的信息。
- 从 PlatformIO 社区 获取帮助。

10.2 CLion

10.2.1 CLion 是什么？

CLion 是用于 C 和 C++ 编程的跨平台集成开发环境 (IDE)。CLion 还提供了专门支持 ESP-IDF 的功能，使开发人员能够无缝使用 ESP-IDF 框架。

10.2.2 安装

安装 CLion，请参考 CLion 安装指南，可根据操作系统（Windows, macOS, or Linux）选择适用的版本。

10.2.3 配置

要在 CLion 中配置一个 ESP-IDF 项目，请参考 在 CLion 中配置 ESP CMake 项目。本指南将带您逐步完成配置项目的必要步骤。

10.2.4 资源

有关 CLion 和 ESP-IDF 集成的更多信息，请参考以下资源：

- CLion 文档：CLion 官方文档提供了有关 IDE 各方面内容的详细信息，包括 ESP-IDF 集成。

10.3 VisualGDB

10.3.1 VisualGDB 是什么?

VisualGDB 是 Microsoft Visual Studio 的一个强大扩展，为嵌入式系统提供包括支持 ESP-IDF 框架在内的一系列先进的开发工具和功能。通过 VisualGDB，能够在 ESP-IDF 项目中利用熟悉且功能丰富的 Visual Studio 环境，从而实现高效的编码、调试与部署。

10.3.2 安装

请参照 下载和安装 VisualGDB，下载和安装 VisualGDB。

10.3.3 配置

请参照 使用 ESP-IDF 在 VisualGDB 中创建高级 ESP32 项目，在 VisualGDB 中配置 ESP-IDF 项目。

也可以参考 高级 ESP-IDF 项目结构，了解更多关于使用 VisualGDB 开发 ESP-IDF 项目的信息。

10.3.4 资源

有关 VisualGDB 和 ESP-IDF 集成的更多信息，请参阅以下资源：

- VisualGDB 文档: VisualGDB 的官方文档提供了关于使用 VisualGDB 与 ESP-IDF 的全面指南和教程。

有关这些第三方工具的问题，请寻求该工具的支持渠道或用户社区的帮助。

10.4 有用的链接

- 请在 ESP32 论坛 中提出问题，访问社区资源。
- 请通过 GitHub 的 Issues 版块提交 bug 或功能请求。在提交新 Issue 之前，请先查看现有的 Issues。
- 请在 ESP IoT Solution 库中找到基于 ESP-IDF 的 解决方案、应用实例、组件和驱动 等内容。多数文档均提供中英文版本。
- 通过 Arduino 平台开发应用，请参考 ESP32、ESP32-S2 和 ESP32-C3 芯片的 Arduino 内核。
- 关于 ESP32 的书籍列表，请查看 乐鑫 网站。
- 如果你有兴趣参与到 ESP-IDF 的开发, 请查阅贡献指南。
- 关于 ESP32-P4 的其它信息，请查看官网 文档 版块。
- 关于本文档的 PDF 和 HTML 格式下载（最新版本和早期版本），请点击 下载。

Chapter 11

Copyrights and Licenses

11.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2023 Espressif Systems. This source code is licensed under the Apache License 2.0 as described in the file LICENSE.

Additional third party copyrighted code is included under the following licenses.

Where source code headers specify Copyright & License information, this information takes precedence over the summaries made here.

Some examples use external components which are not Apache licensed, please check the copyright description in each example source code.

11.1.1 Firmware Components

These third party libraries can be included into the application (firmware) produced by ESP-IDF.

- Newlib is licensed under the BSD License and is Copyright of various parties, as described in COPYING.NEWLIB .
- Xtensa header files are Copyright (C) 2013 Tensilica Inc and are licensed under the MIT License as reproduced in the individual header files.
- Original parts of FreeRTOS (components/freertos) are Copyright (C) 2017 Amazon.com, Inc. or its affiliates are licensed under the MIT License, as described in license.txt .
- Original parts of LWIP (components/lwip) are Copyright (C) 2001, 2002 Swedish Institute of Computer Science and are licensed under the BSD License as described in COPYING file .
- wpa_supplicant Copyright (c) 2003-2022 Jouni Malinen <j@w1.fi> and contributors and licensed under the BSD license.
- Fast PBKDF2 Copyright (c) 2015 Joseph Birr-Pixton and licensed under CC0 Public Domain Dedication license.
- FreeBSD net80211 Copyright (c) 2004-2008 Sam Leffler, Errno Consulting and licensed under the BSD license.
- argtable3 argument parsing library Copyright (C) 1998-2001,2003-2011,2013 Stewart Heitmann and licensed under 3-clause BSD license. argtable3 also includes the following software components. For details, please see argtable3 LICENSE file .
  - C Hash Table library, Copyright (c) 2002, Christopher Clark and licensed under 3-clause BSD license.
  - The Better String library, Copyright (c) 2014, Paul Hsieh and licensed under 3-clause BSD license.
  - TCL library, Copyright the Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, ActiveState Corporation and other parties, and licensed under TCL/TK License.
- linenoise line editing library Copyright (c) 2010-2014 Salvatore Sanfilippo, Copyright (c) 2010-2013 Pieter Noordhuis, licensed under 2-clause BSD license.
- FatFS library, Copyright (C) 2017 ChaN, is licensed under a BSD-style license .

- cJSON library, Copyright (c) 2009-2017 Dave Gamble and cJSON contributors, is licensed under MIT license as described in LICENSE file .
- micro-ecc library, Copyright (c) 2014 Kenneth MacKay, is licensed under 2-clause BSD license.
- Mbed TLS library, Copyright (C) 2006-2018 ARM Limited, is licensed under Apache License 2.0 as described in LICENSE file .
- SPIFFS library, Copyright (c) 2013-2017 Peter Andersson, is licensed under MIT license as described in LICENSE file .
- SD/MMC driver is derived from OpenBSD SD/MMC driver, Copyright (c) 2006 Uwe Stuehler, and is licensed under BSD license.
- ESP-MQTT MQTT Package (contiki-mqtt) - Copyright (c) 2014, Stephen Robinson, MQTT-ESP - Tuan PM <tuanpm at live dot com> is licensed under Apache License 2.0 as described in LICENSE file .
- BLE Mesh is adapted from Zephyr Project, Copyright (c) 2017-2018 Intel Corporation and licensed under Apache License 2.0.
- mynewt-nimble Apache Mynewt NimBLE, Copyright 2015-2018, The Apache Software Foundation, is licensed under Apache License 2.0 as described in LICENSE file .
- TLSF allocator Two Level Segregated Fit memory allocator, Copyright (c) 2006-2016, Matthew Conte, and licensed under the BSD 3-clause license.
- openthread, Copyright (c) The OpenThread Authors, is licensed under BSD License as described in LICENSE file .
- UBSAN runtime —Copyright (c) 2016, Linaro Limited and Jiří Zárevúcky, licensed under the BSD 2-clause license.
- HTTP Parser Based on src/http/ngx_http_parse.c from NGINX copyright Igor Sysoev. Additional changes are licensed under the same terms as NGINX and Joyent, Inc. and other Node contributors. For details please check LICENSE file .
- SEGGER SystemView target-side library, Copyright (c) 1995-2021 SEGGER Microcontroller GmbH, is licensed under BSD 1-clause license.
- protobuf-c Protocol Buffers implementation in C, Copyright (c) 2008-2022, Dave Benson and the protobuf-c authors. For details please check LICENSE file .
- CMock Mock/stub generator for C, Copyright (c) 2007-14 Mike Karlesky, Mark VanderVoord, Greg Williams, is licensed under MIT license as described in LICENSE file .
- Unity Simple Unit Testing library, Copyright (c) 2007-23 Mike Karlesky, Mark VanderVoord, Greg Williams, is licensed under MIT license as described in LICENSE file .

11.1.2 Documentation

- HTML version of the ESP-IDF Programming Guide uses the Sphinx theme sphinx_idf_theme, which is Copyright (c) 2013-2020 Dave Snider, Read the Docs, Inc. & contributors, and Espressif Systems (Shanghai) CO., LTD. It is based on sphinx_rtd_theme. Both are licensed under MIT license.

11.2 ROM Source Code Copyrights

Espressif SoCs mask ROM hardware includes binaries compiled from portions of the following third party software:

- Newlib , licensed under the BSD License and is Copyright of various parties, as described in COPYING.NEWLIB .
- Xtensa libhal, Copyright (c) Tensilica Inc and licensed under the MIT license (see below).
- TinyBasic Plus, Copyright Mike Field & Scott Lawrence and licensed under the MIT license (see below).
- miniz, by Rich Geldreich - placed into the public domain.
- TJpgDec Copyright (C) 2011, ChaN, all right reserved. See below for license.
- **Parts of Zephyr RTOS USB stack:**
  - DesignWare USB device driver Copyright (c) 2016 Intel Corporation and licensed under Apache 2.0 license.
  - Generic USB device driver Copyright (c) 2006 Bertrik Sikken (bertrik@sikken.nl), 2016 Intel Corporation and licensed under BSD 3-clause license.
  - USB descriptors functionality Copyright (c) 2017 PHYTEC Messtechnik GmbH, 2017-2018 Intel Corporation and licensed under Apache 2.0 license.

- USB DFU class driver Copyright (c) 2015-2016 Intel Corporation, 2017 PHYTEC Messtechnik GmbH and licensed under BSD 3-clause license.
- USB CDC ACM class driver Copyright (c) 2015-2016 Intel Corporation and licensed under Apache 2.0 license.

11.3 Xtensa libhal MIT License

Copyright (c) 2003, 2006, 2010 Tensilica Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ”Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ”AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

11.4 TinyBasic Plus MIT License

Copyright (c) 2012-2013

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ”Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ”AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

11.5 TJpgDec License

TJpgDec - Tiny JPEG Decompressor R0.01 (C) ChaN, 2011 The TJpgDec is a generic JPEG decompressor module for tiny embedded systems. This is a free software that opened for education, research and commercial developments under license policy of following terms.

Copyright (C) 2011, ChaN, all right reserved.

- The TJpgDec module is a free software and there is NO WARRANTY.
- No restriction on use. You can use, modify and redistribute it for personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY.
- Redistributions of source code must retain the above copyright notice.

Chapter 12

关于本指南

本指南为 ESP32-P4 官方应用开发框架 ESP-IDF 的配套文档。

ESP32-P4 是一款高性能 MCU，支持超大片上内存，具有强大的图像和语音处理能力。该款 MCU 包含一个高性能 (HP) 系统和一个低功耗 (LP) 系统。HP 系统由 RISC-V 双核处理器驱动，主频高达 400 MHz，并包含丰富的外设；LP 系统由 RISC-V 单核处理器驱动，主频高达 40 MHz，其外设针对低功耗应用进行了优化。

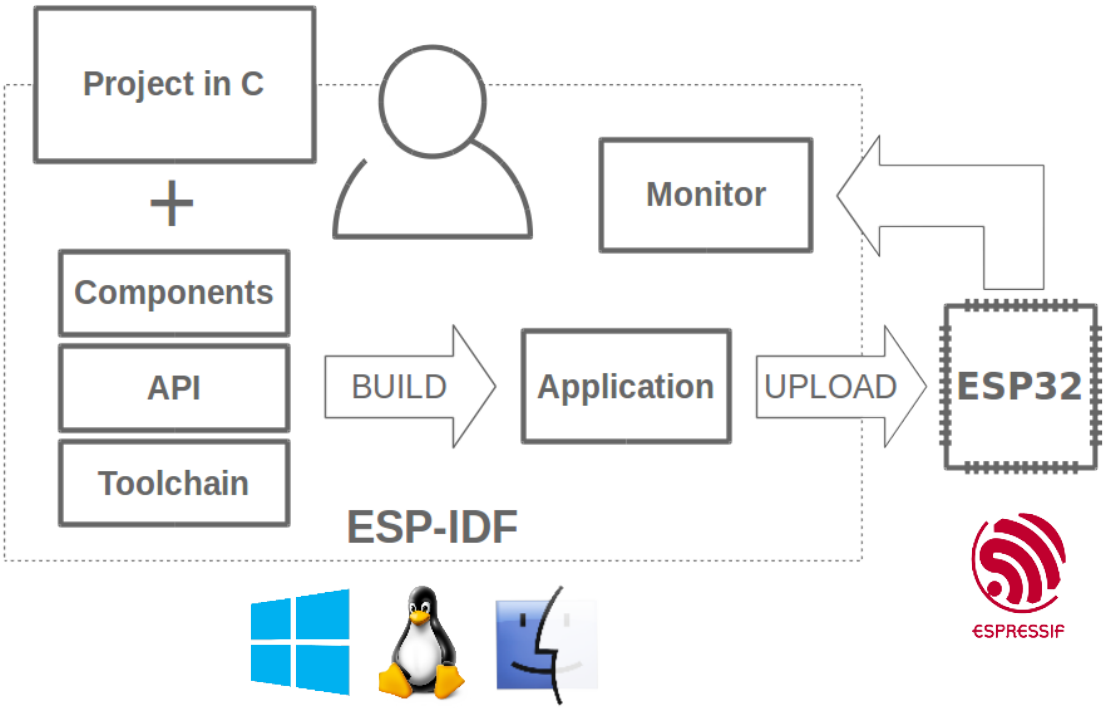


图 1: 乐鑫物联网综合开发框架

ESP-IDF 即乐鑫物联网开发框架，可为在 Windows、Linux 和 macOS 系统平台上开发 ESP32-P4 应用程序提供工具链、API、组件和工作流程的支持。

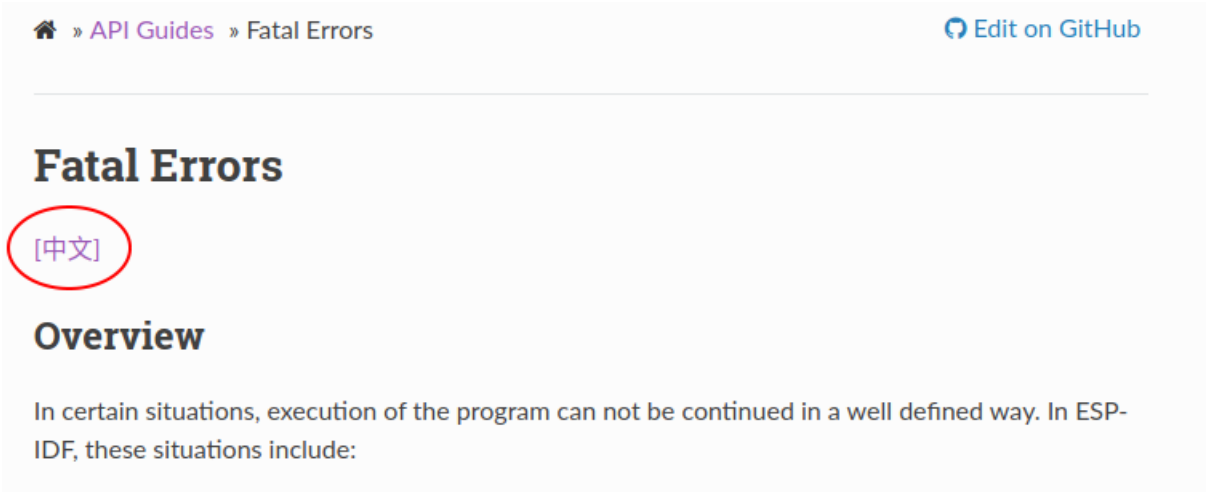
Chapter 13

切换语言

《ESP-IDF 编程指南》部分文档现在有两种语言的版本。如有出入，请以英文版本为准。

- 英文
- 中文

如下图所示，如果该文档两种语言版本均具备，可以通过点击文档上方的语言链接，轻松进行语言切换。



索引

符号

_ESP_LOG_EARLY_ENABLED (*C macro*), 1634
_ip_addr (*C++ struct*), 249
_ip_addr::ip4 (*C++ member*), 249
_ip_addr::ip6 (*C++ member*), 249
_ip_addr::type (*C++ member*), 249
_ip_addr::u_addr (*C++ member*), 249
[anonymous] (*C++ enum*), 690, 1696
[anonymous]::ESP_ERR_FLASH_NO_RESPONSE (*C++ enumerator*), 690
[anonymous]::ESP_ERR_FLASH_SIZE_NOT_MATCH (*C++ enumerator*), 690
[anonymous]::ESP_ERR_SLEEP_REJECT (*C++ enumerator*), 1696
[anonymous]::ESP_ERR_SLEEP_TOO_SHORT_SLEEP_DURATION (*C++ enumerator*), 1696

A

adc_atten_t (*C++ enum*), 260
adc_atten_t::ADC_ATTEN_DB_0 (*C++ enumerator*), 260
adc_atten_t::ADC_ATTEN_DB_11 (*C++ enumerator*), 261
adc_atten_t::ADC_ATTEN_DB_12 (*C++ enumerator*), 260
adc_atten_t::ADC_ATTEN_DB_2_5 (*C++ enumerator*), 260
adc_atten_t::ADC_ATTEN_DB_6 (*C++ enumerator*), 260
adc_bitwidth_t (*C++ enum*), 261
adc_bitwidth_t::ADC_BITWIDTH_10 (*C++ enumerator*), 261
adc_bitwidth_t::ADC_BITWIDTH_11 (*C++ enumerator*), 261
adc_bitwidth_t::ADC_BITWIDTH_12 (*C++ enumerator*), 261
adc_bitwidth_t::ADC_BITWIDTH_13 (*C++ enumerator*), 261
adc_bitwidth_t::ADC_BITWIDTH_9 (*C++ enumerator*), 261
adc_bitwidth_t::ADC_BITWIDTH_DEFAULT (*C++ enumerator*), 261
adc_cali_check_scheme (*C++ function*), 276
adc_cali_handle_t (*C++ type*), 276
adc_cali_raw_to_voltage (*C++ function*), 276
adc_cali_scheme_ver_t (*C++ enum*), 276
adc_cali_scheme_ver_t::ADC_CALI_SCHEME_VER_CURVE_ (*C++ enumerator*), 276
adc_cali_scheme_ver_t::ADC_CALI_SCHEME_VER_LINE_F (*C++ enumerator*), 276
adc_channel_t (*C++ enum*), 259
adc_channel_t::ADC_CHANNEL_0 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_1 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_2 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_3 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_4 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_5 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_6 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_7 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_8 (*C++ enumerator*), 260
adc_channel_t::ADC_CHANNEL_9 (*C++ enumerator*), 260
adc_continuous_callback_t (*C++ type*), 274
adc_continuous_channel_to_io (*C++ function*), 272
adc_continuous_clk_src_t (*C++ type*), 259
adc_continuous_config (*C++ function*), 270
adc_continuous_config_t (*C++ struct*), 272
adc_continuous_config_t::adc_pattern (*C++ member*), 273
adc_continuous_config_t::conv_mode (*C++ member*), 273
adc_continuous_config_t::format (*C++ member*), 273
adc_continuous_config_t::pattern_num (*C++ member*), 273
adc_continuous_config_t::sample_freq_hz (*C++ member*), 273
adc_continuous_deinit (*C++ function*), 271
adc_continuous_evt_cbs_t (*C++ struct*), 273
adc_continuous_evt_cbs_t::on_conv_done (*C++ member*), 274
adc_continuous_evt_cbs_t::on_pool_ovf (*C++ member*), 274

adc_continuous_evt_data_t (*C++ struct*), 273
adc_continuous_evt_data_t::conv_frame_buffer (*C++ member*), 273
adc_continuous_evt_data_t::size (*C++ member*), 273
adc_continuous_flush_pool (*C++ function*), 271
adc_continuous_handle_cfg_t (*C++ struct*), 272
adc_continuous_handle_cfg_t::conv_frame_size (*C++ member*), 272
adc_continuous_handle_cfg_t::flags (*C++ member*), 272
adc_continuous_handle_cfg_t::flush_pool (*C++ member*), 272
adc_continuous_handle_cfg_t::max_store_buf_size (*C++ member*), 272
adc_continuous_handle_t (*C++ type*), 274
adc_continuous_io_to_channel (*C++ function*), 272
adc_continuous_new_handle (*C++ function*), 270
adc_continuous_read (*C++ function*), 271
adc_continuous_register_event_callbacks (*C++ function*), 270
adc_continuous_start (*C++ function*), 271
adc_continuous_stop (*C++ function*), 271
adc_digi_convert_mode_t (*C++ enum*), 261
adc_digi_convert_mode_t::ADC_CONV_ALTER_UNIT (*C++ enumerator*), 262
adc_digi_convert_mode_t::ADC_CONV_BOTH_UNIT (*C++ enumerator*), 261
adc_digi_convert_mode_t::ADC_CONV_SINGLE_UNIT_1 (*C++ enumerator*), 261
adc_digi_convert_mode_t::ADC_CONV_SINGLE_UNIT_2 (*C++ enumerator*), 261
adc_digi_iir_filter_coeff_t (*C++ enum*), 262
adc_digi_iir_filter_coeff_t::ADC_DIGI_IIR_FILTER_COEFF_16 (*C++ enumerator*), 262
adc_digi_iir_filter_coeff_t::ADC_DIGI_IIR_FILTER_COEFF_2 (*C++ enumerator*), 262
adc_digi_iir_filter_coeff_t::ADC_DIGI_IIR_FILTER_COEFF_4 (*C++ enumerator*), 262
adc_digi_iir_filter_coeff_t::ADC_DIGI_IIR_FILTER_COEFF_64 (*C++ enumerator*), 262
adc_digi_iir_filter_coeff_t::ADC_DIGI_IIR_FILTER_COEFF_8 (*C++ enumerator*), 262
adc_digi_iir_filter_t (*C++ enum*), 262
adc_digi_iir_filter_t::ADC_DIGI_IIR_FILTER_0 (*C++ enumerator*), 262
adc_digi_iir_filter_t::ADC_DIGI_IIR_FILTER_1 (*C++ enumerator*), 262
adc_digi_output_data_t (*C++ struct*), 259
adc_digi_output_data_t::channel (*C++ member*), 259
adc_digi_output_data_t::data (*C++ member*), 259
adc_digi_output_data_t::reserved12 (*C++ member*), 259
adc_digi_output_data_t::reserved17_31 (*C++ member*), 259
adc_digi_output_data_t::type2 (*C++ member*), 259
adc_digi_output_data_t::unit (*C++ member*), 259
adc_digi_output_data_t::val (*C++ member*), 259
adc_digi_output_format_t (*C++ enum*), 262
adc_digi_output_format_t::ADC_DIGI_OUTPUT_FORMAT_ (*C++ enumerator*), 262
adc_digi_output_format_t::ADC_DIGI_OUTPUT_FORMAT_ (*C++ enumerator*), 262
adc_digi_pattern_config_t (*C++ struct*), 258
adc_digi_pattern_config_t::atten (*C++ member*), 258
adc_digi_pattern_config_t::bit_width (*C++ member*), 258
adc_digi_pattern_config_t::channel (*C++ member*), 258
adc_digi_pattern_config_t::unit (*C++ member*), 258
ADC_MAX_DELAY (*C macro*), 274
adc_monitor_id_t (*C++ enum*), 262
adc_monitor_id_t::ADC_MONITOR_0 (*C++ enumerator*), 262
adc_monitor_id_t::ADC_MONITOR_1 (*C++ enumerator*), 263
adc_monitor_mode_t (*C++ enum*), 263
adc_monitor_mode_t::ADC_MONITOR_MODE_HIGH (*C++ enumerator*), 263
adc_monitor_mode_t::ADC_MONITOR_MODE_LOW (*C++ enumerator*), 263
adc_oneshot_chan_cfg_t (*C++ struct*), 265
adc_oneshot_chan_cfg_t::atten (*C++ member*), 265
adc_oneshot_chan_cfg_t::bitwidth (*C++ member*), 265
adc_oneshot_channel_to_io (*C++ function*), 264
adc_oneshot_clk_src_t (*C++ type*), 259
adc_oneshot_config_channel (*C++ function*), 263
adc_oneshot_del_unit (*C++ function*), 264
adc_oneshot_get_calibrated_result (*C++ function*), 265
adc_oneshot_io_to_channel (*C++ function*), 264
adc_oneshot_new_unit (*C++ function*), 263
adc_oneshot_read (*C++ function*), 264
adc_oneshot_unit_handle_t (*C++ type*), 266
adc_oneshot_unit_init_cfg_t (*C++ struct*), 265
adc_oneshot_unit_init_cfg_t::clk_src (*C++ member*), 265
adc_oneshot_unit_init_cfg_t::ulp_mode

(C++ member), 265
adc_oneshot_unit_init_cfg_t::unit_id (C++ member), 265
adc_ulp_mode_t (C++ enum), 261
adc_ulp_mode_t::ADC_ULP_MODE_DISABLE (C++ enumerator), 261
adc_ulp_mode_t::ADC_ULP_MODE_FSM (C++ enumerator), 261
adc_ulp_mode_t::ADC_ULP_MODE_RISCV (C++ enumerator), 261
adc_unit_t (C++ enum), 259
adc_unit_t::ADC_UNIT_1 (C++ enumerator), 259
adc_unit_t::ADC_UNIT_2 (C++ enumerator), 259
ana_cmpr_channel_type_t (C++ enum), 285
ana_cmpr_channel_type_t::ANA_CMPR_EXT_REF_CHAN (C++ enumerator), 286
ana_cmpr_channel_type_t::ANA_CMPR_SOURCE_CHAN (C++ enumerator), 285
ana_cmpr_clk_src_t (C++ type), 285
ana_cmpr_config_t (C++ struct), 283
ana_cmpr_config_t::clk_src (C++ member), 283
ana_cmpr_config_t::cross_type (C++ member), 283
ana_cmpr_config_t::flags (C++ member), 283
ana_cmpr_config_t::intr_priority (C++ member), 283
ana_cmpr_config_t::io_loop_back (C++ member), 283
ana_cmpr_config_t::ref_src (C++ member), 283
ana_cmpr_config_t::unit (C++ member), 283
ana_cmpr_cross_cb_t (C++ type), 285
ana_cmpr_cross_event_data_t (C++ struct), 284
ana_cmpr_cross_event_data_t::cross_type (C++ member), 284
ana_cmpr_cross_type_t (C++ enum), 286
ana_cmpr_cross_type_t::ANA_CMPR_CROSS_ANY (C++ enumerator), 286
ana_cmpr_cross_type_t::ANA_CMPR_CROSS_DISABLE (C++ enumerator), 286
ana_cmpr_cross_type_t::ANA_CMPR_CROSS_NEG (C++ enumerator), 286
ana_cmpr_cross_type_t::ANA_CMPR_CROSS_POS (C++ enumerator), 286
ana_cmpr_debounce_config_t (C++ struct), 283
ana_cmpr_debounce_config_t::wait_us (C++ member), 284
ana_cmpr_del_unit (C++ function), 281
ana_cmpr_disable (C++ function), 282
ana_cmpr_enable (C++ function), 282
ana_cmpr_event_callbacks_t (C++ struct), 284
ana_cmpr_event_callbacks_t::on_cross (C++ member), 284
ana_cmpr_get_gpio (C++ function), 282
ana_cmpr_handle_t (C++ type), 285
ana_cmpr_internal_ref_config_t (C++ struct), 283
ana_cmpr_internal_ref_config_t::ref_volt (C++ member), 283
ana_cmpr_new_unit (C++ function), 280
ana_cmpr_ref_source_t (C++ enum), 285
ana_cmpr_ref_source_t::ANA_CMPR_REF_SRC_EXTERNAL (C++ enumerator), 285
ana_cmpr_ref_source_t::ANA_CMPR_REF_SRC_INTERNAL (C++ enumerator), 285
ana_cmpr_ref_voltage_t (C++ enum), 286
ana_cmpr_ref_voltage_t::ANA_CMPR_REF_VOLT_0_PCT_V (C++ enumerator), 286
ana_cmpr_ref_voltage_t::ANA_CMPR_REF_VOLT_10_PCT_ (C++ enumerator), 286
ana_cmpr_ref_voltage_t::ANA_CMPR_REF_VOLT_20_PCT_ (C++ enumerator), 286
ana_cmpr_ref_voltage_t::ANA_CMPR_REF_VOLT_30_PCT_ (C++ enumerator), 286
ana_cmpr_ref_voltage_t::ANA_CMPR_REF_VOLT_40_PCT_ (C++ enumerator), 286
ana_cmpr_ref_voltage_t::ANA_CMPR_REF_VOLT_50_PCT_ (C++ enumerator), 286
ana_cmpr_ref_voltage_t::ANA_CMPR_REF_VOLT_60_PCT_ (C++ enumerator), 286
ana_cmpr_ref_voltage_t::ANA_CMPR_REF_VOLT_70_PCT_ (C++ enumerator), 286
ana_cmpr_register_event_callbacks (C++ function), 282
ana_cmpr_set_cross_type (C++ function), 282
ana_cmpr_set_debounce (C++ function), 281
ana_cmpr_set_internal_reference (C++ function), 281
ANA_CMPR_UNIT_0 (C macro), 285
ana_cmpr_unit_t (C++ type), 285
async_memcpy_config_t (C++ struct), 1728
async_memcpy_config_t::backlog (C++ member), 1728
async_memcpy_config_t::dma_burst_size (C++ member), 1728
async_memcpy_config_t::flags (C++ member), 1728
async_memcpy_config_t::psram_trans_align (C++ member), 1728
async_memcpy_config_t::sram_trans_align (C++ member), 1728
ASYNC_MEMCPY_DEFAULT_CONFIG (C macro), 1729
async_memcpy_etm_event_t (C++ enum), 1729
async_memcpy_etm_event_t::ASYNC_MEMCPY_ETM_EVENT_ (C++ enumerator), 1729
async_memcpy_event_t (C++ struct), 1728
async_memcpy_event_t::data (C++ member), 1728

async_memcpy_handle_t (C++ type), 1729
async_memcpy_isr_cb_t (C++ type), 1729

B

bootloader_fill_random (C++ function), 1680
bootloader_random_disable (C++ function), 1680
bootloader_random_enable (C++ function), 1680
bridgeif_config (C++ struct), 242
bridgeif_config::max_fdb_dyn_entries (C++ member), 242
bridgeif_config::max_fdb_sta_entries (C++ member), 242
bridgeif_config::max_ports (C++ member), 242
bridgeif_config_t (C++ type), 245

C

CHIP_FEATURE_BLE (C macro), 1645
CHIP_FEATURE_BT (C macro), 1646
CHIP_FEATURE_EMB_FLASH (C macro), 1645
CHIP_FEATURE_EMB_PSRAM (C macro), 1646
CHIP_FEATURE_IEEE802154 (C macro), 1646
CHIP_FEATURE_WIFI_BGN (C macro), 1645
CONFIG_HEAP_TRACING_STACK_DEPTH (C macro), 1609

D

dedic_gpio_bundle_config_t (C++ struct), 358
dedic_gpio_bundle_config_t::array_size (C++ member), 358
dedic_gpio_bundle_config_t::flags (C++ member), 359
dedic_gpio_bundle_config_t::gpio_array (C++ member), 358
dedic_gpio_bundle_config_t::in_en (C++ member), 359
dedic_gpio_bundle_config_t::in_invert (C++ member), 359
dedic_gpio_bundle_config_t::out_en (C++ member), 359
dedic_gpio_bundle_config_t::out_invert (C++ member), 359
dedic_gpio_bundle_handle_t (C++ type), 359
dedic_gpio_bundle_read_in (C++ function), 358
dedic_gpio_bundle_read_out (C++ function), 358
dedic_gpio_bundle_write (C++ function), 358
dedic_gpio_del_bundle (C++ function), 357
dedic_gpio_get_in_mask (C++ function), 357
dedic_gpio_get_in_offset (C++ function), 357
dedic_gpio_get_out_mask (C++ function), 357
dedic_gpio_get_out_offset (C++ function), 357
dedic_gpio_new_bundle (C++ function), 357
DEFAULT_HTTP_BUF_SIZE (C macro), 85
DHT_TC_NUM (C macro), 746
DHT_TH_NUM (C macro), 746

E

EFD_SUPPORT_ISR (C macro), 1341
efuse_hal_blk_version (C++ function), 1363
efuse_hal_chip_revision (C++ function), 1363
efuse_hal_flash_encryption_enabled (C++ function), 1363
efuse_hal_get_disable_blk_version_major (C++ function), 1363
efuse_hal_get_disable_wafer_version_major (C++ function), 1363
efuse_hal_get_mac (C++ function), 1363
efuse_hal_get_major_chip_version (C++ function), 1363
efuse_hal_get_minor_chip_version (C++ function), 1363
efuse_hal_set_ecdsa_key (C++ function), 1364
emac_esp_smi_gpio_config_t (C++ struct), 189
emac_esp_smi_gpio_config_t::mdc_num (C++ member), 189
emac_esp_smi_gpio_config_t::mdio_num (C++ member), 189
emac_rmii_clock_gpio_t (C++ type), 192
emac_rmii_clock_mode_t (C++ enum), 192
emac_rmii_clock_mode_t::EMAC_CLK_DEFAULT (C++ enumerator), 192
emac_rmii_clock_mode_t::EMAC_CLK_EXT_IN (C++ enumerator), 192
emac_rmii_clock_mode_t::EMAC_CLK_OUT (C++ enumerator), 192
eNotifyAction (C++ enum), 1466
eNotifyAction::eIncrement (C++ enumerator), 1466
eNotifyAction::eNoAction (C++ enumerator), 1466
eNotifyAction::eSetBits (C++ enumerator), 1466
eNotifyAction::eSetValueWithoutOverwrite (C++ enumerator), 1466
eNotifyAction::eSetValueWithOverwrite (C++ enumerator), 1466
eSleepModeStatus (C++ enum), 1466
eSleepModeStatus::eAbortSleep (C++ enumerator), 1467
eSleepModeStatus::eStandardSleep (C++ enumerator), 1467
esp_alloc_failed_hook_t (C++ type), 1581
ESP_APP_DESC_MAGIC_WORD (C macro), 1654
esp_app_desc_t (C++ struct), 1653
esp_app_desc_t::app_elf_sha256 (C++ member), 1653

esp_app_desc_t::date (C++ member), 1653
esp_app_desc_t::idf_ver (C++ member), 1653
esp_app_desc_t::magic_word (C++ member), 1653
esp_app_desc_t::max_efuse_blk_rev_full (C++ member), 1653
esp_app_desc_t::min_efuse_blk_rev_full (C++ member), 1653
esp_app_desc_t::project_name (C++ member), 1653
esp_app_desc_t::reserv1 (C++ member), 1653
esp_app_desc_t::reserv2 (C++ member), 1653
esp_app_desc_t::secure_version (C++ member), 1653
esp_app_desc_t::time (C++ member), 1653
esp_app_desc_t::version (C++ member), 1653
esp_app_get_description (C++ function), 1652
esp_app_get_elf_sha256 (C++ function), 1652
esp_app_get_elf_sha256_str (C++ function), 1652
esp_apptrace_buffer_get (C++ function), 1353
esp_apptrace_buffer_put (C++ function), 1354
esp_apptrace_dest_t (C++ enum), 1357
esp_apptrace_dest_t::ESP_APPTRACE_DEST_JTAG (C++ enumerator), 1357
esp_apptrace_dest_t::ESP_APPTRACE_DEST_MAX (C++ enumerator), 1357
esp_apptrace_dest_t::ESP_APPTRACE_DEST_NUM (C++ enumerator), 1357
esp_apptrace_dest_t::ESP_APPTRACE_DEST_TRAX (C++ enumerator), 1357
esp_apptrace_dest_t::ESP_APPTRACE_DEST_UART (C++ enumerator), 1357
esp_apptrace_down_buffer_config (C++ function), 1353
esp_apptrace_down_buffer_get (C++ function), 1355
esp_apptrace_down_buffer_put (C++ function), 1355
esp_apptrace_fclose (C++ function), 1355
esp_apptrace_feof (C++ function), 1356
esp_apptrace_flush (C++ function), 1354
esp_apptrace_flush_nolock (C++ function), 1354
esp_apptrace_fopen (C++ function), 1355
esp_apptrace_fread (C++ function), 1356
esp_apptrace_fseek (C++ function), 1356
esp_apptrace_fstop (C++ function), 1356
esp_apptrace_ftell (C++ function), 1356
esp_apptrace_fwrite (C++ function), 1356
esp_apptrace_host_is_connected (C++ function), 1355
esp_apptrace_init (C++ function), 1353
esp_apptrace_read (C++ function), 1355
esp_apptrace_vprintf (C++ function), 1354
esp_apptrace_vprintf_to (C++ function), 1354
esp_apptrace_write (C++ function), 1354
esp_async_memcpy (C++ function), 1727
esp_async_memcpy_install (C++ function), 1727
esp_async_memcpy_install_gdma_ahb (C++ function), 1726
esp_async_memcpy_install_gdma_axi (C++ function), 1726
esp_async_memcpy_new_etm_event (C++ function), 1727
esp_async_memcpy_uninstall (C++ function), 1727
esp_base_mac_addr_get (C++ function), 1642
esp_base_mac_addr_set (C++ function), 1642
ESP_BOOTLOADER_DESC_MAGIC_BYTE (C macro), 1352
esp_bootloader_desc_t (C++ struct), 1352
esp_bootloader_desc_t::date_time (C++ member), 1352
esp_bootloader_desc_t::idf_ver (C++ member), 1352
esp_bootloader_desc_t::magic_byte (C++ member), 1352
esp_bootloader_desc_t::reserved (C++ member), 1352
esp_bootloader_desc_t::reserved2 (C++ member), 1352
esp_bootloader_desc_t::version (C++ member), 1352
esp_bootloader_get_description (C++ function), 1352
esp_cache_msync (C++ function), 1596
ESP_CACHE_MSYNC_FLAG_DIR_C2M (C macro), 1597
ESP_CACHE_MSYNC_FLAG_DIR_M2C (C macro), 1597
ESP_CACHE_MSYNC_FLAG_INVALIDATE (C macro), 1597
ESP_CACHE_MSYNC_FLAG_TYPE_DATA (C macro), 1597
ESP_CACHE_MSYNC_FLAG_TYPE_INST (C macro), 1597
ESP_CACHE_MSYNC_FLAG_UNALIGNED (C macro), 1597
esp_cam_ctlr_csi_config_t (C++ struct), 515
esp_cam_ctlr_csi_config_t::bk_buffer_dis (C++ member), 516
esp_cam_ctlr_csi_config_t::byte_swap_en (C++ member), 516
esp_cam_ctlr_csi_config_t::clk_src (C++ member), 515
esp_cam_ctlr_csi_config_t::ctlr_id

(C++ member), 515
esp_cam_ctlr_csi_config_t::data_lane_num (C++ member), 515
esp_cam_ctlr_csi_config_t::h_res (C++ member), 515
esp_cam_ctlr_csi_config_t::input_data_color_type (C++ member), 516
esp_cam_ctlr_csi_config_t::lane_bit_rate_mbps (C++ member), 516
esp_cam_ctlr_csi_config_t::output_data_color_type (C++ member), 516
esp_cam_ctlr_csi_config_t::queue_items (C++ member), 516
esp_cam_ctlr_csi_config_t::v_res (C++ member), 515
esp_cam_ctlr_csi_config_t::[anonymous] (C++ member), 516
esp_cam_ctlr_del (C++ function), 512
esp_cam_ctlr_disable (C++ function), 512
esp_cam_ctlr_enable (C++ function), 512
esp_cam_ctlr_evt_cbs_t (C++ struct), 514
esp_cam_ctlr_evt_cbs_t::on_get_new_trans (C++ member), 514
esp_cam_ctlr_evt_cbs_t::on_trans_finished (C++ member), 514
esp_cam_ctlr_get_frame_buffer (C++ function), 513
esp_cam_ctlr_get_frame_buffer_len (C++ function), 513
esp_cam_ctlr_handle_t (C++ type), 515
ESP_CAM_CTLR_MAX_DELAY (C macro), 514
esp_cam_ctlr_receive (C++ function), 512
esp_cam_ctlr_register_event_callbacks (C++ function), 513
esp_cam_ctlr_start (C++ function), 512
esp_cam_ctlr_stop (C++ function), 512
esp_cam_ctlr_trans_t (C++ struct), 514
esp_cam_ctlr_trans_t::buffer (C++ member), 514
esp_cam_ctlr_trans_t::buflen (C++ member), 514
esp_cam_ctlr_trans_t::received_size (C++ member), 514
esp_cam_new_csi_ctlr (C++ function), 515
esp_chip_id_t (C++ enum), 1348
esp_chip_id_t::ESP_CHIP_ID_ESP32 (C++ enumerator), 1348
esp_chip_id_t::ESP_CHIP_ID_ESP32C2 (C++ enumerator), 1349
esp_chip_id_t::ESP_CHIP_ID_ESP32C3 (C++ enumerator), 1348
esp_chip_id_t::ESP_CHIP_ID_ESP32C6 (C++ enumerator), 1349
esp_chip_id_t::ESP_CHIP_ID_ESP32H2 (C++ enumerator), 1349
esp_chip_id_t::ESP_CHIP_ID_ESP32P4 (C++ enumerator), 1349
esp_chip_id_t::ESP_CHIP_ID_ESP32S2 (C++ enumerator), 1348
esp_chip_id_t::ESP_CHIP_ID_ESP32S3 (C++ enumerator), 1348
esp_chip_id_t::ESP_CHIP_ID_INVALID (C++ enumerator), 1349
esp_chip_info (C++ function), 1645
esp_chip_info_t (C++ struct), 1645
esp_chip_info_t::cores (C++ member), 1645
esp_chip_info_t::features (C++ member), 1645
esp_chip_info_t::model (C++ member), 1645
esp_chip_info_t::revision (C++ member), 1645
esp_chip_model_t (C++ enum), 1646
esp_chip_model_t::CHIP_ESP32 (C++ enumerator), 1646
esp_chip_model_t::CHIP_ESP32C2 (C++ enumerator), 1646
esp_chip_model_t::CHIP_ESP32C3 (C++ enumerator), 1646
esp_chip_model_t::CHIP_ESP32C6 (C++ enumerator), 1646
esp_chip_model_t::CHIP_ESP32C61 (C++ enumerator), 1646
esp_chip_model_t::CHIP_ESP32H2 (C++ enumerator), 1646
esp_chip_model_t::CHIP_ESP32P4 (C++ enumerator), 1646
esp_chip_model_t::CHIP_ESP32S2 (C++ enumerator), 1646
esp_chip_model_t::CHIP_ESP32S3 (C++ enumerator), 1646
esp_chip_model_t::CHIP_POSIX_LINUX (C++ enumerator), 1646
esp_clk_tree_src_freq_precision_t (C++ enum), 306
esp_clk_tree_src_freq_precision_t::ESP_CLK_TREE_S (C++ enumerator), 306
esp_clk_tree_src_freq_precision_t::ESP_CLK_TREE_S (C++ enumerator), 306
esp_clk_tree_src_freq_precision_t::ESP_CLK_TREE_S (C++ enumerator), 306
esp_clk_tree_src_freq_precision_t::ESP_CLK_TREE_S (C++ enumerator), 306
esp_clk_tree_src_get_freq_hz (C++ function), 305
esp_console_cmd_func_t (C++ type), 1374
esp_console_cmd_func_with_context_t (C++ type), 1374
esp_console_cmd_register (C++ function), 1368
esp_console_cmd_t (C++ struct), 1372
esp_console_cmd_t::argtable (C++ member), 1373
esp_console_cmd_t::command (C++ member), 1372
esp_console_cmd_t::context (C++ member), 1373

esp_console_cmd_t::func (C++ member), 1373
esp_console_cmd_t::func_w_context (C++ member), 1373
esp_console_cmd_t::help (C++ member), 1373
esp_console_cmd_t::hint (C++ member), 1373
ESP_CONSOLE_CONFIG_DEFAULT (C macro), 1373
esp_console_config_t (C++ struct), 1371
esp_console_config_t::heap_alloc_caps (C++ member), 1371
esp_console_config_t::hint_bold (C++ member), 1371
esp_console_config_t::hint_color (C++ member), 1371
esp_console_config_t::max_cmdline_args (C++ member), 1371
esp_console_config_t::max_cmdline_length (C++ member), 1371
esp_console_deinit (C++ function), 1367
ESP_CONSOLE_DEV_CDC_CONFIG_DEFAULT (C macro), 1374
ESP_CONSOLE_DEV_UART_CONFIG_DEFAULT (C macro), 1374
esp_console_dev_uart_config_t (C++ struct), 1372
esp_console_dev_uart_config_t::baud_rate (C++ member), 1372
esp_console_dev_uart_config_t::channel (C++ member), 1372
esp_console_dev_uart_config_t::rx_gpio_num (C++ member), 1372
esp_console_dev_uart_config_t::tx_gpio_num (C++ member), 1372
esp_console_dev_usb_cdc_config_t (C++ struct), 1372
ESP_CONSOLE_DEV_USB_SERIAL_JTAG_CONFIG_DEFAULT (C macro), 1374
esp_console_dev_usb_serial_jtag_config_t (C++ struct), 1372
esp_console_get_completion (C++ function), 1369
esp_console_get_hint (C++ function), 1369
esp_console_init (C++ function), 1367
esp_console_new_repl_uart (C++ function), 1369
esp_console_new_repl_usb_cdc (C++ function), 1370
esp_console_new_repl_usb_serial_jtag (C++ function), 1370
esp_console_register_help_command (C++ function), 1369
ESP_CONSOLE_REPL_CONFIG_DEFAULT (C macro), 1373
esp_console_repl_config_t (C++ struct), 1371
esp_console_repl_config_t::history_save_path (C++ member), 1371
esp_console_repl_config_t::max_cmdline_length (C++ member), 1372
esp_console_repl_config_t::max_history_len (C++ member), 1371
esp_console_repl_config_t::prompt (C++ member), 1372
esp_console_repl_config_t::task_core_id (C++ member), 1372
esp_console_repl_config_t::task_priority (C++ member), 1371
esp_console_repl_config_t::task_stack_size (C++ member), 1371
esp_console_repl_s (C++ struct), 1373
esp_console_repl_s::del (C++ member), 1373
esp_console_repl_t (C++ type), 1374
esp_console_run (C++ function), 1368
esp_console_split_argv (C++ function), 1368
esp_console_start_repl (C++ function), 1371
esp_cpu_branch_prediction_disable (C++ function), 1651
esp_cpu_branch_prediction_enable (C++ function), 1651
esp_cpu_clear_breakpoint (C++ function), 1650
esp_cpu_clear_watchpoint (C++ function), 1650
esp_cpu_compare_and_set (C++ function), 1650
esp_cpu_configure_region_protection (C++ function), 1649
esp_cpu_cycle_count_t (C++ type), 1651
esp_cpu_dbgr_break (C++ function), 1650
esp_cpu_dbgr_is_attached (C++ function), 1650
esp_cpu_get_call_addr (C++ function), 1650
esp_cpu_get_core_id (C++ function), 1647
esp_cpu_get_cycle_count (C++ function), 1647
esp_cpu_get_sp (C++ function), 1647
ESP_CPU_INTR_DESC_FLAG_RESVD (C macro), 1651
ESP_CPU_INTR_DESC_FLAG_SPECIAL (C macro), 1651
esp_cpu_intr_desc_t (C++ struct), 1651
esp_cpu_intr_desc_t::flags (C++ member), 1651
esp_cpu_intr_desc_t::priority (C++ member), 1651
esp_cpu_intr_desc_t::type (C++ member), 1651
esp_cpu_intr_disable (C++ function), 1649
esp_cpu_intr_edge_ack (C++ function), 1649
esp_cpu_intr_enable (C++ function), 1649
esp_cpu_intr_get_desc (C++ function), 1647
esp_cpu_intr_get_enabled_mask (C++ function), 1649

esp_cpu_intr_get_handler_arg (C++ function), 1649
esp_cpu_intr_get_priority (C++ function), 1648
esp_cpu_intr_get_type (C++ function), 1648
esp_cpu_intr_handler_t (C++ type), 1651
esp_cpu_intr_has_handler (C++ function), 1648
esp_cpu_intr_set_handler (C++ function), 1649
esp_cpu_intr_set_ivt_addr (C++ function), 1648
esp_cpu_intr_set_mtvt_addr (C++ function), 1648
esp_cpu_intr_set_priority (C++ function), 1648
esp_cpu_intr_set_type (C++ function), 1648
esp_cpu_intr_type_t (C++ enum), 1652
esp_cpu_intr_type_t::ESP_CPU_INTR_TYPE_EDGE (C++ enumerator), 1652
esp_cpu_intr_type_t::ESP_CPU_INTR_TYPE_LEVEL (C++ enumerator), 1652
esp_cpu_intr_type_t::ESP_CPU_INTR_TYPE_NA (C++ enumerator), 1652
esp_cpu_pc_to_addr (C++ function), 1647
esp_cpu_reset (C++ function), 1647
esp_cpu_set_breakpoint (C++ function), 1649
esp_cpu_set_cycle_count (C++ function), 1647
esp_cpu_set_watchpoint (C++ function), 1650
esp_cpu_stall (C++ function), 1647
esp_cpu_unstall (C++ function), 1647
esp_cpu_wait_for_intr (C++ function), 1647
esp_cpu_watchpoint_trigger_t (C++ enum), 1652
esp_cpu_watchpoint_trigger_t::ESP_CPU_WATCHPOINT_ACCESS (C++ enumerator), 1652
esp_cpu_watchpoint_trigger_t::ESP_CPU_WATCHPOINT_LOAD (C++ enumerator), 1652
esp_cpu_watchpoint_trigger_t::ESP_CPU_WATCHPOINT_STORE (C++ enumerator), 1652
esp_crt_bundle_attach (C++ function), 114
esp_crt_bundle_detach (C++ function), 114
esp_crt_bundle_in_use (C++ function), 114
esp_crt_bundle_set (C++ function), 114
esp_deep_sleep (C++ function), 1692
esp_deep_sleep_cb_t (C++ type), 1693
esp_deep_sleep_deregister_hook (C++ function), 1692
esp_deep_sleep_disable_rom_logging (C++ function), 1693
esp_deep_sleep_enable_gpio_wakeup (C++ function), 1689
esp_deep_sleep_register_hook (C++ function), 1692
esp_deep_sleep_start (C++ function), 1691
esp_deep_sleep_try (C++ function), 1691
esp_deep_sleep_try_to_start (C++ function), 1691
esp_deep_sleep_wake_stub_fn_t (C++ type), 1693
esp_deepsleep_gpio_wake_up_mode_t (C++ enum), 1694
esp_deepsleep_gpio_wake_up_mode_t::ESP_GPIO_WAKEU (C++ enumerator), 1694
esp_deepsleep_gpio_wake_up_mode_t::ESP_GPIO_WAKEU (C++ enumerator), 1694
esp_default_wake_deep_sleep (C++ function), 1693
esp_deregister_freertos_idle_hook (C++ function), 1565
esp_deregister_freertos_idle_hook_for_cpu (C++ function), 1565
esp_deregister_freertos_tick_hook (C++ function), 1566
esp_deregister_freertos_tick_hook_for_cpu (C++ function), 1565
esp_derive_local_mac (C++ function), 1643
esp_digital_signature_data (C++ struct), 367
esp_digital_signature_data::c (C++ member), 367
esp_digital_signature_data::iv (C++ member), 367
esp_digital_signature_data::rsa_length (C++ member), 367
esp_digital_signature_length_t (C++ enum), 369
esp_digital_signature_length_t::ESP_DS_RSA_1024 (C++ enumerator), 369
esp_digital_signature_length_t::ESP_DS_RSA_2048 (C++ enumerator), 369
esp_digital_signature_length_t::ESP_DS_RSA_3072 (C++ enumerator), 369
esp_digital_signature_length_t::ESP_DS_RSA_4096 (C++ enumerator), 369
ESP_DRAM_LOGD (C macro), 1635
ESP_DRAM_LOGE (C macro), 1635
ESP_DRAM_LOGI (C macro), 1635
ESP_DRAM_LOGV (C macro), 1636
ESP_DRAM_LOGW (C macro), 1635
ESP_DS_C_LEN (C macro), 368
esp_ds_context_t (C++ type), 368
esp_ds_data_t (C++ type), 368
esp_ds_encrypt_params (C++ function), 366
esp_ds_finish_sign (C++ function), 366
esp_ds_is_busy (C++ function), 366
ESP_DS_IV_BIT_LEN (C macro), 368
ESP_DS_IV_LEN (C macro), 368
esp_ds_p_data_t (C++ struct), 367
esp_ds_p_data_t::length (C++ member), 368
esp_ds_p_data_t::M (C++ member), 368
esp_ds_p_data_t::M_prime (C++ member), 368
esp_ds_p_data_t::Rb (C++ member), 368
esp_ds_p_data_t::Y (C++ member), 368

esp_ds_sign (C++ function), 365
ESP_DS_SIGNATURE_L_BIT_LEN (C macro), 368
ESP_DS_SIGNATURE_M_PRIME_BIT_LEN (C macro), 368
ESP_DS_SIGNATURE_MAX_BIT_LEN (C macro), 368
ESP_DS_SIGNATURE_MD_BIT_LEN (C macro), 368
ESP_DS_SIGNATURE_PADDING_BIT_LEN (C macro), 368
esp_ds_start_sign (C++ function), 365
ESP_EARLY_LOGD (C macro), 1634
ESP_EARLY_LOGE (C macro), 1633
ESP_EARLY_LOGI (C macro), 1634
ESP_EARLY_LOGV (C macro), 1634
ESP_EARLY_LOGW (C macro), 1634
esp_efuse_batch_write_begin (C++ function), 1398
esp_efuse_batch_write_cancel (C++ function), 1399
esp_efuse_batch_write_commit (C++ function), 1399
esp_efuse_block_is_empty (C++ function), 1399
esp_efuse_block_t (C++ enum), 1390
esp_efuse_block_t::EFUSE_BLK0 (C++ enumerator), 1390
esp_efuse_block_t::EFUSE_BLK1 (C++ enumerator), 1390
esp_efuse_block_t::EFUSE_BLK10 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK2 (C++ enumerator), 1390
esp_efuse_block_t::EFUSE_BLK3 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK4 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK5 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK6 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK7 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK8 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK9 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_KEY0 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_KEY1 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_KEY2 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_KEY3 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_KEY4 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_KEY5 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_KEY_MAX (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_MAX (C++ enumerator), 1392
esp_efuse_block_t::EFUSE_BLK_SYS_DATA_PART1 (C++ enumerator), 1390
esp_efuse_block_t::EFUSE_BLK_SYS_DATA_PART2 (C++ enumerator), 1391
esp_efuse_block_t::EFUSE_BLK_USER_DATA (C++ enumerator), 1391
esp_efuse_check_errors (C++ function), 1403
esp_efuse_check_secure_version (C++ function), 1398
esp_efuse_coding_scheme_t (C++ enum), 1392
esp_efuse_coding_scheme_t::EFUSE_CODING_SCHEME_NO (C++ enumerator), 1392
esp_efuse_coding_scheme_t::EFUSE_CODING_SCHEME_RS (C++ enumerator), 1392
esp_efuse_count_unused_key_blocks (C++ function), 1401
esp_efuse_desc_t (C++ struct), 1404
esp_efuse_desc_t::bit_count (C++ member), 1404
esp_efuse_desc_t::bit_start (C++ member), 1404
esp_efuse_desc_t::efuse_block (C++ member), 1404
esp_efuse_destroy_block (C++ function), 1403
esp_efuse_disable_rom_download_mode (C++ function), 1397
esp_efuse_enable_rom_secure_download_mode (C++ function), 1397
esp_efuse_find_purpose (C++ function), 1400
esp_efuse_find_unused_key_block (C++ function), 1401
esp_efuse_get_coding_scheme (C++ function), 1396
esp_efuse_get_digest_revoke (C++ function), 1401
esp_efuse_get_field_size (C++ function), 1395
esp_efuse_get_key (C++ function), 1401
esp_efuse_get_key_dis_read (C++ function), 1399
esp_efuse_get_key_dis_write (C++ function), 1400
esp_efuse_get_key_purpose (C++ function), 1400
esp_efuse_get_keypurpose_dis_write (C++ function), 1400
esp_efuse_get_pkg_ver (C++ function), 1396
esp_efuse_get_purpose_field (C++ function), 1400
esp_efuse_get_write_protect_of_digest_revoke (C++ function), 1401

esp_efuse_key_block_unused (*C++ function*), 1400
esp_efuse_mac_get_custom (*C++ function*), 1643
esp_efuse_mac_get_default (*C++ function*), 1643
esp_efuse_purpose_t (*C++ enum*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_ECDSA_KEY (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_HMAC_UP (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_KM_INIT_KEY (*C++ enumerator*), 1393
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_MAX (*C++ enumerator*), 1393
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_DIGEST0 (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_DIGEST1 (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_DIGEST2 (*C++ enumerator*), 1393
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_USER (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_XTS_AES_128_KEY (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_1 (*C++ enumerator*), 1392
esp_efuse_purpose_t::ESP_EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_2 (*C++ enumerator*), 1392
esp_efuse_read_block (*C++ function*), 1396
esp_efuse_read_field_bit (*C++ function*), 1393
esp_efuse_read_field_blob (*C++ function*), 1393
esp_efuse_read_field_cnt (*C++ function*), 1394
esp_efuse_read_reg (*C++ function*), 1395
esp_efuse_read_secure_version (*C++ function*), 1398
esp_efuse_reset (*C++ function*), 1397
esp_efuse_rom_log_scheme_t (*C++ enum*), 1405
esp_efuse_rom_log_scheme_t::ESP_EFUSE_ROM_LOG_ALWAYS_ON (*C++ enumerator*), 1405
esp_efuse_rom_log_scheme_t::ESP_EFUSE_ROM_LOG_ON_GPIO_HIGH (*C++ enumerator*), 1405
esp_efuse_rom_log_scheme_t::ESP_EFUSE_ROM_LOG_ON_GPIO_LOW (*C++ enumerator*), 1405
esp_efuse_set_digest_revoke (*C++ function*), 1401
esp_efuse_set_key_dis_read (*C++ function*), 1399
esp_efuse_set_key_dis_write (*C++ function*), 1400
esp_efuse_set_key_purpose (*C++ function*), 1401
esp_efuse_set_keypurpose_dis_write (*C++ function*), 1401
esp_efuse_set_read_protect (*C++ function*), 1395
esp_efuse_set_rom_log_scheme (*C++ function*), 1397
esp_efuse_set_write_protect (*C++ function*), 1395
esp_efuse_set_write_protect_of_digest_revoke (*C++ function*), 1402
esp_efuse_update_secure_version (*C++ function*), 1398
esp_efuse_write_block (*C++ function*), 1396
esp_efuse_write_field_bit (*C++ function*), 1395
esp_efuse_write_field_blob (*C++ function*), 1394
esp_efuse_write_field_cnt (*C++ function*), 1394
esp_efuse_write_keys (*C++ function*), 1402
esp_efuse_write_reg (*C++ function*), 1395
ESP_ERR_CODING (*C macro*), 1404
ESP_ERR_EFUSE (*C macro*), 1404
ESP_ERR_EFUSE_CNT_IS_FULL (*C macro*), 1404
ESP_ERR_EFUSE_REPEATED_PROG (*C macro*), 1404
ESP_ERR_ESP_NETIF_BASE (*C macro*), 244
ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED (*C macro*), 244
ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED (*C macro*), 244
ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED (*C macro*), 244
ESP_ERR_ESP_NETIF_DHCPC_START_FAILED (*C macro*), 244
ESP_ERR_ESP_NETIF_DHCPS_START_FAILED (*C macro*), 244
ESP_ERR_ESP_NETIF_DNS_NOT_CONFIGURED (*C macro*), 244
ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED (*C macro*), 244
ESP_ERR_ESP_NETIF_IF_NOT_READY (*C macro*), 244
ESP_ERR_ESP_NETIF_INIT_FAILED (*C macro*), 244
ESP_ERR_ESP_NETIF_INVALID_PARAMS (*C macro*), 244
ESP_ERR_ESP_NETIF_IP6_ADDR_FAILED (*C macro*), 244
ESP_ERR_ESP_NETIF_MLD6_FAILED (*C macro*),

244
ESP_ERR_ESP_NETIF_NO_MEM (*C macro*), 244
ESP_ERR_ESP_TLS_BASE (*C macro*), 69
ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET (*C macro*), 69
ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME (*C macro*), 69
ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT (*C macro*), 69
ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST (*C macro*), 69
ESP_ERR_ESP_TLS_SE_FAILED (*C macro*), 70
ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED (*C macro*), 69
ESP_ERR_ESP_TLS_TCP_CLOSED_FIN (*C macro*), 70
ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY (*C macro*), 69
ESP_ERR_FLASH_BASE (*C macro*), 1408
ESP_ERR_FLASH_NOT_INITIALISED (*C macro*), 690
ESP_ERR_FLASH_OP_FAIL (*C macro*), 683
ESP_ERR_FLASH_OP_TIMEOUT (*C macro*), 683
ESP_ERR_FLASH_PROTECTED (*C macro*), 690
ESP_ERR_FLASH_UNSUPPORTED_CHIP (*C macro*), 690
ESP_ERR_FLASH_UNSUPPORTED_HOST (*C macro*), 690
ESP_ERR_HTTP_BASE (*C macro*), 85
ESP_ERR_HTTP_CONNECT (*C macro*), 85
ESP_ERR_HTTP_CONNECTING (*C macro*), 86
ESP_ERR_HTTP_CONNECTION_CLOSED (*C macro*), 86
ESP_ERR_HTTP_EAGAIN (*C macro*), 86
ESP_ERR_HTTP_FETCH_HEADER (*C macro*), 85
ESP_ERR_HTTP_INVALID_TRANSPORT (*C macro*), 85
ESP_ERR_HTTP_MAX_REDIRECT (*C macro*), 85
ESP_ERR_HTTP_WRITE_DATA (*C macro*), 85
ESP_ERR_HTTPD_ALLOC_MEM (*C macro*), 139
ESP_ERR_HTTPD_BASE (*C macro*), 139
ESP_ERR_HTTPD_HANDLER_EXISTS (*C macro*), 139
ESP_ERR_HTTPD_HANDLERS_FULL (*C macro*), 139
ESP_ERR_HTTPD_INVALID_REQ (*C macro*), 139
ESP_ERR_HTTPD_RESP_HDR (*C macro*), 139
ESP_ERR_HTTPD_RESP_SEND (*C macro*), 139
ESP_ERR_HTTPD_RESULT_TRUNC (*C macro*), 139
ESP_ERR_HTTPD_TASK (*C macro*), 139
ESP_ERR_HTTPS_OTA_BASE (*C macro*), 1414
ESP_ERR_HTTPS_OTA_IN_PROGRESS (*C macro*), 1414
ESP_ERR_HW_CRYPTO_BASE (*C macro*), 1408
ESP_ERR_INVALID_ARG (*C macro*), 1407
ESP_ERR_INVALID_CRC (*C macro*), 1407
ESP_ERR_INVALID_MAC (*C macro*), 1407
ESP_ERR_INVALID_RESPONSE (*C macro*), 1407
ESP_ERR_INVALID_SIZE (*C macro*), 1407
ESP_ERR_INVALID_STATE (*C macro*), 1407
ESP_ERR_INVALID_VERSION (*C macro*), 1407
ESP_ERR_MBEDTLS_CERT_PARTLY_OK (*C macro*), 70
ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_CONF_OWN_CERT_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_SETUP_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_TICKET_SETUP_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_SSL_WRITE_FAILED (*C macro*), 70
ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED (*C macro*), 70
ESP_ERR_MEMPROT_BASE (*C macro*), 1408
ESP_ERR_MESH_BASE (*C macro*), 1407
ESP_ERR_NO_MEM (*C macro*), 1407
ESP_ERR_NOT_ALLOWED (*C macro*), 1407
ESP_ERR_NOT_ENOUGH_UNUSED_KEY_BLOCKS (*C macro*), 1404
ESP_ERR_NOT_FINISHED (*C macro*), 1407
ESP_ERR_NOT_FOUND (*C macro*), 1407
ESP_ERR_NOT_SUPPORTED (*C macro*), 1407
ESP_ERR_NVS_BASE (*C macro*), 1288
ESP_ERR_NVS_CONTENT_DIFFERS (*C macro*), 1289
ESP_ERR_NVS_CORRUPT_KEY_PART (*C macro*), 1289
ESP_ERR_NVS_ENCR_NOT_SUPPORTED (*C macro*), 1289
ESP_ERR_NVS_INVALID_HANDLE (*C macro*), 1288
ESP_ERR_NVS_INVALID_LENGTH (*C macro*), 1288
ESP_ERR_NVS_INVALID_NAME (*C macro*), 1288
ESP_ERR_NVS_INVALID_STATE (*C macro*), 1288
ESP_ERR_NVS_KEY_TOO_LONG (*C macro*), 1288
ESP_ERR_NVS_KEYS_NOT_INITIALIZED (*C macro*), 1289
ESP_ERR_NVS_NEW_VERSION_FOUND (*C macro*), 1289
ESP_ERR_NVS_NO_FREE_PAGES (*C macro*), 1288
ESP_ERR_NVS_NOT_ENOUGH_SPACE (*C macro*), 1288
ESP_ERR_NVS_NOT_FOUND (*C macro*), 1288

ESP_ERR_NVS_NOT_INITIALIZED (*C macro*), 1288
ESP_ERR_NVS_PAGE_FULL (*C macro*), 1288
ESP_ERR_NVS_PART_NOT_FOUND (*C macro*), 1289
ESP_ERR_NVS_READ_ONLY (*C macro*), 1288
ESP_ERR_NVS_REMOVE_FAILED (*C macro*), 1288
ESP_ERR_NVS_SEC_BASE (*C macro*), 1296
ESP_ERR_NVS_SEC_HMAC_KEY_BLK_ALREADY_USED (*C macro*), 1296
ESP_ERR_NVS_SEC_HMAC_KEY_GENERATION_FAILED (*C macro*), 1296
ESP_ERR_NVS_SEC_HMAC_KEY_NOT_FOUND (*C macro*), 1296
ESP_ERR_NVS_SEC_HMAC_XTS_KEYS_DERIV_FAILED (*C macro*), 1296
ESP_ERR_NVS_TYPE_MISMATCH (*C macro*), 1288
ESP_ERR_NVS_VALUE_TOO_LONG (*C macro*), 1289
ESP_ERR_NVS_WRONG_ENCRYPTION (*C macro*), 1289
ESP_ERR_NVS_XTS_CFG_FAILED (*C macro*), 1289
ESP_ERR_NVS_XTS_CFG_NOT_FOUND (*C macro*), 1289
ESP_ERR_NVS_XTS_DECR_FAILED (*C macro*), 1289
ESP_ERR_NVS_XTS_ENCR_FAILED (*C macro*), 1289
ESP_ERR_OTA_BASE (*C macro*), 1665
ESP_ERR_OTA_PARTITION_CONFLICT (*C macro*), 1665
ESP_ERR_OTA_ROLLBACK_FAILED (*C macro*), 1665
ESP_ERR_OTA_ROLLBACK_INVALID_STATE (*C macro*), 1665
ESP_ERR_OTA_SELECT_INFO_INVALID (*C macro*), 1665
ESP_ERR_OTA_SMALL_SEC_VER (*C macro*), 1665
ESP_ERR_OTA_VALIDATE_FAILED (*C macro*), 1665
esp_err_t (*C++ type*), 1408
ESP_ERR_TIMEOUT (*C macro*), 1407
esp_err_to_name (*C++ function*), 1406
esp_err_to_name_r (*C++ function*), 1406
ESP_ERR_WIFI_BASE (*C macro*), 1407
ESP_ERR_WOLFSSL_CERT_VERIFY_SETUP_FAILED (*C macro*), 71
ESP_ERR_WOLFSSL_CTX_SETUP_FAILED (*C macro*), 71
ESP_ERR_WOLFSSL_KEY_VERIFY_SETUP_FAILED (*C macro*), 71
ESP_ERR_WOLFSSL_SSL_CONF_ALPN_PROTOCOLS_FAILED (*C macro*), 70
ESP_ERR_WOLFSSL_SSL_HANDSHAKE_FAILED (*C macro*), 71
ESP_ERR_WOLFSSL_SSL_SET_HOSTNAME_FAILED (*C macro*), 70
ESP_ERR_WOLFSSL_SSL_SETUP_FAILED (*C macro*), 71
ESP_ERR_WOLFSSL_SSL_WRITE_FAILED (*C macro*), 71
ESP_ERROR_CHECK (*C macro*), 1408
ESP_ERROR_CHECK_WITHOUT_ABORT (*C macro*), 1408
esp_eth_config_t (*C++ struct*), 178
esp_eth_config_t::check_link_period_ms (*C++ member*), 178
esp_eth_config_t::mac (*C++ member*), 178
esp_eth_config_t::on_lowlevel_deinit_done (*C++ member*), 178
esp_eth_config_t::on_lowlevel_init_done (*C++ member*), 178
esp_eth_config_t::phy (*C++ member*), 178
esp_eth_config_t::read_phy_reg (*C++ member*), 178
esp_eth_config_t::stack_input (*C++ member*), 178
esp_eth_config_t::write_phy_reg (*C++ member*), 179
esp_eth_decrease_reference (*C++ function*), 177
esp_eth_del_netif_glue (*C++ function*), 206
esp_eth_driver_install (*C++ function*), 174
esp_eth_driver_uninstall (*C++ function*), 174
esp_eth_get_mac_instance (*C++ function*), 177
esp_eth_get_phy_instance (*C++ function*), 177
esp_eth_handle_t (*C++ type*), 179
esp_eth_increase_reference (*C++ function*), 177
esp_eth_io_cmd_t (*C++ enum*), 179
esp_eth_io_cmd_t::ETH_CMD_CUSTOM_MAC_CMDS (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_CUSTOM_PHY_CMDS (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_G_AUTONEGO (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_G_DUPLEX_MODE (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_G_MAC_ADDR (*C++ enumerator*), 179
esp_eth_io_cmd_t::ETH_CMD_G_PHY_ADDR (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_G_SPEED (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_READ_PHY_REG (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_S_AUTONEGO (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_S_DUPLEX_MODE (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_S_FLOW_CTRL (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_S_MAC_ADDR (*C++ enumerator*), 180
esp_eth_io_cmd_t::ETH_CMD_S_PHY_ADDR

(C++ enumerator), 180
esp_eth_io_cmd_t::ETH_CMD_S_PHY_LOOPBACK (C++ enumerator), 180
esp_eth_io_cmd_t::ETH_CMD_S_PROMISCUOUS (C++ enumerator), 180
esp_eth_io_cmd_t::ETH_CMD_S_SPEED (C++ enumerator), 180
esp_eth_io_cmd_t::ETH_CMD_WRITE_PHY_REG (C++ enumerator), 180
esp_eth_ioctl (C++ function), 176
esp_eth_mac_new_esp32 (C++ function), 188
esp_eth_mac_s (C++ struct), 183
esp_eth_mac_s::custom_ioctl (C++ member), 187
esp_eth_mac_s::deinit (C++ member), 183
esp_eth_mac_s::del (C++ member), 187
esp_eth_mac_s::enable_flow_ctrl (C++ member), 186
esp_eth_mac_s::get_addr (C++ member), 186
esp_eth_mac_s::init (C++ member), 183
esp_eth_mac_s::read_phy_reg (C++ member), 185
esp_eth_mac_s::receive (C++ member), 185
esp_eth_mac_s::set_addr (C++ member), 185
esp_eth_mac_s::set_duplex (C++ member), 186
esp_eth_mac_s::set_link (C++ member), 186
esp_eth_mac_s::set_mediator (C++ member), 183
esp_eth_mac_s::set_peer_pause_ability (C++ member), 187
esp_eth_mac_s::set_promiscuous (C++ member), 186
esp_eth_mac_s::set_speed (C++ member), 186
esp_eth_mac_s::start (C++ member), 184
esp_eth_mac_s::stop (C++ member), 184
esp_eth_mac_s::transmit (C++ member), 184
esp_eth_mac_s::transmit_vargs (C++ member), 184
esp_eth_mac_s::write_phy_reg (C++ member), 185
esp_eth_mac_t (C++ type), 188
esp_eth_mediator_s (C++ struct), 181
esp_eth_mediator_s::on_state_changed (C++ member), 181
esp_eth_mediator_s::phy_reg_read (C++ member), 181
esp_eth_mediator_s::phy_reg_write (C++ member), 181
esp_eth_mediator_s::stack_input (C++ member), 181
esp_eth_mediator_t (C++ type), 182
esp_eth_netif_glue_handle_t (C++ type), 206
esp_eth_new_netif_glue (C++ function), 206
esp_eth_phy_802_3_advertise_pause_ability (C++ function), 201
esp_eth_phy_802_3_autonego_ctrl (C++ function), 200
esp_eth_phy_802_3_basic_phy_deinit (C++ function), 202
esp_eth_phy_802_3_basic_phy_init (C++ function), 202
esp_eth_phy_802_3_deinit (C++ function), 202
esp_eth_phy_802_3_del (C++ function), 202
esp_eth_phy_802_3_detect_phy_addr (C++ function), 202
esp_eth_phy_802_3_get_addr (C++ function), 200
esp_eth_phy_802_3_get_mmd_addr (C++ function), 203
esp_eth_phy_802_3_init (C++ function), 201
esp_eth_phy_802_3_loopback (C++ function), 201
esp_eth_phy_802_3_mmd_func_t (C++ enum), 206
esp_eth_phy_802_3_mmd_func_t::MMD_FUNC_ADDRESS (C++ enumerator), 206
esp_eth_phy_802_3_mmd_func_t::MMD_FUNC_DATA_NOINC (C++ enumerator), 206
esp_eth_phy_802_3_mmd_func_t::MMD_FUNC_DATA_RWINC (C++ enumerator), 206
esp_eth_phy_802_3_mmd_func_t::MMD_FUNC_DATA_WINCR (C++ enumerator), 206
esp_eth_phy_802_3_obj_config_init (C++ function), 205
esp_eth_phy_802_3_pwrctl (C++ function), 200
esp_eth_phy_802_3_read_manufac_info (C++ function), 203
esp_eth_phy_802_3_read_mmd_data (C++ function), 204
esp_eth_phy_802_3_read_mmd_register (C++ function), 204
esp_eth_phy_802_3_read_oui (C++ function), 203
esp_eth_phy_802_3_reset (C++ function), 200
esp_eth_phy_802_3_reset_hw (C++ function), 202
esp_eth_phy_802_3_set_addr (C++ function), 200
esp_eth_phy_802_3_set_duplex (C++ function), 201
esp_eth_phy_802_3_set_link (C++ function), 201
esp_eth_phy_802_3_set_mediator (C++ function), 200
esp_eth_phy_802_3_set_mmd_addr (C++ function), 203
esp_eth_phy_802_3_set_speed (C++ function), 201
esp_eth_phy_802_3_write_mmd_data (C++ function), 204
esp_eth_phy_802_3_write_mmd_register

(C++ function), 204
ESP_ETH_PHY_ADDR_AUTO (C macro), 199
esp_eth_phy_into_phy_802_3 (C++ function), 205
esp_eth_phy_new_dp83848 (C++ function), 195
esp_eth_phy_new_ip101 (C++ function), 195
esp_eth_phy_new_ksz80xx (C++ function), 195
esp_eth_phy_new_lan87xx (C++ function), 195
esp_eth_phy_new_rtl8201 (C++ function), 195
esp_eth_phy_reg_rw_data_t (C++ struct), 179
esp_eth_phy_reg_rw_data_t::reg_addr (C++ member), 179
esp_eth_phy_reg_rw_data_t::reg_value_p (C++ member), 179
esp_eth_phy_s (C++ struct), 195
esp_eth_phy_s::advertise_pause_ability (C++ member), 197
esp_eth_phy_s::autonego_ctrl (C++ member), 196
esp_eth_phy_s::custom_ioctl (C++ member), 198
esp_eth_phy_s::deinit (C++ member), 196
esp_eth_phy_s::del (C++ member), 198
esp_eth_phy_s::get_addr (C++ member), 197
esp_eth_phy_s::get_link (C++ member), 196
esp_eth_phy_s::init (C++ member), 196
esp_eth_phy_s::loopback (C++ member), 197
esp_eth_phy_s::pwrctl (C++ member), 197
esp_eth_phy_s::reset (C++ member), 196
esp_eth_phy_s::reset_hw (C++ member), 196
esp_eth_phy_s::set_addr (C++ member), 197
esp_eth_phy_s::set_duplex (C++ member), 198
esp_eth_phy_s::set_link (C++ member), 197
esp_eth_phy_s::set_mediator (C++ member), 195
esp_eth_phy_s::set_speed (C++ member), 197
esp_eth_phy_t (C++ type), 199
esp_eth_start (C++ function), 175
esp_eth_state_t (C++ enum), 182
esp_eth_state_t::ETH_STATE_DEINIT (C++ enumerator), 182
esp_eth_state_t::ETH_STATE_DUPLEX (C++ enumerator), 182
esp_eth_state_t::ETH_STATE_LINK (C++ enumerator), 182
esp_eth_state_t::ETH_STATE_LLINIT (C++ enumerator), 182
esp_eth_state_t::ETH_STATE_PAUSE (C++ enumerator), 182
esp_eth_state_t::ETH_STATE_SPEED (C++ enumerator), 182
esp_eth_stop (C++ function), 175
esp_eth_transmit (C++ function), 175
esp_eth_transmit_vargs (C++ function), 176
esp_eth_update_input_path (C++ function), 175
esp_etm_channel_config_t (C++ struct), 311
esp_etm_channel_connect (C++ function), 310
esp_etm_channel_disable (C++ function), 309
esp_etm_channel_enable (C++ function), 309
esp_etm_channel_handle_t (C++ type), 311
esp_etm_del_channel (C++ function), 309
esp_etm_del_event (C++ function), 310
esp_etm_del_task (C++ function), 310
esp_etm_dump (C++ function), 311
esp_etm_event_handle_t (C++ type), 311
esp_etm_new_channel (C++ function), 309
esp_etm_task_handle_t (C++ type), 311
ESP_EVENT_ANY_BASE (C macro), 1428
ESP_EVENT_ANY_ID (C macro), 1428
ESP_EVENT_DECLARE_BASE (C macro), 1428
ESP_EVENT_DEFINE_BASE (C macro), 1428
esp_event_dump (C++ function), 1426
esp_event_handler_instance_register (C++ function), 1422
esp_event_handler_instance_register_with (C++ function), 1421
esp_event_handler_instance_t (C++ type), 1428
esp_event_handler_instance_unregister (C++ function), 1424
esp_event_handler_instance_unregister_with (C++ function), 1423
esp_event_handler_register (C++ function), 1420
esp_event_handler_register_with (C++ function), 1421
esp_event_handler_t (C++ type), 1428
esp_event_handler_unregister (C++ function), 1423
esp_event_handler_unregister_with (C++ function), 1423
esp_event_isr_post (C++ function), 1425
esp_event_isr_post_to (C++ function), 1425
esp_event_loop_args_t (C++ struct), 1427
esp_event_loop_args_t::queue_size (C++ member), 1427
esp_event_loop_args_t::task_core_id (C++ member), 1427
esp_event_loop_args_t::task_name (C++ member), 1427
esp_event_loop_args_t::task_priority (C++ member), 1427
esp_event_loop_args_t::task_stack_size (C++ member), 1427
esp_event_loop_create (C++ function), 1419
esp_event_loop_create_default (C++ function), 1419
esp_event_loop_delete (C++ function), 1419
esp_event_loop_delete_default (C++ function), 1420
esp_event_loop_handle_t (C++ type), 1428
esp_event_loop_run (C++ function), 1420
esp_event_post (C++ function), 1424

esp_event_post_to (C++ function), 1425
ESP_EXECUTE_EXPRESSION_WITH_STACK (C macro), 1360
esp_execute_shared_stack_function (C++ function), 1359
ESP_FAIL (C macro), 1407
esp_fill_random (C++ function), 1679
esp_flash_chip_driver_initialized (C++ function), 674
esp_flash_counter_t (C++ struct), 691
esp_flash_counter_t::bytes (C++ member), 691
esp_flash_counter_t::count (C++ member), 691
esp_flash_counter_t::time (C++ member), 691
esp_flash_counters_t (C++ struct), 691
esp_flash_counters_t::erase (C++ member), 691
esp_flash_counters_t::read (C++ member), 691
esp_flash_counters_t::write (C++ member), 691
esp_flash_dump_counters (C++ function), 691
esp_flash_enc_mode_t (C++ enum), 694
esp_flash_enc_mode_t::ESP_FLASH_ENC_MODE_DEVELOPMENT (C++ enumerator), 694
esp_flash_enc_mode_t::ESP_FLASH_ENC_MODE_DISABLED (C++ enumerator), 694
esp_flash_enc_mode_t::ESP_FLASH_ENC_MODE_RELEASE (C++ enumerator), 694
esp_flash_encrypt_check_and_update (C++ function), 692
esp_flash_encrypt_contents (C++ function), 692
esp_flash_encrypt_enable (C++ function), 692
esp_flash_encrypt_init (C++ function), 692
esp_flash_encrypt_initialized_once (C++ function), 692
esp_flash_encrypt_is_write_protected (C++ function), 692
esp_flash_encrypt_region (C++ function), 693
esp_flash_encrypt_state (C++ function), 692
esp_flash_encryption_cfg_verify_release_mode (C++ function), 693
esp_flash_encryption_enabled (C++ function), 692
esp_flash_encryption_init_checks (C++ function), 693
esp_flash_encryption_set_release_mode (C++ function), 693
esp_flash_erase_chip (C++ function), 675
esp_flash_erase_region (C++ function), 675
esp_flash_get_chip_write_protect (C++ function), 675
esp_flash_get_counters (C++ function), 691
esp_flash_get_physical_size (C++ function), 674
esp_flash_get_protectable_regions (C++ function), 676
esp_flash_get_protected_region (C++ function), 676
esp_flash_get_size (C++ function), 674
esp_flash_init (C++ function), 674
esp_flash_io_mode_t (C++ enum), 689
esp_flash_io_mode_t::SPI_FLASH_DIO (C++ enumerator), 689
esp_flash_io_mode_t::SPI_FLASH_DOUT (C++ enumerator), 689
esp_flash_io_mode_t::SPI_FLASH_FASTRD (C++ enumerator), 689
esp_flash_io_mode_t::SPI_FLASH_OPI_DTR (C++ enumerator), 690
esp_flash_io_mode_t::SPI_FLASH_OPI_STR (C++ enumerator), 689
esp_flash_io_mode_t::SPI_FLASH_QIO (C++ enumerator), 689
esp_flash_io_mode_t::SPI_FLASH_QOUT (C++ enumerator), 689
esp_flash_io_mode_t::SPI_FLASH_READ_MODE_MAX (C++ enumerator), 690
esp_flash_io_mode_t::SPI_FLASH_SLOWRD (C++ enumerator), 689
esp_flash_is_quad_mode (C++ function), 679
esp_flash_os_functions_t (C++ struct), 679
esp_flash_os_functions_t::check_yield (C++ member), 679
esp_flash_os_functions_t::delay_us (C++ member), 679
esp_flash_os_functions_t::end (C++ member), 679
esp_flash_os_functions_t::get_system_time (C++ member), 680
esp_flash_os_functions_t::get_temp_buffer (C++ member), 679
esp_flash_os_functions_t::region_protected (C++ member), 679
esp_flash_os_functions_t::release_temp_buffer (C++ member), 679
esp_flash_os_functions_t::set_flash_op_status (C++ member), 680
esp_flash_os_functions_t::start (C++ member), 679
esp_flash_os_functions_t::yield (C++ member), 680
esp_flash_read (C++ function), 677
esp_flash_read_encrypted (C++ function), 678
esp_flash_read_id (C++ function), 674
esp_flash_read_unique_chip_id (C++ function), 675
esp_flash_region_t (C++ struct), 679
esp_flash_region_t::offset (C++ member), 679

esp_flash_region_t::size (C++ member), 679
esp_flash_reset_counters (C++ function), 691
esp_flash_set_chip_write_protect (C++ function), 676
esp_flash_set_protected_region (C++ function), 677
esp_flash_speed_s (C++ enum), 688
esp_flash_speed_s::ESP_FLASH_10MHZ (C++ enumerator), 689
esp_flash_speed_s::ESP_FLASH_120MHZ (C++ enumerator), 689
esp_flash_speed_s::ESP_FLASH_20MHZ (C++ enumerator), 689
esp_flash_speed_s::ESP_FLASH_26MHZ (C++ enumerator), 689
esp_flash_speed_s::ESP_FLASH_40MHZ (C++ enumerator), 689
esp_flash_speed_s::ESP_FLASH_5MHZ (C++ enumerator), 689
esp_flash_speed_s::ESP_FLASH_80MHZ (C++ enumerator), 689
esp_flash_speed_s::ESP_FLASH_SPEED_MAX (C++ enumerator), 689
esp_flash_speed_t (C++ type), 688
esp_flash_spi_device_config_t (C++ struct), 673
esp_flash_spi_device_config_t::cs_id (C++ member), 673
esp_flash_spi_device_config_t::cs_io_num (C++ member), 673
esp_flash_spi_device_config_t::freq_mhz (C++ member), 673
esp_flash_spi_device_config_t::host_id (C++ member), 673
esp_flash_spi_device_config_t::input_delay_ns (C++ member), 673
esp_flash_spi_device_config_t::io_mode (C++ member), 673
esp_flash_spi_device_config_t::speed (C++ member), 673
esp_flash_t (C++ struct), 680
esp_flash_t::busy (C++ member), 680
esp_flash_t::chip_drv (C++ member), 680
esp_flash_t::chip_id (C++ member), 680
esp_flash_t::host (C++ member), 680
esp_flash_t::hpm_dummy_ena (C++ member), 681
esp_flash_t::os_func (C++ member), 680
esp_flash_t::os_func_data (C++ member), 680
esp_flash_t::read_mode (C++ member), 680
esp_flash_t::reserved_flags (C++ member), 681
esp_flash_t::size (C++ member), 680
esp_flash_write (C++ function), 678
esp_flash_write_encrypted (C++ function), 678
esp_flash_write_protect_crypt_cnt (C++ function), 693
esp_freertos_idle_cb_t (C++ type), 1566
esp_freertos_tick_cb_t (C++ type), 1566
esp_gcov_dump (C++ function), 1356
esp_get_deep_sleep_wake_stub (C++ function), 1693
esp_get_flash_encryption_mode (C++ function), 693
esp_get_free_heap_size (C++ function), 1640
esp_get_free_internal_heap_size (C++ function), 1640
esp_get_idf_version (C++ function), 1641
esp_get_minimum_free_heap_size (C++ function), 1640
ESP_GOTO_ON_ERROR (C macro), 1405
ESP_GOTO_ON_ERROR_ISR (C macro), 1406
ESP_GOTO_ON_FALSE (C macro), 1406
ESP_GOTO_ON_FALSE_ISR (C macro), 1406
esp_hmac_calculate (C++ function), 362
esp_hmac_jtag_disable (C++ function), 362
esp_hmac_jtag_enable (C++ function), 362
esp_http_client_add_auth (C++ function), 80
esp_http_client_auth_type_t (C++ enum), 88
esp_http_client_auth_type_t::HTTP_AUTH_TYPE_BASIC (C++ enumerator), 89
esp_http_client_auth_type_t::HTTP_AUTH_TYPE_DIGEST (C++ enumerator), 89
esp_http_client_auth_type_t::HTTP_AUTH_TYPE_NONE (C++ enumerator), 88
esp_http_client_cancel_request (C++ function), 75
esp_http_client_cleanup (C++ function), 80
esp_http_client_close (C++ function), 79
esp_http_client_config_t (C++ struct), 83
esp_http_client_config_t::auth_type (C++ member), 83
esp_http_client_config_t::buffer_size (C++ member), 84
esp_http_client_config_t::buffer_size_tx (C++ member), 84
esp_http_client_config_t::cert_len (C++ member), 83
esp_http_client_config_t::cert_pem (C++ member), 83
esp_http_client_config_t::client_cert_len (C++ member), 83
esp_http_client_config_t::client_cert_pem (C++ member), 83
esp_http_client_config_t::client_key_len (C++ member), 83
esp_http_client_config_t::client_key_password (C++ member), 84
esp_http_client_config_t::client_key_password_len (C++ member), 84
esp_http_client_config_t::client_key_pem

(C++ member), 83
esp_http_client_config_t::common_name (C++ member), 85
esp_http_client_config_t::crt_bundle_attach (C++ member), 85
esp_http_client_config_t::disable_auto_redirect (C++ member), 84
esp_http_client_config_t::ds_data (C++ member), 85
esp_http_client_config_t::event_handler (C++ member), 84
esp_http_client_config_t::host (C++ member), 83
esp_http_client_config_t::if_name (C++ member), 85
esp_http_client_config_t::is_async (C++ member), 84
esp_http_client_config_t::keep_alive_count (C++ member), 85
esp_http_client_config_t::keep_alive_enable (C++ member), 85
esp_http_client_config_t::keep_alive_idle (C++ member), 85
esp_http_client_config_t::keep_alive_interval (C++ member), 85
esp_http_client_config_t::max_authorization_retries (C++ member), 84
esp_http_client_config_t::max_redirection_count (C++ member), 84
esp_http_client_config_t::method (C++ member), 84
esp_http_client_config_t::password (C++ member), 83
esp_http_client_config_t::path (C++ member), 83
esp_http_client_config_t::port (C++ member), 83
esp_http_client_config_t::query (C++ member), 83
esp_http_client_config_t::skip_cert_common_name_check (C++ member), 85
esp_http_client_config_t::timeout_ms (C++ member), 84
esp_http_client_config_t::tls_version (C++ member), 84
esp_http_client_config_t::transport_type (C++ member), 84
esp_http_client_config_t::url (C++ member), 83
esp_http_client_config_t::use_global_ca_store (C++ member), 84
esp_http_client_config_t::user_agent (C++ member), 84
esp_http_client_config_t::user_data (C++ member), 84
esp_http_client_config_t::username (C++ member), 83
esp_http_client_delete_header (C++ function), 78
esp_http_client_event (C++ struct), 82
esp_http_client_event::client (C++ member), 82
esp_http_client_event::data (C++ member), 82
esp_http_client_event::data_len (C++ member), 82
esp_http_client_event::event_id (C++ member), 82
esp_http_client_event::header_key (C++ member), 82
esp_http_client_event::header_value (C++ member), 82
esp_http_client_event::user_data (C++ member), 82
esp_http_client_event_handle_t (C++ type), 86
esp_http_client_event_id_t (C++ enum), 86
esp_http_client_event_id_t::HTTP_EVENT_DISCONNECTED (C++ enumerator), 87
esp_http_client_event_id_t::HTTP_EVENT_ERROR (C++ enumerator), 86
esp_http_client_event_id_t::HTTP_EVENT_HEADER_SENT (C++ enumerator), 86
esp_http_client_event_id_t::HTTP_EVENT_HEADERS_SENT (C++ enumerator), 86
esp_http_client_event_id_t::HTTP_EVENT_ON_CONNECTED (C++ enumerator), 86
esp_http_client_event_id_t::HTTP_EVENT_ON_DATA (C++ enumerator), 86
esp_http_client_event_id_t::HTTP_EVENT_ON_FINISH (C++ enumerator), 87
esp_http_client_event_id_t::HTTP_EVENT_ON_HEADER (C++ enumerator), 86
esp_http_client_event_id_t::HTTP_EVENT_REDIRECT (C++ enumerator), 87
esp_http_client_event_t (C++ type), 86
esp_http_client_fetch_headers (C++ function), 79
esp_http_client_flush_response (C++ function), 81
esp_http_client_get_chunk_length (C++ function), 81
esp_http_client_get_content_length (C++ function), 79
esp_http_client_get_errno (C++ function), 78
esp_http_client_get_header (C++ function), 76
esp_http_client_get_password (C++ function), 77
esp_http_client_get_post_field (C++ function), 76
esp_http_client_get_status_code (C++ function), 79
esp_http_client_get_transport_type (C++ function), 80

esp_http_client_get_url (*C++ function*), 81
esp_http_client_get_user_data (*C++ function*), 77
esp_http_client_get_username (*C++ function*), 76
esp_http_client_handle_t (*C++ type*), 86
esp_http_client_init (*C++ function*), 75
esp_http_client_is_chunked_response (*C++ function*), 79
esp_http_client_is_complete_data_received (*C++ function*), 81
esp_http_client_method_t (*C++ enum*), 87
esp_http_client_method_t::HTTP_METHOD_COPY (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_DELETE (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_GET (*C++ enumerator*), 87
esp_http_client_method_t::HTTP_METHOD_HEAD (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_LOCK (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_MAX (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_MKCOL (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_MOVE (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_NOTIFY (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_OPTIONS (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_PATCH (*C++ enumerator*), 87
esp_http_client_method_t::HTTP_METHOD_POST (*C++ enumerator*), 87
esp_http_client_method_t::HTTP_METHOD_PROPFIND (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_PROPPATCH (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_PUT (*C++ enumerator*), 87
esp_http_client_method_t::HTTP_METHOD_SUBSCRIBE (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_UNLOCK (*C++ enumerator*), 88
esp_http_client_method_t::HTTP_METHOD_UNSUBSCRIBE (*C++ enumerator*), 88
esp_http_client_on_data (*C++ struct*), 82
esp_http_client_on_data::client (*C++ member*), 82
esp_http_client_on_data::data_process (*C++ member*), 82
esp_http_client_on_data_t (*C++ type*), 86
esp_http_client_open (*C++ function*), 78
esp_http_client_perform (*C++ function*), 75
esp_http_client_proto_ver_t (*C++ enum*), 87
esp_http_client_proto_ver_t::ESP_HTTP_CLIENT_TLS_ (*C++ enumerator*), 87
esp_http_client_proto_ver_t::ESP_HTTP_CLIENT_TLS_ (*C++ enumerator*), 87
esp_http_client_proto_ver_t::ESP_HTTP_CLIENT_TLS_ (*C++ enumerator*), 87
esp_http_client_proto_ver_t::ESP_HTTP_CLIENT_TLS_ (*C++ enumerator*), 87
esp_http_client_read (*C++ function*), 79
esp_http_client_read_response (*C++ function*), 81
esp_http_client_redirect_event_data (*C++ struct*), 82
esp_http_client_redirect_event_data::client (*C++ member*), 82
esp_http_client_redirect_event_data::status_code (*C++ member*), 82
esp_http_client_redirect_event_data_t (*C++ type*), 86
esp_http_client_reset_redirect_counter (*C++ function*), 80
esp_http_client_set_auth_data (*C++ function*), 80
esp_http_client_set_authtype (*C++ function*), 77
esp_http_client_set_header (*C++ function*), 76
esp_http_client_set_method (*C++ function*), 78
esp_http_client_set_password (*C++ function*), 77
esp_http_client_set_post_field (*C++ function*), 76
esp_http_client_set_redirection (*C++ function*), 80
esp_http_client_set_timeout_ms (*C++ function*), 78
esp_http_client_set_url (*C++ function*), 76
esp_http_client_set_user_data (*C++ function*), 78
esp_http_client_set_username (*C++ function*), 77
esp_http_client_transport_t (*C++ enum*), 87
esp_http_client_transport_t::HTTP_TRANSPORT_OVER_ (*C++ enumerator*), 87
esp_http_client_transport_t::HTTP_TRANSPORT_OVER_ (*C++ enumerator*), 87
esp_http_client_transport_t::HTTP_TRANSPORT_UNKNO (*C++ enumerator*), 87
esp_http_client_write (*C++ function*), 79
esp_http_server_event_data (*C++ struct*), 134
esp_http_server_event_data::data_len (*C++ member*), 134
esp_http_server_event_data::fd (*C++ member*), 134
esp_http_server_event_id_t (*C++ enum*),

143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_DISCONNECTED (C++ enumerator), 143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_ERROR (C++ enumerator), 143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_HEADERS_SENT (C++ enumerator), 143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_ON_CONNECTED (C++ enumerator), 143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_ON_DATA (C++ enumerator), 143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_ON_HEADER (C++ enumerator), 143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_SENT_DATA (C++ enumerator), 143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_START (C++ enumerator), 143
esp_http_server_event_id_t::HTTP_SERVER_EVENT_STOP (C++ enumerator), 143
ESP_HTTPD_DEF_CTRL_PORT (C macro), 139
esp_https_ota (C++ function), 1411
esp_https_ota_abort (C++ function), 1412
esp_https_ota_begin (C++ function), 1411
esp_https_ota_config_t (C++ struct), 1414
esp_https_ota_config_t::buffer_caps (C++ member), 1414
esp_https_ota_config_t::bulk_flash_erase (C++ member), 1414
esp_https_ota_config_t::http_client_init_cb (C++ member), 1414
esp_https_ota_config_t::http_config (C++ member), 1414
esp_https_ota_config_t::max_http_request_size (C++ member), 1414
esp_https_ota_config_t::partial_http_download (C++ member), 1414
esp_https_ota_event_t (C++ enum), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_ABORT (C++ enumerator), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_CONNECTED (C++ enumerator), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_DECRYPT_CB (C++ enumerator), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_FINISH (C++ enumerator), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_GET_IMG_DESC (C++ enumerator), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_START (C++ enumerator), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_UPDATE_BOOT_PARTITION (C++ enumerator), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_VERIFY_CHIP_ID (C++ enumerator), 1415
esp_https_ota_event_t::ESP_HTTPS_OTA_WRITE_FLASH (C++ enumerator), 1415
esp_https_ota_finish (C++ function), 1412
esp_https_ota_get_image_len_read (C++ function), 1413
esp_https_ota_get_image_size (C++ function), 1413
esp_https_ota_get_img_desc (C++ function), 1413
esp_https_ota_get_status_code (C++ function), 1413
esp_https_ota_handle_t (C++ type), 1414
esp_https_ota_is_complete_data_received (C++ function), 1412
esp_https_ota_perform (C++ function), 1412
esp_https_server_event_id_t (C++ enum), 147
esp_https_server_event_id_t::HTTPS_SERVER_EVENT_DISCONNECTED (C++ enumerator), 147
esp_https_server_event_id_t::HTTPS_SERVER_EVENT_ERROR (C++ enumerator), 147
esp_https_server_event_id_t::HTTPS_SERVER_EVENT_ON_CONNECTED (C++ enumerator), 147
esp_https_server_event_id_t::HTTPS_SERVER_EVENT_ON_DATA (C++ enumerator), 147
esp_https_server_event_id_t::HTTPS_SERVER_EVENT_SENT_DATA (C++ enumerator), 147
esp_https_server_event_id_t::HTTPS_SERVER_EVENT_START (C++ enumerator), 147
esp_https_server_event_id_t::HTTPS_SERVER_EVENT_STOP (C++ enumerator), 147
esp_https_server_last_error_t (C++ type), 147
esp_https_server_user_cb (C++ type), 147
esp_https_server_user_cb_arg (C++ struct), 145
esp_https_server_user_cb_arg::tls (C++ member), 145
esp_https_server_user_cb_arg::user_cb_state (C++ member), 145
esp_https_server_user_cb_arg_t (C++ type), 147
ESP_IDF_VERSION (C macro), 1642
ESP_IDF_VERSION_MAJOR (C macro), 1642
ESP_IDF_VERSION_MINOR (C macro), 1642
ESP_IDF_VERSION_PATCH (C macro), 1642
ESP_IDF_VERSION_VAL (C macro), 1642
esp_iface_mac_addr_set (C++ function), 1643
esp_image_flash_size_t (C++ enum), 1350
esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_128MB (C++ enumerator), 1350
esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_16MB (C++ enumerator), 1350
esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_1MB (C++ enumerator), 1350
esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_2MB (C++ enumerator), 1350
esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_32MB (C++ enumerator), 1350
esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_4MB (C++ enumerator), 1350
esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_64MB (C++ enumerator), 1350

esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_8MB (C++ enumerator), 1350
esp_image_flash_size_t::ESP_IMAGE_FLASH_SIZE_MAX (C++ enumerator), 1350
ESP_IMAGE_HEADER_MAGIC (C macro), 1348
esp_image_header_t (C++ struct), 1347
esp_image_header_t::chip_id (C++ member), 1347
esp_image_header_t::entry_addr (C++ member), 1347
esp_image_header_t::hash_appended (C++ member), 1348
esp_image_header_t::magic (C++ member), 1347
esp_image_header_t::max_chip_rev_full (C++ member), 1348
esp_image_header_t::min_chip_rev (C++ member), 1347
esp_image_header_t::min_chip_rev_full (C++ member), 1347
esp_image_header_t::reserved (C++ member), 1348
esp_image_header_t::segment_count (C++ member), 1347
esp_image_header_t::spi_mode (C++ member), 1347
esp_image_header_t::spi_pin_drv (C++ member), 1347
esp_image_header_t::spi_size (C++ member), 1347
esp_image_header_t::spi_speed (C++ member), 1347
esp_image_header_t::wp_pin (C++ member), 1347
ESP_IMAGE_MAX_SEGMENTS (C macro), 1348
esp_image_segment_header_t (C++ struct), 1348
esp_image_segment_header_t::data_len (C++ member), 1348
esp_image_segment_header_t::load_addr (C++ member), 1348
esp_image_spi_freq_t (C++ enum), 1349
esp_image_spi_freq_t::ESP_IMAGE_SPI_SPEED_DIV_1 (C++ enumerator), 1350
esp_image_spi_freq_t::ESP_IMAGE_SPI_SPEED_DIV_2 (C++ enumerator), 1349
esp_image_spi_freq_t::ESP_IMAGE_SPI_SPEED_DIV_3 (C++ enumerator), 1349
esp_image_spi_freq_t::ESP_IMAGE_SPI_SPEED_DIV_4 (C++ enumerator), 1349
esp_image_spi_mode_t (C++ enum), 1349
esp_image_spi_mode_t::ESP_IMAGE_SPI_MODE_DIO (C++ enumerator), 1349
esp_image_spi_mode_t::ESP_IMAGE_SPI_MODE_DOUT (C++ enumerator), 1349
esp_image_spi_mode_t::ESP_IMAGE_SPI_MODE_FAST_READ (C++ enumerator), 1349
esp_image_spi_mode_t::ESP_IMAGE_SPI_MODE_QIO (C++ enumerator), 1349
esp_image_spi_mode_t::ESP_IMAGE_SPI_MODE_QOUT (C++ enumerator), 1349
esp_image_spi_mode_t::ESP_IMAGE_SPI_MODE_SLOW_READ (C++ enumerator), 1349
esp_intr_alloc (C++ function), 1624
esp_intr_alloc_intrstatus (C++ function), 1624
esp_intr_cpu_affinity_t (C++ enum), 1623
esp_intr_cpu_affinity_t::ESP_INTR_CPU_AFFINITY_0 (C++ enumerator), 1623
esp_intr_cpu_affinity_t::ESP_INTR_CPU_AFFINITY_1 (C++ enumerator), 1623
esp_intr_cpu_affinity_t::ESP_INTR_CPU_AFFINITY_AUTO (C++ enumerator), 1623
ESP_INTR_CPU_AFFINITY_TO_CORE_ID (C macro), 1623
ESP_INTR_DISABLE (C macro), 1628
esp_intr_disable (C++ function), 1625
esp_intr_disable_source (C++ function), 1626
esp_intr_dump (C++ function), 1626
ESP_INTR_ENABLE (C macro), 1628
esp_intr_enable (C++ function), 1626
esp_intr_enable_source (C++ function), 1626
ESP_INTR_FLAG_EDGE (C macro), 1627
ESP_INTR_FLAG_HIGH (C macro), 1627
ESP_INTR_FLAG_INTRDISABLED (C macro), 1627
ESP_INTR_FLAG_IRAM (C macro), 1627
ESP_INTR_FLAG_LEVEL1 (C macro), 1627
ESP_INTR_FLAG_LEVEL2 (C macro), 1627
ESP_INTR_FLAG_LEVEL3 (C macro), 1627
ESP_INTR_FLAG_LEVEL4 (C macro), 1627
ESP_INTR_FLAG_LEVEL5 (C macro), 1627
ESP_INTR_FLAG_LEVEL6 (C macro), 1627
ESP_INTR_FLAG_LEVELMASK (C macro), 1627
ESP_INTR_FLAG_LOWMED (C macro), 1627
ESP_INTR_FLAG_NMI (C macro), 1627
ESP_INTR_FLAG_SHARED (C macro), 1627
esp_intr_flags_to_level (C++ function), 1626
esp_intr_free (C++ function), 1625
esp_intr_get_cpu (C++ function), 1625
esp_intr_get_intno (C++ function), 1625
esp_intr_level_to_flags (C++ function), 1626
esp_intr_mark_shared (C++ function), 1624
esp_intr_noniram_disable (C++ function), 1626
esp_intr_noniram_enable (C++ function), 1626
esp_intr_ptr_in_isr_region (C++ function), 1626
esp_intr_reserve (C++ function), 1624
esp_intr_set_in_iram (C++ function), 1626
esp_ip4_addr (C++ struct), 249
esp_ip4_addr1 (C macro), 249
esp_ip4_addr1_16 (C macro), 250

esp_ip4_addr2 (*C macro*), 250
esp_ip4_addr2_16 (*C macro*), 250
esp_ip4_addr3 (*C macro*), 250
esp_ip4_addr3_16 (*C macro*), 250
esp_ip4_addr4 (*C macro*), 250
esp_ip4_addr4_16 (*C macro*), 250
esp_ip4_addr::addr (*C++ member*), 249
esp_ip4_addr_get_byte (*C macro*), 249
esp_ip4_addr_t (*C++ type*), 250
esp_ip4addr_aton (*C++ function*), 235
ESP_IP4ADDR_INIT (*C macro*), 250
esp_ip4addr_ntoa (*C++ function*), 235
ESP_IP4TOADDR (*C macro*), 250
ESP_IP4TOUINT32 (*C macro*), 250
esp_ip6_addr (*C++ struct*), 248
esp_ip6_addr::addr (*C++ member*), 248
esp_ip6_addr::zone (*C++ member*), 248
ESP_IP6_ADDR_BLOCK1 (*C macro*), 249
ESP_IP6_ADDR_BLOCK2 (*C macro*), 249
ESP_IP6_ADDR_BLOCK3 (*C macro*), 249
ESP_IP6_ADDR_BLOCK4 (*C macro*), 249
ESP_IP6_ADDR_BLOCK5 (*C macro*), 249
ESP_IP6_ADDR_BLOCK6 (*C macro*), 249
ESP_IP6_ADDR_BLOCK7 (*C macro*), 249
ESP_IP6_ADDR_BLOCK8 (*C macro*), 249
esp_ip6_addr_t (*C++ type*), 250
esp_ip6_addr_type_t (*C++ enum*), 250
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_GLOBAL (*C++ enumerator*), 250
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_IPV4_MAPPED_IPV6 (*C++ enumerator*), 251
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_LINK_LOCAL (*C++ enumerator*), 251
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_SITE_LOCAL (*C++ enumerator*), 251
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_UNIQUE_LOCAL (*C++ enumerator*), 251
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_UNKNOWN (*C++ enumerator*), 250
ESP_IP6ADDR_INIT (*C macro*), 250
esp_ip_addr_t (*C++ type*), 250
ESP_IP_IS_ANY (*C macro*), 250
ESP_IPADDR_TYPE_ANY (*C macro*), 250
ESP_IPADDR_TYPE_V4 (*C macro*), 250
ESP_IPADDR_TYPE_V6 (*C macro*), 250
esp_ipc_call (*C++ function*), 1618
esp_ipc_call_blocking (*C++ function*), 1618
esp_ipc_func_t (*C++ type*), 1619
esp_ipc_isr_asm_call (*C macro*), 1620
esp_ipc_isr_asm_call_blocking (*C macro*), 1620
esp_ipc_isr_call (*C++ function*), 1619
esp_ipc_isr_call_blocking (*C++ function*), 1619
esp_ipc_isr_func_t (*C++ type*), 1620
esp_ipc_isr_release_other_cpu (*C++ function*), 1620
esp_ipc_isr_stall_abort (*C++ function*), 1620
esp_ipc_isr_stall_other_cpu (*C++ function*), 1619
esp_ipc_isr_stall_pause (*C++ function*), 1620
esp_ipc_isr_stall_resume (*C++ function*), 1620
esp_isp_af_config_t (*C++ struct*), 454
esp_isp_af_config_t::edge_thresh (*C++ member*), 454
esp_isp_af_config_t::intr_priority (*C++ member*), 454
esp_isp_af_config_t::window (*C++ member*), 454
esp_isp_af_controller_disable (*C++ function*), 452
esp_isp_af_controller_enable (*C++ function*), 451
esp_isp_af_controller_get_oneshot_statistics (*C++ function*), 452
esp_isp_af_controller_set_env_detector (*C++ function*), 453
esp_isp_af_controller_set_env_detector_threshold (*C++ function*), 453
esp_isp_af_controller_start_continuous_statistics (*C++ function*), 452
esp_isp_af_controller_stop_continuous_statistics (*C++ function*), 452
esp_isp_af_env_config_t (*C++ struct*), 454
esp_isp_af_env_config_t::interval (*C++ member*), 454
esp_isp_af_env_detector_callback_t (*C++ type*), 455
esp_isp_af_env_detector_evt_cbs_t (*C++ struct*), 454
esp_isp_af_env_detector_evt_cbs_t::on_env_change (*C++ member*), 454
esp_isp_af_env_detector_evt_cbs_t::on_env_statist (*C++ member*), 454
esp_isp_af_env_detector_evt_data_t (*C++ struct*), 454
esp_isp_af_env_detector_evt_data_t::af_result (*C++ member*), 454
esp_isp_af_env_detector_register_event_callbacks (*C++ function*), 453
esp_isp_del_af_controller (*C++ function*), 451
esp_isp_new_af_controller (*C++ function*), 451
esp_lcd_color_conv_config_t (*C++ struct*), 477
esp_lcd_color_conv_config_t::conv_std (*C++ member*), 477
esp_lcd_color_conv_config_t::in_color_range (*C++ member*), 477
esp_lcd_color_conv_config_t::in_pack_order (*C++ member*), 478

esp_lcd_color_conv_config_t::out_color_range (C++ type), 473
(C++ member), 477
esp_lcd_color_conv_config_t::spec
(C++ member), 478
esp_lcd_color_conv_config_t::yuv (C++
member), 478
esp_lcd_color_conv_config_t::yuv422
(C++ member), 478
esp_lcd_dbi_io_config_t (C++ struct), 471
esp_lcd_dbi_io_config_t::lcd_cmd_bits
(C++ member), 471
esp_lcd_dbi_io_config_t::lcd_param_bits
(C++ member), 471
esp_lcd_dbi_io_config_t::virtual_channel
(C++ member), 471
esp_lcd_del_dsi_bus (C++ function), 469
esp_lcd_del_i80_bus (C++ function), 464
esp_lcd_dpi_panel_color_trans_done_cb_t
(C++ type), 473
esp_lcd_dpi_panel_config_t (C++ struct),
471
esp_lcd_dpi_panel_config_t::dpi_clk_src
(C++ member), 471
esp_lcd_dpi_panel_config_t::dpi_clock_freq_mhz
(C++ member), 471
esp_lcd_dpi_panel_config_t::extra_dpi_panel_config_t
(C++ struct), 472
esp_lcd_dpi_panel_config_t::extra_dpi_esp_lcd_i80_bus_config_t::dc_gpio_num
(C++ member), 472
esp_lcd_dpi_panel_config_t::extra_dpi_esp_lcd_i80_bus_config_t::dma_burst_size
(C++ member), 472
esp_lcd_dpi_panel_config_t::flags
(C++ member), 472
esp_lcd_dpi_panel_config_t::in_color_format
(C++ member), 471
esp_lcd_dpi_panel_config_t::num_fbs
(C++ member), 472
esp_lcd_dpi_panel_config_t::out_color_format
(C++ member), 472
esp_lcd_dpi_panel_config_t::pixel_format
(C++ member), 471
esp_lcd_dpi_panel_config_t::video_timing
(C++ member), 472
esp_lcd_dpi_panel_config_t::virtual_channel
(C++ member), 471
esp_lcd_dpi_panel_event_callbacks_t
(C++ struct), 472
esp_lcd_dpi_panel_event_callbacks_t::on_color_trans_done
(C++ member), 472
esp_lcd_dpi_panel_event_callbacks_t::on_refresh_done
(C++ member), 472
esp_lcd_dpi_panel_event_data_t (C++
struct), 472
esp_lcd_dpi_panel_general_cb_t (C++
type), 472
esp_lcd_dpi_panel_get_frame_buffer
(C++ function), 470
esp_lcd_dpi_panel_refresh_done_cb_t
(C++ type), 473
esp_lcd_dpi_panel_register_event_callbacks
(C++ function), 470
esp_lcd_dpi_panel_set_color_conversion
(C++ function), 470
esp_lcd_dpi_panel_set_pattern (C++ func-
tion), 470
esp_lcd_dsi_bus_config_t (C++ struct), 470
esp_lcd_dsi_bus_config_t::bus_id (C++
member), 471
esp_lcd_dsi_bus_config_t::lane_bit_rate_mbps
(C++ member), 471
esp_lcd_dsi_bus_config_t::num_data_lanes
(C++ member), 471
esp_lcd_dsi_bus_config_t::phy_clk_src
(C++ member), 471
esp_lcd_dsi_bus_handle_t (C++ type), 472
esp_lcd_i2c_bus_handle_t (C++ type), 461
esp_lcd_i80_alloc_draw_buffer (C++ func-
tion), 464
esp_lcd_i80_bus_config_t (C++ struct), 464
esp_lcd_i80_bus_config_t::bus_width
(C++ member), 465
esp_lcd_i80_bus_config_t::clk_src
(C++ member), 464
esp_lcd_i80_bus_config_t::data_gpio_nums
(C++ member), 465
esp_lcd_i80_bus_config_t::dc_gpio_num
(C++ member), 464
esp_lcd_i80_bus_config_t::dma_burst_size
(C++ member), 465
esp_lcd_i80_bus_config_t::max_transfer_bytes
(C++ member), 465
esp_lcd_i80_bus_config_t::psram_trans_align
(C++ member), 465
esp_lcd_i80_bus_config_t::sram_trans_align
(C++ member), 465
esp_lcd_i80_bus_config_t::wr_gpio_num
(C++ member), 464
esp_lcd_i80_bus_handle_t (C++ type), 466
ESP_LCD_I80_BUS_WIDTH_MAX (C macro), 466
esp_lcd_new_dsi_bus (C++ function), 469
esp_lcd_new_i80_bus (C++ function), 463
esp_lcd_new_panel_dpi (C++ function), 469
esp_lcd_new_panel_io_dbi (C++ function),
469
esp_lcd_new_panel_io_i2c (C macro), 461
esp_lcd_new_panel_io_i2c_v1 (C++ func-
tion), 460
esp_lcd_new_panel_io_i2c_v2 (C++ func-
tion), 460
esp_lcd_new_panel_io_i80 (C++ function),
464
esp_lcd_new_panel_io_spi (C++ function),
457
esp_lcd_panel_del (C++ function), 481
esp_lcd_panel_disp_off (C++ function), 482
esp_lcd_panel_disp_on_off (C++ function),

482
esp_lcd_panel_disp_sleep (C++ function), 483
esp_lcd_panel_draw_bitmap (C++ function), 481
esp_lcd_panel_handle_t (C++ type), 478
esp_lcd_panel_init (C++ function), 481
esp_lcd_panel_invert_color (C++ function), 482
esp_lcd_panel_io_callbacks_t (C++ struct), 477
esp_lcd_panel_io_callbacks_t::on_color_trans_done (C++ member), 477
esp_lcd_panel_io_color_trans_done_cb_t (C++ type), 478
esp_lcd_panel_io_del (C++ function), 480
esp_lcd_panel_io_event_data_t (C++ struct), 477
esp_lcd_panel_io_handle_t (C++ type), 478
esp_lcd_panel_io_i2c_config_t (C++ struct), 460
esp_lcd_panel_io_i2c_config_t::control_phase_bytes (C++ member), 461
esp_lcd_panel_io_i2c_config_t::dc_bit_offset (C++ member), 461
esp_lcd_panel_io_i2c_config_t::dc_low_on_data (C++ member), 461
esp_lcd_panel_io_i2c_config_t::dev_addr (C++ member), 461
esp_lcd_panel_io_i2c_config_t::disable_control_phase (C++ member), 461
esp_lcd_panel_io_i2c_config_t::flags (C++ member), 461
esp_lcd_panel_io_i2c_config_t::lcd_cmd_bits (C++ member), 461
esp_lcd_panel_io_i2c_config_t::lcd_param_bits (C++ member), 461
esp_lcd_panel_io_i2c_config_t::on_color_trans_done (C++ member), 461
esp_lcd_panel_io_i2c_config_t::scl_speed_hz (C++ member), 461
esp_lcd_panel_io_i2c_config_t::user_ctx (C++ member), 461
esp_lcd_panel_io_i80_config_t (C++ struct), 465
esp_lcd_panel_io_i80_config_t::cs_active_high (C++ member), 466
esp_lcd_panel_io_i80_config_t::cs_gpio_num (C++ member), 465
esp_lcd_panel_io_i80_config_t::dc_cmd_level (C++ member), 466
esp_lcd_panel_io_i80_config_t::dc_data_level (C++ member), 466
esp_lcd_panel_io_i80_config_t::dc_dummy_level (C++ member), 466
esp_lcd_panel_io_i80_config_t::dc_idle_level (C++ member), 465
esp_lcd_panel_io_i80_config_t::dc_levels (C++ member), 466
esp_lcd_panel_io_i80_config_t::flags (C++ member), 466
esp_lcd_panel_io_i80_config_t::lcd_cmd_bits (C++ member), 465
esp_lcd_panel_io_i80_config_t::lcd_param_bits (C++ member), 465
esp_lcd_panel_io_i80_config_t::on_color_trans_done (C++ member), 465
esp_lcd_panel_io_i80_config_t::pclk_active_neg (C++ member), 466
esp_lcd_panel_io_i80_config_t::pclk_hz (C++ member), 465
esp_lcd_panel_io_i80_config_t::pclk_idle_low (C++ member), 466
esp_lcd_panel_io_i80_config_t::reverse_color_bits (C++ member), 466
esp_lcd_panel_io_i80_config_t::swap_color_bytes (C++ member), 466
esp_lcd_panel_io_i80_config_t::trans_queue_depth (C++ member), 465
esp_lcd_panel_io_i80_config_t::user_ctx (C++ member), 465
esp_lcd_panel_io_register_event_callbacks (C++ function), 480
esp_lcd_panel_io_rx_param (C++ function), 479
esp_lcd_panel_io_spi_config_t (C++ struct), 457
esp_lcd_panel_io_spi_config_t::cs_gpio_num (C++ member), 457
esp_lcd_panel_io_spi_config_t::cs_high_active (C++ member), 458
esp_lcd_panel_io_spi_config_t::dc_gpio_num (C++ member), 457
esp_lcd_panel_io_spi_config_t::dc_high_on_cmd (C++ member), 458
esp_lcd_panel_io_spi_config_t::dc_low_on_data (C++ member), 458
esp_lcd_panel_io_spi_config_t::dc_low_on_param (C++ member), 458
esp_lcd_panel_io_spi_config_t::flags (C++ member), 458
esp_lcd_panel_io_spi_config_t::lcd_cmd_bits (C++ member), 458
esp_lcd_panel_io_spi_config_t::lcd_param_bits (C++ member), 458
esp_lcd_panel_io_spi_config_t::lsb_first (C++ member), 458
esp_lcd_panel_io_spi_config_t::octal_mode (C++ member), 458
esp_lcd_panel_io_spi_config_t::on_color_trans_done (C++ member), 458
esp_lcd_panel_io_spi_config_t::pclk_hz (C++ member), 458
esp_lcd_panel_io_spi_config_t::quad_mode (C++ member), 458
esp_lcd_panel_io_spi_config_t::sio_mode

(C++ member), 458
esp_lcd_panel_io_spi_config_t::spi_mode (C++ member), 457
esp_lcd_panel_io_spi_config_t::trans_queue_depth (C++ member), 458
esp_lcd_panel_io_spi_config_t::user_ctx (C++ member), 458
esp_lcd_panel_io_tx_color (C++ function), 479
esp_lcd_panel_io_tx_param (C++ function), 479
esp_lcd_panel_mirror (C++ function), 481
esp_lcd_panel_reset (C++ function), 480
esp_lcd_panel_set_gap (C++ function), 482
esp_lcd_panel_swap_xy (C++ function), 481
esp_lcd_spi_bus_handle_t (C++ type), 459
esp_lcd_video_timing_t (C++ struct), 476
esp_lcd_video_timing_t::h_size (C++ member), 477
esp_lcd_video_timing_t::hsync_back_porch (C++ member), 477
esp_lcd_video_timing_t::hsync_front_porch (C++ member), 477
esp_lcd_video_timing_t::hsync_pulse_width (C++ member), 477
esp_lcd_video_timing_t::v_size (C++ member), 477
esp_lcd_video_timing_t::vsync_back_porch (C++ member), 477
esp_lcd_video_timing_t::vsync_front_porch (C++ member), 477
esp_lcd_video_timing_t::vsync_pulse_width (C++ member), 477
esp_ldo_acquire_channel (C++ function), 484
esp_ldo_channel_adjust_voltage (C++ function), 485
esp_ldo_channel_config_t (C++ struct), 485
esp_ldo_channel_config_t::chan_id (C++ member), 485
esp_ldo_channel_config_t::flags (C++ member), 485
esp_ldo_channel_config_t::ldo_extra_flags (C++ struct), 485
esp_ldo_channel_config_t::ldo_extra_flags::adjustable (C++ member), 486
esp_ldo_channel_config_t::ldo_extra_flags::owned_by_hw (C++ member), 486
esp_ldo_channel_config_t::voltage_mv (C++ member), 485
esp_ldo_channel_handle_t (C++ type), 486
esp_ldo_dump (C++ function), 485
esp_ldo_release_channel (C++ function), 485
esp_light_sleep_start (C++ function), 1691
esp_local_ctrl_add_property (C++ function), 93
esp_local_ctrl_config (C++ struct), 97
esp_local_ctrl_config::handlers (C++ member), 97
esp_local_ctrl_config::max_properties (C++ member), 97
esp_local_ctrl_config::proto_sec (C++ member), 97
esp_local_ctrl_config::transport (C++ member), 97
esp_local_ctrl_config::transport_config (C++ member), 97
esp_local_ctrl_config_t (C++ type), 98
esp_local_ctrl_get_property (C++ function), 93
esp_local_ctrl_get_transport_ble (C++ function), 93
esp_local_ctrl_get_transport_httpd (C++ function), 93
esp_local_ctrl_handlers (C++ struct), 95
esp_local_ctrl_handlers::get_prop_values (C++ member), 95
esp_local_ctrl_handlers::set_prop_values (C++ member), 96
esp_local_ctrl_handlers::usr_ctx (C++ member), 96
esp_local_ctrl_handlers::usr_ctx_free_fn (C++ member), 96
esp_local_ctrl_handlers_t (C++ type), 97
esp_local_ctrl_prop (C++ struct), 94
esp_local_ctrl_prop::ctx (C++ member), 95
esp_local_ctrl_prop::ctx_free_fn (C++ member), 95
esp_local_ctrl_prop::flags (C++ member), 94
esp_local_ctrl_prop::name (C++ member), 94
esp_local_ctrl_prop::size (C++ member), 94
esp_local_ctrl_prop::type (C++ member), 94
esp_local_ctrl_prop_t (C++ type), 97
esp_local_ctrl_prop_val (C++ struct), 95
esp_local_ctrl_prop_val::data (C++ member), 95
esp_local_ctrl_prop_val::free_fn (C++ member), 95
esp_local_ctrl_prop_val::size (C++ member), 95
esp_local_ctrl_prop_val_t (C++ type), 97
esp_local_ctrl_proto_sec (C++ enum), 98
esp_local_ctrl_proto_sec::PROTOCOM_SEC0 (C++ enumerator), 98
esp_local_ctrl_proto_sec::PROTOCOM_SEC1 (C++ enumerator), 98
esp_local_ctrl_proto_sec::PROTOCOM_SEC2 (C++ enumerator), 98
esp_local_ctrl_proto_sec::PROTOCOM_SEC_CUSTOM (C++ enumerator), 98
esp_local_ctrl_proto_sec_cfg (C++ struct), 96
esp_local_ctrl_proto_sec_cfg::custom_handle

(C++ member), 96
esp_local_ctrl_proto_sec_cfg::pop (C++ member), 96
esp_local_ctrl_proto_sec_cfg::sec_params (C++ member), 96
esp_local_ctrl_proto_sec_cfg::version (C++ member), 96
esp_local_ctrl_proto_sec_cfg_t (C++ type), 98
esp_local_ctrl_proto_sec_t (C++ type), 98
esp_local_ctrl_remove_property (C++ function), 93
esp_local_ctrl_security1_params_t (C++ type), 98
esp_local_ctrl_security2_params_t (C++ type), 98
esp_local_ctrl_set_handler (C++ function), 93
esp_local_ctrl_start (C++ function), 93
esp_local_ctrl_stop (C++ function), 93
ESP_LOCAL_CTRL_TRANSPORT_BLE (C macro), 97
esp_local_ctrl_transport_config_ble_t (C++ type), 97
esp_local_ctrl_transport_config_httpd_t (C++ type), 98
esp_local_ctrl_transport_config_t (C++ union), 94
esp_local_ctrl_transport_config_t::ble (C++ member), 94
esp_local_ctrl_transport_config_t::httpd (C++ member), 94
ESP_LOCAL_CTRL_TRANSPORT_HTTPD (C macro), 97
esp_local_ctrl_transport_t (C++ type), 97
ESP_LOG_BUFFER_CHAR (C macro), 1633
ESP_LOG_BUFFER_CHAR_LEVEL (C macro), 1633
ESP_LOG_BUFFER_HEX (C macro), 1633
ESP_LOG_BUFFER_HEX_LEVEL (C macro), 1632
ESP_LOG_BUFFER_HEXDUMP (C macro), 1633
ESP_LOG_EARLY_IMPL (C macro), 1634
esp_log_early_timestamp (C++ function), 1632
esp_log_get_level_master (C++ function), 1631
ESP_LOG_LEVEL (C macro), 1634
esp_log_level_get (C++ function), 1631
ESP_LOG_LEVEL_LOCAL (C macro), 1635
esp_log_level_set (C++ function), 1631
esp_log_level_t (C++ enum), 1636
esp_log_level_t::ESP_LOG_DEBUG (C++ enumerator), 1636
esp_log_level_t::ESP_LOG_ERROR (C++ enumerator), 1636
esp_log_level_t::ESP_LOG_INFO (C++ enumerator), 1636
esp_log_level_t::ESP_LOG_NONE (C++ enumerator), 1636
esp_log_level_t::ESP_LOG_VERBOSE (C++ enumerator), 1636
esp_log_level_t::ESP_LOG_WARN (C++ enumerator), 1636
esp_log_set_level_master (C++ function), 1631
esp_log_set_vprintf (C++ function), 1631
esp_log_system_timestamp (C++ function), 1632
esp_log_timestamp (C++ function), 1632
esp_log_write (C++ function), 1632
esp_log_writev (C++ function), 1632
ESP_LOGD (C macro), 1634
ESP_LOGE (C macro), 1634
ESP_LOGI (C macro), 1634
ESP_LOGV (C macro), 1634
ESP_LOGW (C macro), 1634
esp_mac_addr_len_get (C++ function), 1644
esp_mac_type_t (C++ enum), 1644
esp_mac_type_t::ESP_MAC_BASE (C++ enumerator), 1644
esp_mac_type_t::ESP_MAC_BT (C++ enumerator), 1644
esp_mac_type_t::ESP_MAC_EFUSE_CUSTOM (C++ enumerator), 1645
esp_mac_type_t::ESP_MAC_EFUSE_EXT (C++ enumerator), 1645
esp_mac_type_t::ESP_MAC_EFUSE_FACTORY (C++ enumerator), 1644
esp_mac_type_t::ESP_MAC_ETH (C++ enumerator), 1644
esp_mac_type_t::ESP_MAC_IEEE802154 (C++ enumerator), 1644
esp_mac_type_t::ESP_MAC_WIFI_SOFTAP (C++ enumerator), 1644
esp_mac_type_t::ESP_MAC_WIFI_STA (C++ enumerator), 1644
esp_mmu_map (C++ function), 1591
esp_mmu_map_dump_mapped_blocks (C++ function), 1592
esp_mmu_map_get_max_consecutive_free_block_size (C++ function), 1591
ESP_MMU_MMAP_FLAG_PADDR_SHARED (C macro), 1593
esp_mmu_paddr_find_caps (C++ function), 1592
esp_mmu_paddr_to_vaddr (C++ function), 1592
esp_mmu_unmap (C++ function), 1591
esp_mmu_vaddr_to_paddr (C++ function), 1592
esp_mqtt_client_config_t (C++ struct), 43
esp_mqtt_client_config_t (C++ type), 50
esp_mqtt_client_config_t::broker (C++ member), 43
esp_mqtt_client_config_t::broker_t (C++ struct), 43
esp_mqtt_client_config_t::broker_t::address (C++ member), 44
esp_mqtt_client_config_t::broker_t::address_t

(C++ *struct*), 44
esp_mqtt_client_config_t::broker_t::address (C++ *member*), 44
esp_mqtt_client_config_t::broker_t::address::hostname (C++ *member*), 44
esp_mqtt_client_config_t::broker_t::address::path (C++ *member*), 44
esp_mqtt_client_config_t::broker_t::address::port (C++ *member*), 44
esp_mqtt_client_config_t::broker_t::address::transport (C++ *member*), 44
esp_mqtt_client_config_t::broker_t::address::uri (C++ *member*), 44
esp_mqtt_client_config_t::broker_t::verification (C++ *member*), 44
esp_mqtt_client_config_t::broker_t::verification::alpn_protos (C++ *member*), 45
esp_mqtt_client_config_t::broker_t::verification::certificate (C++ *member*), 45
esp_mqtt_client_config_t::broker_t::verification::common_name (C++ *member*), 45
esp_mqtt_client_config_t::broker_t::verification::crt_bundle_attach (C++ *member*), 44
esp_mqtt_client_config_t::broker_t::verification::psk_hint_key (C++ *member*), 45
esp_mqtt_client_config_t::broker_t::verification::skip_cert_common_name_check (C++ *member*), 45
esp_mqtt_client_config_t::broker_t::verification::use_global_ca_store (C++ *member*), 44
esp_mqtt_client_config_t::buffer (C++ *member*), 43
esp_mqtt_client_config_t::buffer_t (C++ *struct*), 45
esp_mqtt_client_config_t::buffer_t::out_size (C++ *member*), 45
esp_mqtt_client_config_t::buffer_t::size (C++ *member*), 45
esp_mqtt_client_config_t::credentials (C++ *member*), 43
esp_mqtt_client_config_t::credentials_t (C++ *struct*), 45
esp_mqtt_client_config_t::credentials_t::authentication (C++ *member*), 46
esp_mqtt_client_config_t::credentials_t::authentication::certificate (C++ *member*), 46
esp_mqtt_client_config_t::credentials_t::authentication::key (C++ *member*), 46
esp_mqtt_client_config_t::credentials_t::authentication::key_len (C++ *member*), 46
esp_mqtt_client_config_t::credentials_t::authentication::key_password (C++ *member*), 46
esp_mqtt_client_config_t::credentials_t::authentication::password (C++ *member*), 46
esp_mqtt_client_config_t::credentials_t::client_id (C++ *member*), 45
esp_mqtt_client_config_t::credentials_t::set_null_client_id (C++ *member*), 46
esp_mqtt_client_config_t::credentials_t::username (C++ *member*), 45
esp_mqtt_client_config_t::network (C++ *member*), 43
esp_mqtt_client_config_t::network_t (C++ *struct*), 47
esp_mqtt_client_config_t::network_t::disable_auto_reconnect (C++ *member*), 47
esp_mqtt_client_config_t::network_t::if_name (C++ *member*), 47
esp_mqtt_client_config_t::network_t::reconnect_timeout_ms (C++ *member*), 47
esp_mqtt_client_config_t::network_t::refresh_connection_after_ms (C++ *member*), 47
esp_mqtt_client_config_t::network_t::timeout_ms (C++ *member*), 47
esp_mqtt_client_config_t::network_t::transport (C++ *member*), 47
esp_mqtt_client_config_t::outbox (C++ *member*), 43
esp_mqtt_client_config_t::outbox_config_t (C++ *struct*), 47
esp_mqtt_client_config_t::outbox_config_t::limit (C++ *member*), 47
esp_mqtt_client_config_t::session (C++ *member*), 43
esp_mqtt_client_config_t::session_t (C++ *struct*), 47
esp_mqtt_client_config_t::session_t::disable_clean_session (C++ *member*), 47
esp_mqtt_client_config_t::session_t::disable_keepalive (C++ *member*), 48
esp_mqtt_client_config_t::session_t::keepalive (C++ *member*), 48
esp_mqtt_client_config_t::session_t::last_will (C++ *member*), 47
esp_mqtt_client_config_t::session_t::last_will_t (C++ *struct*), 48
esp_mqtt_client_config_t::session_t::last_will_t::msg_len (C++ *member*), 48
esp_mqtt_client_config_t::session_t::last_will_t::msg_to_heap (C++ *member*), 48
esp_mqtt_client_config_t::session_t::last_will_t::qos (C++ *member*), 48
esp_mqtt_client_config_t::session_t::last_will_t::retain (C++ *member*), 48
esp_mqtt_client_config_t::session_t::last_will_t::topic (C++ *member*), 48

(C++ *member*), 48
esp_mqtt_client_config_t::session_t::message_retransmit_timeout (C++ *member*), 48
esp_mqtt_client_config_t::session_t::protocol_ver (C++ *member*), 48
esp_mqtt_client_config_t::task (C++ *member*), 43
esp_mqtt_client_config_t::task_t (C++ *struct*), 48
esp_mqtt_client_config_t::task_t::priority (C++ *member*), 48
esp_mqtt_client_config_t::task_t::stack_size (C++ *member*), 48
esp_mqtt_client_destroy (C++ *function*), 40
esp_mqtt_client_disconnect (C++ *function*), 38
esp_mqtt_client_enqueue (C++ *function*), 40
esp_mqtt_client_get_outbox_size (C++ *function*), 41
esp_mqtt_client_handle_t (C++ *type*), 49
esp_mqtt_client_init (C++ *function*), 38
esp_mqtt_client_publish (C++ *function*), 39
esp_mqtt_client_reconnect (C++ *function*), 38
esp_mqtt_client_register_event (C++ *function*), 40
esp_mqtt_client_set_uri (C++ *function*), 38
esp_mqtt_client_start (C++ *function*), 38
esp_mqtt_client_stop (C++ *function*), 38
esp_mqtt_client_subscribe (C *macro*), 49
esp_mqtt_client_subscribe_multiple (C++ *function*), 39
esp_mqtt_client_subscribe_single (C++ *function*), 39
esp_mqtt_client_unregister_event (C++ *function*), 41
esp_mqtt_client_unsubscribe (C++ *function*), 39
esp_mqtt_connect_return_code_t (C++ *enum*), 51
esp_mqtt_connect_return_code_t (C++ *type*), 49
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED (C++ *enumerator*), 51
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_REFUSE_BAD_USERNAME (C++ *enumerator*), 52
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_REFUSE_ID_REJECTED (C++ *enumerator*), 52
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_REFUSE_NOT_AUTHORIZED (C++ *enumerator*), 52
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_REFUSE_PROTOCOL (C++ *enumerator*), 52
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_REFUSE_SERVER_UNAVAILABLE (C++ *enumerator*), 52
esp_mqtt_dispatch_custom_event (C++ *function*), 41
esp_mqtt_error_codes (C++ *struct*), 41
esp_mqtt_error_codes::connect_return_code (C++ *member*), 42
esp_mqtt_error_codes::esp_tls_cert_verify_flags (C++ *member*), 42
esp_mqtt_error_codes::esp_tls_last_esp_err (C++ *member*), 41
esp_mqtt_error_codes::esp_tls_stack_err (C++ *member*), 42
esp_mqtt_error_codes::esp_transport_sock_errno (C++ *member*), 42
esp_mqtt_error_codes_t (C++ *type*), 50
esp_mqtt_error_type_t (C++ *enum*), 52
esp_mqtt_error_type_t (C++ *type*), 49
esp_mqtt_error_type_t::MQTT_ERROR_TYPE_CONNECTION_REFUSED (C++ *enumerator*), 52
esp_mqtt_error_type_t::MQTT_ERROR_TYPE_NONE (C++ *enumerator*), 52
esp_mqtt_error_type_t::MQTT_ERROR_TYPE_SUBSCRIBE_FAILED (C++ *enumerator*), 52
esp_mqtt_error_type_t::MQTT_ERROR_TYPE_TCP_TRANSPORT (C++ *enumerator*), 52
esp_mqtt_event_handle_t (C++ *type*), 50
esp_mqtt_event_id_t (C++ *enum*), 50
esp_mqtt_event_id_t (C++ *type*), 49
esp_mqtt_event_id_t::MQTT_EVENT_ANY (C++ *enumerator*), 50
esp_mqtt_event_id_t::MQTT_EVENT_BEFORE_CONNECT (C++ *enumerator*), 51
esp_mqtt_event_id_t::MQTT_EVENT_CONNECTED (C++ *enumerator*), 50
esp_mqtt_event_id_t::MQTT_EVENT_DATA (C++ *enumerator*), 51
esp_mqtt_event_id_t::MQTT_EVENT_DELETED (C++ *enumerator*), 51
esp_mqtt_event_id_t::MQTT_EVENT_DISCONNECTED (C++ *enumerator*), 51
esp_mqtt_event_id_t::MQTT_EVENT_ERROR (C++ *enumerator*), 50
esp_mqtt_event_id_t::MQTT_EVENT_PUBLISHED (C++ *enumerator*), 51
esp_mqtt_event_id_t::MQTT_EVENT_SUBSCRIBED (C++ *enumerator*), 51
esp_mqtt_event_id_t::MQTT_EVENT_UNSUBSCRIBED (C++ *enumerator*), 51
esp_mqtt_event_id_t::MQTT_USER_EVENT (C++ *enumerator*), 51
esp_mqtt_event_t (C++ *struct*), 42
esp_mqtt_event_t (C++ *type*), 50
esp_mqtt_event_t::client (C++ *member*), 42
esp_mqtt_event_t::current_data_offset (C++ *member*), 42
esp_mqtt_event_t::data (C++ *member*), 42
esp_mqtt_event_t::data_len (C++ *member*), 42
esp_mqtt_event_t::dup (C++ *member*), 43
esp_mqtt_event_t::error_handle (C++ *member*), 43

esp_mqtt_event_t::event_id (C++ member), 42
esp_mqtt_event_t::msg_id (C++ member), 42
esp_mqtt_event_t::protocol_ver (C++ member), 43
esp_mqtt_event_t::qos (C++ member), 43
esp_mqtt_event_t::retain (C++ member), 43
esp_mqtt_event_t::session_present (C++ member), 42
esp_mqtt_event_t::topic (C++ member), 42
esp_mqtt_event_t::topic_len (C++ member), 42
esp_mqtt_event_t::total_data_len (C++ member), 42
esp_mqtt_protocol_ver_t (C++ enum), 52
esp_mqtt_protocol_ver_t (C++ type), 50
esp_mqtt_protocol_ver_t::MQTT_PROTOCOL_UNDEFINED (C++ enumerator), 53
esp_mqtt_protocol_ver_t::MQTT_PROTOCOL_V_3_1 (C++ enumerator), 53
esp_mqtt_protocol_ver_t::MQTT_PROTOCOL_V_3_1_1 (C++ enumerator), 53
esp_mqtt_protocol_ver_t::MQTT_PROTOCOL_V_5 (C++ enumerator), 53
esp_mqtt_set_config (C++ function), 40
esp_mqtt_topic_t (C++ type), 50
esp_mqtt_transport_t (C++ enum), 52
esp_mqtt_transport_t (C++ type), 49
esp_mqtt_transport_t::MQTT_TRANSPORT_OVER_SSL (C++ enumerator), 52
esp_mqtt_transport_t::MQTT_TRANSPORT_OVER_TCP (C++ enumerator), 52
esp_mqtt_transport_t::MQTT_TRANSPORT_OVER_WS (C++ enumerator), 52
esp_mqtt_transport_t::MQTT_TRANSPORT_OVER_WSS (C++ enumerator), 52
esp_mqtt_transport_t::MQTT_TRANSPORT_UNKNOWN (C++ enumerator), 52
esp_netif_action_add_ip6_address (C++ function), 227
esp_netif_action_connected (C++ function), 226
esp_netif_action_disconnected (C++ function), 226
esp_netif_action_got_ip (C++ function), 226
esp_netif_action_join_ip6_multicast_group (C++ function), 226
esp_netif_action_leave_ip6_multicast_group (C++ function), 227
esp_netif_action_remove_ip6_address (C++ function), 227
esp_netif_action_start (C++ function), 225
esp_netif_action_stop (C++ function), 225
esp_netif_add_ip6_address (C++ function), 235
esp_netif_attach (C++ function), 225
ESP_NETIF_BR_DROP (C macro), 244
ESP_NETIF_BR_FDW_CPU (C macro), 244
ESP_NETIF_BR_FLOOD (C macro), 244
esp_netif_callback_fn (C++ type), 237
esp_netif_config (C++ struct), 243
esp_netif_config::base (C++ member), 243
esp_netif_config::driver (C++ member), 243
esp_netif_config::stack (C++ member), 243
esp_netif_config_t (C++ type), 245
esp_netif_create_ip6_linklocal (C++ function), 234
ESP_NETIF_DEFAULT_OPENTHREAD (C macro), 216
esp_netif_deinit (C++ function), 224
esp_netif_destroy (C++ function), 224
esp_netif_dhcp_option_id_t (C++ enum), 246
esp_netif_dhcp_option_id_t::ESP_NETIF_DOMAIN_NAME (C++ enumerator), 246
esp_netif_dhcp_option_id_t::ESP_NETIF_IP_ADDRESS_LEASE_TIME (C++ enumerator), 246
esp_netif_dhcp_option_id_t::ESP_NETIF_IP_REQUEST_RETRY_COUNT (C++ enumerator), 246
esp_netif_dhcp_option_id_t::ESP_NETIF_REQUESTED_IP_ADDRESS (C++ enumerator), 246
esp_netif_dhcp_option_id_t::ESP_NETIF_ROUTER_SOLICITATION (C++ enumerator), 246
esp_netif_dhcp_option_id_t::ESP_NETIF_SUBNET_MASK (C++ enumerator), 246
esp_netif_dhcp_option_id_t::ESP_NETIF_VENDOR_CLASS_IDENTIFIER (C++ enumerator), 246
esp_netif_dhcp_option_id_t::ESP_NETIF_VENDOR_SPECIFIC_INFO (C++ enumerator), 247
esp_netif_dhcp_option_mode_t (C++ enum), 246
esp_netif_dhcp_option_mode_t::ESP_NETIF_OP_GET (C++ enumerator), 246
esp_netif_dhcp_option_mode_t::ESP_NETIF_OP_MAX (C++ enumerator), 246
esp_netif_dhcp_option_mode_t::ESP_NETIF_OP_SET (C++ enumerator), 246
esp_netif_dhcp_option_mode_t::ESP_NETIF_OP_START (C++ enumerator), 246
esp_netif_dhcp_status_t (C++ enum), 245
esp_netif_dhcp_status_t::ESP_NETIF_DHCP_INIT (C++ enumerator), 245
esp_netif_dhcp_status_t::ESP_NETIF_DHCP_STARTED (C++ enumerator), 246
esp_netif_dhcp_status_t::ESP_NETIF_DHCP_STATUS_MAX (C++ enumerator), 246
esp_netif_dhcp_status_t::ESP_NETIF_DHCP_STOPPED (C++ enumerator), 246
esp_netif_dhcpc_get_status (C++ function), 232
esp_netif_dhcpc_option (C++ function), 231
esp_netif_dhcpc_start (C++ function), 231
esp_netif_dhcpc_stop (C++ function), 232
esp_netif_dhcps_get_clients_by_mac (C++ function), 233

esp_netif_dhcps_get_status (*C++ function*), 232
esp_netif_dhcps_option (*C++ function*), 231
esp_netif_dhcps_start (*C++ function*), 232
esp_netif_dhcps_stop (*C++ function*), 232
esp_netif_dns_info_t (*C++ struct*), 240
esp_netif_dns_info_t::ip (*C++ member*), 240
esp_netif_dns_type_t (*C++ enum*), 245
esp_netif_dns_type_t::ESP_NETIF_DNS_BACKUP (*C++ enumerator*), 245
esp_netif_dns_type_t::ESP_NETIF_DNS_FALLBACK (*C++ enumerator*), 245
esp_netif_dns_type_t::ESP_NETIF_DNS_MAIN (*C++ enumerator*), 245
esp_netif_dns_type_t::ESP_NETIF_DNS_MAX (*C++ enumerator*), 245
esp_netif_driver_base_s (*C++ struct*), 243
esp_netif_driver_base_s::netif (*C++ member*), 243
esp_netif_driver_base_s::post_attach (*C++ member*), 243
esp_netif_driver_base_t (*C++ type*), 245
esp_netif_driver_ifconfig (*C++ struct*), 243
esp_netif_driver_ifconfig::driver_free_rx_buffer (*C++ member*), 243
esp_netif_driver_ifconfig::handle (*C++ member*), 243
esp_netif_driver_ifconfig::transmit (*C++ member*), 243
esp_netif_driver_ifconfig::transmit_wrap (*C++ member*), 243
esp_netif_driver_ifconfig_t (*C++ type*), 245
esp_netif_find_if (*C++ function*), 237
esp_netif_find_predicate_t (*C++ type*), 237
esp_netif_flags (*C++ enum*), 247
esp_netif_flags::ESP_NETIF_DHCP_CLIENT (*C++ enumerator*), 247
esp_netif_flags::ESP_NETIF_DHCP_SERVER (*C++ enumerator*), 247
esp_netif_flags::ESP_NETIF_FLAG_AUTOUP (*C++ enumerator*), 247
esp_netif_flags::ESP_NETIF_FLAG_EVENT_IP_MODIFIED (*C++ enumerator*), 247
esp_netif_flags::ESP_NETIF_FLAG_GARP (*C++ enumerator*), 247
esp_netif_flags::ESP_NETIF_FLAG_IS_BRIDGE (*C++ enumerator*), 248
esp_netif_flags::ESP_NETIF_FLAG_IS_PPP (*C++ enumerator*), 247
esp_netif_flags::ESP_NETIF_FLAG_MLDV6_REPORT (*C++ enumerator*), 248
esp_netif_flags_t (*C++ type*), 245
esp_netif_free_rx_buffer (*C++ function*), 255
esp_netif_get_all_ip6 (*C++ function*), 234
esp_netif_get_all_preferred_ip6 (*C++ function*), 234
esp_netif_get_default_netif (*C++ function*), 227
esp_netif_get_desc (*C++ function*), 236
esp_netif_get_dns_info (*C++ function*), 233
esp_netif_get_event_id (*C++ function*), 236
esp_netif_get_flags (*C++ function*), 236
esp_netif_get_handle_from_ifkey (*C++ function*), 236
esp_netif_get_handle_from_netif_impl (*C++ function*), 254
esp_netif_get_hostname (*C++ function*), 228
esp_netif_get_ifkey (*C++ function*), 236
esp_netif_get_io_driver (*C++ function*), 236
esp_netif_get_ip6_global (*C++ function*), 234
esp_netif_get_ip6_linklocal (*C++ function*), 234
esp_netif_get_ip_info (*C++ function*), 229
esp_netif_get_mac (*C++ function*), 228
esp_netif_get_netif_impl (*C++ function*), 254
esp_netif_get_netif_impl_index (*C++ function*), 230
esp_netif_get_netif_impl_name (*C++ function*), 230
esp_netif_get_nr_of_ifs (*C++ function*), 237
esp_netif_get_old_ip_info (*C++ function*), 229
esp_netif_get_route_prio (*C++ function*), 236
esp_netif_htonl (*C macro*), 249
esp_netif_inherent_config (*C++ struct*), 242
esp_netif_inherent_config::bridge_info (*C++ member*), 242
esp_netif_inherent_config::flags (*C++ member*), 242
esp_netif_inherent_config::get_ip_event (*C++ member*), 242
esp_netif_inherent_config::if_desc (*C++ member*), 242
esp_netif_inherent_config::if_key (*C++ member*), 242
esp_netif_inherent_config::ip_info (*C++ member*), 242
esp_netif_inherent_config::lost_ip_event (*C++ member*), 242
esp_netif_inherent_config::mac (*C++ member*), 242
esp_netif_inherent_config::route_prio (*C++ member*), 242
esp_netif_inherent_config_t (*C++ type*), 245
ESP_NETIF_INHERENT_DEFAULT_OPENTHREAD (*C macro*), 216
esp_netif_init (*C++ function*), 224
esp_netif_iodriver_handle (*C++ type*), 245
esp_netif_ip4_makeu32 (*C macro*), 249

esp_netif_ip6_get_addr_type (*C++ function*), 248
esp_netif_ip6_info_t (*C++ struct*), 240
esp_netif_ip6_info_t::ip (*C++ member*), 240
esp_netif_ip_addr_copy (*C++ function*), 248
esp_netif_ip_event_type (*C++ enum*), 248
esp_netif_ip_event_type::ESP_NETIF_IP_EVENT_GOT_IP (*C++ enumerator*), 248
esp_netif_ip_event_type::ESP_NETIF_IP_EVENT_LOST_IP (*C++ enumerator*), 248
esp_netif_ip_event_type_t (*C++ type*), 245
esp_netif_ip_info_t (*C++ struct*), 240
esp_netif_ip_info_t::gw (*C++ member*), 240
esp_netif_ip_info_t::ip (*C++ member*), 240
esp_netif_ip_info_t::netmask (*C++ member*), 240
esp_netif_is_netif_up (*C++ function*), 229
esp_netif_join_ip6_multicast_group (*C++ function*), 227
esp_netif_leave_ip6_multicast_group (*C++ function*), 228
esp_netif_napt_disable (*C++ function*), 231
esp_netif_napt_enable (*C++ function*), 230
esp_netif_netstack_buf_free (*C++ function*), 237
esp_netif_netstack_buf_ref (*C++ function*), 237
esp_netif_netstack_config_t (*C++ type*), 245
esp_netif_new (*C++ function*), 224
esp_netif_next (*C++ function*), 236
esp_netif_next_unsafe (*C++ function*), 237
esp_netif_pair_mac_ip_t (*C++ struct*), 243
esp_netif_pair_mac_ip_t::ip (*C++ member*), 244
esp_netif_pair_mac_ip_t::mac (*C++ member*), 244
esp_netif_receive (*C++ function*), 225
esp_netif_receive_t (*C++ type*), 245
esp_netif_remove_ip6_address (*C++ function*), 235
esp_netif_set_default_netif (*C++ function*), 227
esp_netif_set_dns_info (*C++ function*), 233
esp_netif_set_driver_config (*C++ function*), 224
esp_netif_set_hostname (*C++ function*), 228
esp_netif_set_ip4_addr (*C++ function*), 235
esp_netif_set_ip_info (*C++ function*), 229
esp_netif_set_link_speed (*C++ function*), 254
esp_netif_set_mac (*C++ function*), 228
esp_netif_set_old_ip_info (*C++ function*), 230
ESP_NETIF_SNTP_DEFAULT_CONFIG (*C macro*), 239
ESP_NETIF_SNTP_DEFAULT_CONFIG_MULTIPLE (*C macro*), 239
esp_netif_sntp_deinit (*C++ function*), 238
esp_netif_sntp_init (*C++ function*), 238
esp_netif_sntp_reachability (*C++ function*), 238
esp_netif_sntp_start (*C++ function*), 238
esp_netif_sntp_sync_wait (*C++ function*), 238
esp_netif_str_to_ip4 (*C++ function*), 235
esp_netif_str_to_ip6 (*C++ function*), 236
esp_netif_t (*C++ type*), 245
esp_netif_tcpip_exec (*C++ function*), 237
esp_netif_transmit (*C++ function*), 254
esp_netif_transmit_wrap (*C++ function*), 255
esp_ng_type_t (*C++ enum*), 1250
esp_ng_type_t::ESP_NG_3072 (*C++ enumerator*), 1250
ESP_OK (*C macro*), 1407
ESP_OK_EFUSE_CNT (*C macro*), 1404
esp_openthread_auto_start (*C++ function*), 207
esp_openthread_border_router_deinit (*C++ function*), 217
esp_openthread_border_router_init (*C++ function*), 216
esp_openthread_compatibility_error_callback (*C++ type*), 212
esp_openthread_dataset_changed_event_t (*C++ struct*), 208
esp_openthread_dataset_changed_event_t::new_datas (*C++ member*), 209
esp_openthread_dataset_changed_event_t::type (*C++ member*), 209
esp_openthread_dataset_type_t (*C++ enum*), 213
esp_openthread_dataset_type_t::OPENTHREAD_ACTIVE_ (*C++ enumerator*), 213
esp_openthread_dataset_type_t::OPENTHREAD_PENDING (*C++ enumerator*), 213
esp_openthread_deinit (*C++ function*), 208
esp_openthread_event_t (*C++ enum*), 212
esp_openthread_event_t::OPENTHREAD_EVENT_ATTACHED (*C++ enumerator*), 212
esp_openthread_event_t::OPENTHREAD_EVENT_DATASET_ (*C++ enumerator*), 213
esp_openthread_event_t::OPENTHREAD_EVENT_DETACHED (*C++ enumerator*), 212
esp_openthread_event_t::OPENTHREAD_EVENT_GOT_IP6 (*C++ enumerator*), 212
esp_openthread_event_t::OPENTHREAD_EVENT_IF_DOWN (*C++ enumerator*), 212
esp_openthread_event_t::OPENTHREAD_EVENT_IF_UP (*C++ enumerator*), 212
esp_openthread_event_t::OPENTHREAD_EVENT_LOST_IP6 (*C++ enumerator*), 212
esp_openthread_event_t::OPENTHREAD_EVENT_MULTICAS (*C++ enumerator*), 212
esp_openthread_event_t::OPENTHREAD_EVENT_MULTICAS

(C++ enumerator), 213
esp_openthread_event_t::OPENTHREAD_EVENT_PUBLISH_MESHCOP_E (C++ enumerator), 213
esp_openthread_event_t::OPENTHREAD_EVENT_REMOVE_MESHCOP_E (C++ enumerator), 213
esp_openthread_event_t::OPENTHREAD_EVENT_ROLE_CHANGED (C++ enumerator), 212
esp_openthread_event_t::OPENTHREAD_EVENT_SET_DNS_SERVER (C++ enumerator), 213
esp_openthread_event_t::OPENTHREAD_EVENT_START (C++ enumerator), 212
esp_openthread_event_t::OPENTHREAD_EVENT_STOP (C++ enumerator), 212
esp_openthread_event_t::OPENTHREAD_EVENT_TREL_ADD_IP6 (C++ enumerator), 213
esp_openthread_event_t::OPENTHREAD_EVENT_TREL_MULTICAST_GROUP_JOIN (C++ enumerator), 213
esp_openthread_event_t::OPENTHREAD_EVENT_TREL_REMOVE_IP6 (C++ enumerator), 213
esp_openthread_get_backbone_netif (C++ function), 217
esp_openthread_get_instance (C++ function), 208
esp_openthread_get_netif (C++ function), 216
esp_openthread_host_connection_config_t (C++ struct), 211
esp_openthread_host_connection_config_t::host_connection_mode (C++ member), 211
esp_openthread_host_connection_config_t::host_uart_config (C++ member), 211
esp_openthread_host_connection_config_t::host_uart_port (C++ member), 211
esp_openthread_host_connection_config_t::spi_slave_config (C++ member), 211
esp_openthread_host_connection_mode_t (C++ enum), 214
esp_openthread_host_connection_mode_t::HOST_CONNECTION_MODE_CLI_UART (C++ enumerator), 214
esp_openthread_host_connection_mode_t::HOST_CONNECTION_MODE_CLI_USB (C++ enumerator), 214
esp_openthread_host_connection_mode_t::HOST_CONNECTION_MODE_MAX (C++ enumerator), 214
esp_openthread_host_connection_mode_t::HOST_CONNECTION_MODE_NONE (C++ enumerator), 214
esp_openthread_host_connection_mode_t::HOST_CONNECTION_MODE_RCP_SPI (C++ enumerator), 214
esp_openthread_host_connection_mode_t::HOST_CONNECTION_MODE_RCP_UART (C++ enumerator), 214
esp_openthread_init (C++ function), 207
esp_openthread_launch_mainloop (C++ function), 207
esp_openthread_lock_acquire (C++ function), 215
esp_openthread_lock_deinit (C++ function), 214
esp_openthread_lock_init (C++ function), 214
esp_openthread_lock_release (C++ function), 215
esp_openthread_mainloop_context_t (C++ struct), 209
esp_openthread_mainloop_context_t::error_fds (C++ member), 209
esp_openthread_mainloop_context_t::max_fd (C++ member), 209
esp_openthread_mainloop_context_t::read_fds (C++ member), 209
esp_openthread_mainloop_context_t::timeout (C++ member), 209
esp_openthread_mainloop_context_t::write_fds (C++ member), 209
esp_openthread_netif_glue_deinit (C++ function), 216
esp_openthread_netif_glue_init (C++ function), 216
esp_openthread_platform_config_t (C++ struct), 211
esp_openthread_platform_config_t::host_config (C++ member), 212
esp_openthread_platform_config_t::port_config (C++ member), 212
esp_openthread_platform_config_t::radio_config (C++ member), 211
esp_openthread_port_config_t (C++ struct), 211
esp_openthread_port_config_t::netif_queue_size (C++ member), 211
esp_openthread_port_config_t::storage_partition_n (C++ member), 211
esp_openthread_port_config_t::task_queue_size (C++ member), 211
esp_openthread_radio_config_t (C++ struct), 210
esp_openthread_radio_config_t::radio_mode (C++ member), 210
esp_openthread_radio_config_t::radio_spi_config (C++ member), 210
esp_openthread_radio_config_t::radio_uart_config (C++ member), 210
esp_openthread_radio_mode_t (C++ enum), 213
esp_openthread_radio_mode_t::RADIO_MODE_MAX (C++ enumerator), 214
esp_openthread_radio_mode_t::RADIO_MODE_NATIVE (C++ enumerator), 213
esp_openthread_radio_mode_t::RADIO_MODE_SPI_RCP (C++ enumerator), 213
esp_openthread_radio_mode_t::RADIO_MODE_TREL (C++ enumerator), 213
esp_openthread_radio_mode_t::RADIO_MODE_UART_RCP (C++ enumerator), 213
esp_openthread_rcp_deinit (C++ function), 217
esp_openthread_rcp_failure_handler (C++ type), 212

esp_openthread_rcp_init (*C++ function*), 217
esp_openthread_register_meshcop_e_handler (*C++ function*), 216
esp_openthread_register_rcp_failure_handler (*C++ function*), 217
esp_openthread_role_changed_event_t (*C++ struct*), 208
esp_openthread_role_changed_event_t::current_role (*C++ member*), 208
esp_openthread_role_changed_event_t::previous_role (*C++ member*), 208
esp_openthread_set_backbone_netif (*C++ function*), 216
esp_openthread_set_compatibility_error_callback (*C++ function*), 217
esp_openthread_set_meshcop_instance_name (*C++ function*), 217
esp_openthread_spi_host_config_t (*C++ struct*), 210
esp_openthread_spi_host_config_t::dma_channel (*C++ member*), 210
esp_openthread_spi_host_config_t::host_device (*C++ member*), 210
esp_openthread_spi_host_config_t::intr_pin (*C++ member*), 210
esp_openthread_spi_host_config_t::spi_device (*C++ member*), 210
esp_openthread_spi_host_config_t::spi_interface (*C++ member*), 210
esp_openthread_spi_slave_config_t (*C++ struct*), 210
esp_openthread_spi_slave_config_t::bus_config (*C++ member*), 210
esp_openthread_spi_slave_config_t::host_device (*C++ member*), 210
esp_openthread_spi_slave_config_t::intr_pin (*C++ member*), 210
esp_openthread_spi_slave_config_t::slave_config (*C++ member*), 210
esp_openthread_task_switching_lock_acquire (*C++ function*), 215
esp_openthread_task_switching_lock_release (*C++ function*), 215
esp_openthread_uart_config_t (*C++ struct*), 209
esp_openthread_uart_config_t::port (*C++ member*), 209
esp_openthread_uart_config_t::rx_pin (*C++ member*), 209
esp_openthread_uart_config_t::tx_pin (*C++ member*), 209
esp_openthread_uart_config_t::uart_config (*C++ member*), 209
esp_ota_abort (*C++ function*), 1662
esp_ota_begin (*C++ function*), 1660
esp_ota_check_rollback_is_possible (*C++ function*), 1664
esp_ota_end (*C++ function*), 1661
esp_ota_erase_last_boot_app_partition (*C++ function*), 1664
esp_ota_get_app_description (*C++ function*), 1659
esp_ota_get_app_elf_sha256 (*C++ function*), 1660
esp_ota_get_app_partition_count (*C++ function*), 1663
esp_ota_get_boot_partition (*C++ function*), 1662
esp_ota_get_bootloader_description (*C++ function*), 1663
esp_ota_get_last_invalid_partition (*C++ function*), 1664
esp_ota_get_next_update_partition (*C++ function*), 1663
esp_ota_get_partition_description (*C++ function*), 1663
esp_ota_get_running_partition (*C++ function*), 1662
esp_ota_get_state_partition (*C++ function*), 1664
esp_ota_handle_t (*C++ type*), 1665
esp_ota_mark_app_invalid_rollback_and_reboot (*C++ function*), 1664
esp_ota_mark_app_valid_cancel_rollback (*C++ function*), 1663
esp_ota_revoke_secure_boot_public_key (*C++ function*), 1664
esp_ota_secure_boot_public_key_index_t (*C++ enum*), 1665
esp_ota_secure_boot_public_key_index_t::SECURE_BOOT_PUBLIC_KEY_INDEX_0 (*C++ enumerator*), 1665
esp_ota_secure_boot_public_key_index_t::SECURE_BOOT_PUBLIC_KEY_INDEX_1 (*C++ enumerator*), 1665
esp_ota_secure_boot_public_key_index_t::SECURE_BOOT_PUBLIC_KEY_INDEX_2 (*C++ enumerator*), 1666
esp_ota_set_boot_partition (*C++ function*), 1662
esp_ota_write (*C++ function*), 1661
esp_ota_write_with_offset (*C++ function*), 1661
esp_paddr_t (*C++ type*), 1593
esp_partition_check_identity (*C++ function*), 1316
esp_partition_deregister_external (*C++ function*), 1316
esp_partition_erase_range (*C++ function*), 1314
esp_partition_find (*C++ function*), 1312
esp_partition_find_first (*C++ function*), 1312
esp_partition_get (*C++ function*), 1312
esp_partition_get_sha256 (*C++ function*), 1315
esp_partition_iterator_release (*C++ function*), 1313
esp_partition_iterator_t (*C++ type*), 1317

esp_partition_mmap (*C++ function*), 1315
esp_partition_mmap_handle_t (*C++ type*), 1317
esp_partition_mmap_memory_t (*C++ enum*), 1318
esp_partition_mmap_memory_t::ESP_PARTITION_MMAP_DATA (*C++ enumerator*), 1318
esp_partition_mmap_memory_t::ESP_PARTITION_MMAP_INST (*C++ enumerator*), 1318
esp_partition_munmap (*C++ function*), 1315
esp_partition_next (*C++ function*), 1312
esp_partition_read (*C++ function*), 1313
esp_partition_read_raw (*C++ function*), 1314
esp_partition_register_external (*C++ function*), 1316
ESP_PARTITION_SUBTYPE_OTA (*C macro*), 1317
esp_partition_subtype_t (*C++ enum*), 1318
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_ANY (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_FACTORY (*C++ enumerator*), 1318
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_0 (*C++ enumerator*), 1318
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_1 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_10 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_11 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_12 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_13 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_14 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_15 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_2 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_3 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_4 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_5 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_6 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_7 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_8 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_9 (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_MAX (*C++ enumerator*), 1319
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_TEST (*C++ enumerator*), 1318
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_ANY (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_COREDUMP (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_EFUSE_EM (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_ESPHTTPD (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_FAT (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_FW_UPGRADE (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_NAMESPACE (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_NEC_FAT (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_NVS (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_NVS_KEYS (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_OTA (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_PHY (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_SPIFFS (*C++ enumerator*), 1320
esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_UNDEFINED (*C++ enumerator*), 1320
esp_partition_t (*C++ struct*), 1316
esp_partition_t::address (*C++ member*), 1317
esp_partition_t::encrypted (*C++ member*), 1317
esp_partition_t::erase_size (*C++ member*), 1317
esp_partition_t::flash_chip (*C++ member*), 1317
esp_partition_t::label (*C++ member*), 1317
esp_partition_t::readonly (*C++ member*), 1317
esp_partition_t::size (*C++ member*), 1317
esp_partition_t::subtype (*C++ member*), 1317
esp_partition_t::type (*C++ member*), 1317
esp_partition_type_t (*C++ enum*), 1318
esp_partition_type_t::ESP_PARTITION_TYPE_ANY (*C++ enumerator*), 1318
esp_partition_type_t::ESP_PARTITION_TYPE_APP (*C++ enumerator*), 1318
esp_partition_type_t::ESP_PARTITION_TYPE_DATA (*C++ enumerator*), 1318
esp_partition_unload_all (*C++ function*), 1316
esp_partition_verify (*C++ function*), 1313
esp_partition_write (*C++ function*), 1313
esp_partition_write_raw (*C++ function*), 1314
ESP_PD_DOMAIN_RTC8M (*C macro*), 1693
esp_ping_callbacks_t (*C++ struct*), 151
esp_ping_callbacks_t::cb_args (*C++ member*), 151
esp_ping_callbacks_t::on_ping_end (*C++ member*), 151

esp_ping_callbacks_t::on_ping_success (C++ member), 151
esp_ping_callbacks_t::on_ping_timeout (C++ member), 151
esp_ping_config_t (C++ struct), 151
esp_ping_config_t::count (C++ member), 152
esp_ping_config_t::data_size (C++ member), 152
esp_ping_config_t::interval_ms (C++ member), 152
esp_ping_config_t::target_addr (C++ member), 152
esp_ping_config_t::task_prio (C++ member), 152
esp_ping_config_t::task_stack_size (C++ member), 152
esp_ping_config_t::timeout_ms (C++ member), 152
esp_ping_config_t::tos (C++ member), 152
esp_ping_config_t::ttl (C++ member), 152
ESP_PING_COUNT_INFINITE (C macro), 152
ESP_PING_DEFAULT_CONFIG (C macro), 152
esp_ping_delete_session (C++ function), 150
esp_ping_get_profile (C++ function), 151
esp_ping_handle_t (C++ type), 152
esp_ping_new_session (C++ function), 150
esp_ping_profile_t (C++ enum), 152
esp_ping_profile_t::ESP_PING_PROF_DURATION (C++ enumerator), 153
esp_ping_profile_t::ESP_PING_PROF_IPADDR (C++ enumerator), 153
esp_ping_profile_t::ESP_PING_PROF_REPLY (C++ enumerator), 153
esp_ping_profile_t::ESP_PING_PROF_REQUEST (C++ enumerator), 153
esp_ping_profile_t::ESP_PING_PROF_SEQNO (C++ enumerator), 152
esp_ping_profile_t::ESP_PING_PROF_SIZE (C++ enumerator), 153
esp_ping_profile_t::ESP_PING_PROF_TIMEGAP (C++ enumerator), 153
esp_ping_profile_t::ESP_PING_PROF_TOS (C++ enumerator), 153
esp_ping_profile_t::ESP_PING_PROF_TTL (C++ enumerator), 153
esp_ping_start (C++ function), 151
esp_ping_stop (C++ function), 151
esp_pm_config_esp32_t (C++ type), 1672
esp_pm_config_esp32c2_t (C++ type), 1672
esp_pm_config_esp32c3_t (C++ type), 1672
esp_pm_config_esp32c6_t (C++ type), 1672
esp_pm_config_esp32s2_t (C++ type), 1672
esp_pm_config_esp32s3_t (C++ type), 1672
esp_pm_config_t (C++ struct), 1672
esp_pm_config_t::light_sleep_enable (C++ member), 1672
esp_pm_config_t::max_freq_mhz (C++ member), 1672
esp_pm_config_t::min_freq_mhz (C++ member), 1672
esp_pm_configure (C++ function), 1670
esp_pm_dump_locks (C++ function), 1671
esp_pm_get_configuration (C++ function), 1670
esp_pm_lock_acquire (C++ function), 1671
esp_pm_lock_create (C++ function), 1670
esp_pm_lock_delete (C++ function), 1671
esp_pm_lock_handle_t (C++ type), 1672
esp_pm_lock_release (C++ function), 1671
esp_pm_lock_type_t (C++ enum), 1672
esp_pm_lock_type_t::ESP_PM_APB_FREQ_MAX (C++ enumerator), 1672
esp_pm_lock_type_t::ESP_PM_CPU_FREQ_MAX (C++ enumerator), 1672
esp_pm_lock_type_t::ESP_PM_LOCK_MAX (C++ enumerator), 1673
esp_pm_lock_type_t::ESP_PM_NO_LIGHT_SLEEP (C++ enumerator), 1673
esp_pthread_cfg_t (C++ struct), 1677
esp_pthread_cfg_t::inherit_cfg (C++ member), 1677
esp_pthread_cfg_t::pin_to_core (C++ member), 1678
esp_pthread_cfg_t::prio (C++ member), 1677
esp_pthread_cfg_t::stack_alloc_caps (C++ member), 1678
esp_pthread_cfg_t::stack_size (C++ member), 1677
esp_pthread_cfg_t::thread_name (C++ member), 1677
esp_pthread_get_cfg (C++ function), 1677
esp_pthread_get_default_config (C++ function), 1676
esp_pthread_init (C++ function), 1677
esp_pthread_set_cfg (C++ function), 1676
esp_random (C++ function), 1679
esp_read_mac (C++ function), 1643
esp_register_freertos_idle_hook (C++ function), 1565
esp_register_freertos_idle_hook_for_cpu (C++ function), 1564
esp_register_freertos_tick_hook (C++ function), 1565
esp_register_freertos_tick_hook_for_cpu (C++ function), 1565
esp_register_shutdown_handler (C++ function), 1639
esp_reset_reason (C++ function), 1639
esp_reset_reason_t (C++ enum), 1640
esp_reset_reason_t::ESP_RST_BROWNOUT (C++ enumerator), 1641
esp_reset_reason_t::ESP_RST_CPU_LOCKUP

(C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_DEEPSLEEP (C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_EFUSE (C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_EXT (C++ *enumerator*), 1640
esp_reset_reason_t::ESP_RST_INT_WDT (C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_JTAG (C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_PANIC (C++ *enumerator*), 1640
esp_reset_reason_t::ESP_RST_POWERON (C++ *enumerator*), 1640
esp_reset_reason_t::ESP_RST_PWR_GLITCH (C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_SDIO (C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_SW (C++ *enumerator*), 1640
esp_reset_reason_t::ESP_RST_TASK_WDT (C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_UNKNOWN (C++ *enumerator*), 1640
esp_reset_reason_t::ESP_RST_USB (C++ *enumerator*), 1641
esp_reset_reason_t::ESP_RST_WDT (C++ *enumerator*), 1641
esp_restart (C++ *function*), 1639
ESP_RETURN_ON_ERROR (C *macro*), 1405
ESP_RETURN_ON_ERROR_ISR (C *macro*), 1405
ESP_RETURN_ON_FALSE (C *macro*), 1406
ESP_RETURN_ON_FALSE_ISR (C *macro*), 1406
ESP_RETURN_VOID_ON_ERROR (C *macro*), 1405
ESP_RETURN_VOID_ON_ERROR_ISR (C *macro*), 1405
ESP_RETURN_VOID_ON_FALSE (C *macro*), 1406
ESP_RETURN_VOID_ON_FALSE_ISR (C *macro*), 1406
esp_rom_delay_us (C++ *function*), 1615
esp_rom_get_cpu_ticks_per_us (C++ *function*), 1616
esp_rom_get_reset_reason (C++ *function*), 1615
esp_rom_install_channel_putc (C++ *function*), 1615
esp_rom_install_uart_printf (C++ *function*), 1615
esp_rom_printf (C++ *function*), 1615
esp_rom_route_intr_matrix (C++ *function*), 1615
esp_rom_set_cpu_ticks_per_us (C++ *function*), 1616
esp_rom_software_reset_cpu (C++ *function*), 1615
esp_rom_software_reset_system (C++ *function*), 1615
esp_secure_boot_key_digests_t (C++ *struct*), 1404
esp_secure_boot_key_digests_t::key_digests (C++ *member*), 1404
esp_secure_boot_read_key_digests (C++ *function*), 1403
esp_set_deep_sleep_wake_stub (C++ *function*), 1693
esp_set_deep_sleep_wake_stub_default_entry (C++ *function*), 1693
esp_sleep_config_gpio_isolate (C++ *function*), 1693
esp_sleep_disable_bt_wakeup (C++ *function*), 1690
esp_sleep_disable_ext1_wakeup_io (C++ *function*), 1688
esp_sleep_disable_wakeup_source (C++ *function*), 1685
esp_sleep_disable_wifi_beacon_wakeup (C++ *function*), 1690
esp_sleep_disable_wifi_wakeup (C++ *function*), 1690
esp_sleep_enable_bt_wakeup (C++ *function*), 1690
esp_sleep_enable_ext1_wakeup (C++ *function*), 1686
esp_sleep_enable_ext1_wakeup_io (C++ *function*), 1687
esp_sleep_enable_ext1_wakeup_with_level_mask (C++ *function*), 1688
esp_sleep_enable_gpio_switch (C++ *function*), 1693
esp_sleep_enable_gpio_wakeup (C++ *function*), 1689
esp_sleep_enable_timer_wakeup (C++ *function*), 1686
esp_sleep_enable_touchpad_wakeup (C++ *function*), 1686
esp_sleep_enable_uart_wakeup (C++ *function*), 1690
esp_sleep_enable_ulp_wakeup (C++ *function*), 1685
esp_sleep_enable_wifi_beacon_wakeup (C++ *function*), 1690
esp_sleep_enable_wifi_wakeup (C++ *function*), 1690
esp_sleep_ext1_wakeup_mode_t (C++ *enum*), 1694
esp_sleep_ext1_wakeup_mode_t::ESP_EXT1_WAKEUP_ALL (C++ *enumerator*), 1694
esp_sleep_ext1_wakeup_mode_t::ESP_EXT1_WAKEUP_ANY (C++ *enumerator*), 1694
esp_sleep_ext1_wakeup_mode_t::ESP_EXT1_WAKEUP_ANY (C++ *enumerator*), 1694
esp_sleep_get_ext1_wakeup_status (C++ *function*), 1691
esp_sleep_get_gpio_wakeup_status (C++ *function*), 1691

esp_sleep_get_touchpad_wakeup_status (C++ function), 1686
esp_sleep_get_wakeup_cause (C++ function), 1692
esp_sleep_is_valid_wakeup_gpio (C++ function), 1686
esp_sleep_mode_t (C++ enum), 1696
esp_sleep_mode_t::ESP_SLEEP_MODE_DEEP_SLEEP (C++ enumerator), 1696
esp_sleep_mode_t::ESP_SLEEP_MODE_LIGHT_SLEEP (C++ enumerator), 1696
esp_sleep_pd_config (C++ function), 1691
esp_sleep_pd_domain_t (C++ enum), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_CNNT (C++ enumerator), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_MAX (C++ enumerator), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_RC32K (C++ enumerator), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_RC_FAST (C++ enumerator), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_RTC_PERIPH (C++ enumerator), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_TOP (C++ enumerator), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_VDDSDIO (C++ enumerator), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_XTAL (C++ enumerator), 1694
esp_sleep_pd_domain_t::ESP_PD_DOMAIN_XTAL32K (C++ enumerator), 1694
esp_sleep_pd_option_t (C++ enum), 1695
esp_sleep_pd_option_t::ESP_PD_OPTION_AUTO (C++ enumerator), 1695
esp_sleep_pd_option_t::ESP_PD_OPTION_OFF (C++ enumerator), 1695
esp_sleep_pd_option_t::ESP_PD_OPTION_ON (C++ enumerator), 1695
ESP_SLEEP_POWER_DOWN_CPU (C macro), 1693
esp_sleep_source_t (C++ enum), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_ALL (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_BT (C++ enumerator), 1696
esp_sleep_source_t::ESP_SLEEP_WAKEUP_COCPU (C++ enumerator), 1696
esp_sleep_source_t::ESP_SLEEP_WAKEUP_COCPU_TRAP_TRIG (C++ enumerator), 1696
esp_sleep_source_t::ESP_SLEEP_WAKEUP_EXT0 (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_EXT1 (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_GPIO (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_TIMER (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_TOUCHPAD (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_UART (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_ULP (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_UNDEFINED (C++ enumerator), 1695
esp_sleep_source_t::ESP_SLEEP_WAKEUP_WIFI (C++ enumerator), 1695
esp_sleep_wakeup_cause_t (C++ type), 1693
esp_sntp_config (C++ struct), 238
esp_sntp_config::index_of_first_server (C++ member), 239
esp_sntp_config::ip_event_to_renew (C++ member), 239
esp_sntp_config::num_of_servers (C++ member), 239
esp_sntp_config::renew_servers_after_new_IP (C++ member), 239
esp_sntp_config::server_from_dhcp (C++ member), 238
esp_sntp_config::servers (C++ member), 239
esp_sntp_config::smooth_sync (C++ member), 238
esp_sntp_config::start (C++ member), 239
esp_sntp_config::sync_cb (C++ member), 239
esp_sntp_config::wait_for_sync (C++ member), 239
esp_sntp_config_t (C++ type), 239
esp_sntp_enabled (C++ function), 1723
esp_sntp_get_sync_interval (C macro), 1724
esp_sntp_get_sync_mode (C macro), 1723
esp_sntp_get_sync_status (C macro), 1723
esp_sntp_getoperatingmode (C++ function), 1723
esp_sntp_getreachability (C++ function), 1723
esp_sntp_getserver (C++ function), 1723
esp_sntp_getservername (C++ function), 1722
esp_sntp_init (C++ function), 1722
esp_sntp_operatingmode_t (C++ enum), 1724
esp_sntp_operatingmode_t::ESP_SNTP_OPMODE_LISTENONLY (C++ enumerator), 1724
esp_sntp_operatingmode_t::ESP_SNTP_OPMODE_POLL (C++ enumerator), 1724
esp_sntp_restart (C macro), 1724
ESP_SNTP_SERVER_LIST (C macro), 239
esp_sntp_set_sync_interval (C macro), 1723
esp_sntp_set_sync_mode (C macro), 1723
esp_sntp_set_sync_status (C macro), 1723
esp_sntp_set_time_sync_notification_cb (C macro), 1723
esp_sntp_setoperatingmode (C++ function), 1722
esp_sntp_setserver (C++ function), 1722
esp_sntp_setservername (C++ function), 1722
esp_sntp_stop (C++ function), 1722

esp_sntp_sync_time (*C macro*), 1723
esp_sntp_time_cb_t (*C++ type*), 239
esp_spiffs_check (*C++ function*), 1324
esp_spiffs_format (*C++ function*), 1323
esp_spiffs_gc (*C++ function*), 1324
esp_spiffs_info (*C++ function*), 1323
esp_spiffs_mounted (*C++ function*), 1323
esp_srp_exchange_proofs (*C++ function*), 1250
esp_srp_free (*C++ function*), 1248
esp_srp_gen_salt_verifier (*C++ function*), 1249
esp_srp_get_session_key (*C++ function*), 1249
esp_srp_handle_t (*C++ type*), 1250
esp_srp_init (*C++ function*), 1248
esp_srp_set_salt_verifier (*C++ function*), 1249
esp_srp_srv_pubkey (*C++ function*), 1248
esp_srp_srv_pubkey_from_salt_verifier (*C++ function*), 1249
esp_system_abort (*C++ function*), 1640
esp_systick_new_etm_alarm_event (*C++ function*), 315
esp_sysview_flush (*C++ function*), 1357
esp_sysview_heap_trace_alloc (*C++ function*), 1358
esp_sysview_heap_trace_free (*C++ function*), 1358
esp_sysview_heap_trace_start (*C++ function*), 1357
esp_sysview_heap_trace_stop (*C++ function*), 1358
esp_sysview_vprintf (*C++ function*), 1357
esp_task_wdt_add (*C++ function*), 1755
esp_task_wdt_add_user (*C++ function*), 1755
esp_task_wdt_config_t (*C++ struct*), 1757
esp_task_wdt_config_t::idle_core_mask (*C++ member*), 1757
esp_task_wdt_config_t::timeout_ms (*C++ member*), 1757
esp_task_wdt_config_t::trigger_panic (*C++ member*), 1757
esp_task_wdt_deinit (*C++ function*), 1755
esp_task_wdt_delete (*C++ function*), 1756
esp_task_wdt_delete_user (*C++ function*), 1756
esp_task_wdt_init (*C++ function*), 1754
esp_task_wdt_isr_user_handler (*C++ function*), 1756
esp_task_wdt_print_triggered_tasks (*C++ function*), 1757
esp_task_wdt_reconfigure (*C++ function*), 1755
esp_task_wdt_reset (*C++ function*), 1756
esp_task_wdt_reset_user (*C++ function*), 1756
esp_task_wdt_status (*C++ function*), 1756
esp_task_wdt_user_handle_t (*C++ type*), 1758
esp_timer_cb_t (*C++ type*), 1614
esp_timer_create (*C++ function*), 1610
esp_timer_create_args_t (*C++ struct*), 1613
esp_timer_create_args_t::arg (*C++ member*), 1614
esp_timer_create_args_t::callback (*C++ member*), 1614
esp_timer_create_args_t::dispatch_method (*C++ member*), 1614
esp_timer_create_args_t::name (*C++ member*), 1614
esp_timer_create_args_t::skip_unhandled_events (*C++ member*), 1614
esp_timer_deinit (*C++ function*), 1610
esp_timer_delete (*C++ function*), 1611
esp_timer_dispatch_t (*C++ enum*), 1614
esp_timer_dispatch_t::ESP_TIMER_ISR (*C++ enumerator*), 1614
esp_timer_dispatch_t::ESP_TIMER_MAX (*C++ enumerator*), 1614
esp_timer_dispatch_t::ESP_TIMER_TASK (*C++ enumerator*), 1614
esp_timer_dump (*C++ function*), 1612
esp_timer_early_init (*C++ function*), 1610
esp_timer_get_expiry_time (*C++ function*), 1612
esp_timer_get_next_alarm (*C++ function*), 1612
esp_timer_get_next_alarm_for_wake_up (*C++ function*), 1612
esp_timer_get_period (*C++ function*), 1612
esp_timer_get_time (*C++ function*), 1612
esp_timer_handle_t (*C++ type*), 1614
esp_timer_init (*C++ function*), 1610
esp_timer_is_active (*C++ function*), 1613
esp_timer_isr_dispatch_need_yield (*C++ function*), 1613
esp_timer_new_etm_alarm_event (*C++ function*), 1613
esp_timer_restart (*C++ function*), 1611
esp_timer_start_once (*C++ function*), 1611
esp_timer_start_periodic (*C++ function*), 1611
esp_timer_stop (*C++ function*), 1611
esp_tls_addr_family (*C++ enum*), 68
esp_tls_addr_family::ESP_TLS_AF_INET (*C++ enumerator*), 68
esp_tls_addr_family::ESP_TLS_AF_INET6 (*C++ enumerator*), 68
esp_tls_addr_family::ESP_TLS_AF_UNSPEC (*C++ enumerator*), 68
esp_tls_addr_family_t (*C++ type*), 67
esp_tls_cfg (*C++ struct*), 63
esp_tls_cfg::addr_family (*C++ member*), 65
esp_tls_cfg::alpn_protos (*C++ member*), 63
esp_tls_cfg::cacert_buf (*C++ member*), 63

esp_tls_cfg::cacert_bytes (C++ member), 63
esp_tls_cfg::cacert_pem_buf (C++ member), 63
esp_tls_cfg::cacert_pem_bytes (C++ member), 63
esp_tls_cfg::ciphersuites_list (C++ member), 65
esp_tls_cfg::clientcert_buf (C++ member), 63
esp_tls_cfg::clientcert_bytes (C++ member), 64
esp_tls_cfg::clientcert_pem_buf (C++ member), 64
esp_tls_cfg::clientcert_pem_bytes (C++ member), 64
esp_tls_cfg::clientkey_buf (C++ member), 64
esp_tls_cfg::clientkey_bytes (C++ member), 64
esp_tls_cfg::clientkey_password (C++ member), 64
esp_tls_cfg::clientkey_password_len (C++ member), 64
esp_tls_cfg::clientkey_pem_buf (C++ member), 64
esp_tls_cfg::clientkey_pem_bytes (C++ member), 64
esp_tls_cfg::common_name (C++ member), 65
esp_tls_cfg::crt_bundle_attach (C++ member), 65
esp_tls_cfg::ds_data (C++ member), 65
esp_tls_cfg::ecdsa_key_efuse_blk (C++ member), 64
esp_tls_cfg::if_name (C++ member), 65
esp_tls_cfg::is_plain_tcp (C++ member), 65
esp_tls_cfg::keep_alive_cfg (C++ member), 65
esp_tls_cfg::non_block (C++ member), 64
esp_tls_cfg::psk_hint_key (C++ member), 65
esp_tls_cfg::skip_common_name (C++ member), 65
esp_tls_cfg::timeout_ms (C++ member), 64
esp_tls_cfg::tls_version (C++ member), 65
esp_tls_cfg::use_ecdsa_peripheral (C++ member), 64
esp_tls_cfg::use_global_ca_store (C++ member), 64
esp_tls_cfg::use_secure_element (C++ member), 64
esp_tls_cfg_server (C++ struct), 65
esp_tls_cfg_server::alpn_protos (C++ member), 65
esp_tls_cfg_server::cacert_buf (C++ member), 66
esp_tls_cfg_server::cacert_bytes (C++ member), 66
esp_tls_cfg_server::cacert_pem_buf (C++ member), 66
esp_tls_cfg_server::cacert_pem_bytes (C++ member), 66
esp_tls_cfg_server::ecdsa_key_efuse_blk (C++ member), 67
esp_tls_cfg_server::servercert_buf (C++ member), 66
esp_tls_cfg_server::servercert_bytes (C++ member), 66
esp_tls_cfg_server::servercert_pem_buf (C++ member), 66
esp_tls_cfg_server::servercert_pem_bytes (C++ member), 66
esp_tls_cfg_server::serverkey_buf (C++ member), 66
esp_tls_cfg_server::serverkey_bytes (C++ member), 66
esp_tls_cfg_server::serverkey_password (C++ member), 66
esp_tls_cfg_server::serverkey_password_len (C++ member), 66
esp_tls_cfg_server::serverkey_pem_buf (C++ member), 66
esp_tls_cfg_server::serverkey_pem_bytes (C++ member), 66
esp_tls_cfg_server::use_ecdsa_peripheral (C++ member), 66
esp_tls_cfg_server::use_secure_element (C++ member), 67
esp_tls_cfg_server::userdata (C++ member), 67
esp_tls_cfg_server_session_tickets_free (C++ function), 57
esp_tls_cfg_server_session_tickets_init (C++ function), 57
esp_tls_cfg_server_t (C++ type), 67
esp_tls_cfg_t (C++ type), 67
esp_tls_conn_destroy (C++ function), 59
esp_tls_conn_http_new (C++ function), 57
esp_tls_conn_http_new_async (C++ function), 58
esp_tls_conn_http_new_sync (C++ function), 58
esp_tls_conn_new_async (C++ function), 58
esp_tls_conn_new_sync (C++ function), 57
esp_tls_conn_read (C++ function), 59
esp_tls_conn_state (C++ enum), 68
esp_tls_conn_state::ESP_TLS_CONNECTING (C++ enumerator), 68
esp_tls_conn_state::ESP_TLS_DONE (C++ enumerator), 68
esp_tls_conn_state::ESP_TLS_FAIL (C++ enumerator), 68
esp_tls_conn_state::ESP_TLS_HANDSHAKE (C++ enumerator), 68
esp_tls_conn_state::ESP_TLS_INIT (C++

enumerator), 68
esp_tls_conn_state_t (*C++ type*), 67
esp_tls_conn_write (*C++ function*), 58
ESP_TLS_ERR_SSL_TIMEOUT (*C macro*), 71
ESP_TLS_ERR_SSL_WANT_READ (*C macro*), 71
ESP_TLS_ERR_SSL_WANT_WRITE (*C macro*), 71
esp_tls_error_handle_t (*C++ type*), 71
esp_tls_error_type_t (*C++ enum*), 71
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_ESP (*C++ enumerator*), 72
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_MAX (*C++ enumerator*), 72
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_MBEDTLS (*C++ enumerator*), 71
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_MBEDTLS_CERT_FLAGS (*C++ enumerator*), 72
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_SYSTEM (*C++ enumerator*), 71
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_UNKNOWN (*C++ enumerator*), 71
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_WOLFSSL (*C++ enumerator*), 72
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_WOLFSSL_CERT_VERIFY (*C++ enumerator*), 72
esp_tls_free_global_ca_store (*C++ function*), 60
esp_tls_get_and_clear_error_type (*C++ function*), 61
esp_tls_get_and_clear_last_error (*C++ function*), 61
esp_tls_get_bytes_avail (*C++ function*), 59
esp_tls_get_ciphersuites_list (*C++ function*), 61
esp_tls_get_conn_sockfd (*C++ function*), 59
esp_tls_get_conn_state (*C++ function*), 60
esp_tls_get_error_handle (*C++ function*), 61
esp_tls_get_global_ca_store (*C++ function*), 61
esp_tls_get_ssl_context (*C++ function*), 60
esp_tls_handshake_callback (*C++ type*), 67
esp_tls_init (*C++ function*), 57
esp_tls_init_global_ca_store (*C++ function*), 60
esp_tls_last_error (*C++ struct*), 69
esp_tls_last_error::esp_tls_error_code (*C++ member*), 69
esp_tls_last_error::esp_tls_flags (*C++ member*), 69
esp_tls_last_error::last_error (*C++ member*), 69
esp_tls_last_error_t (*C++ type*), 71
esp_tls_plain_tcp_connect (*C++ function*), 62
esp_tls_proto_ver_t (*C++ enum*), 68
esp_tls_proto_ver_t::ESP_TLS_VER_ANY (*C++ enumerator*), 68
esp_tls_proto_ver_t::ESP_TLS_VER_TLS_1_2 (*C++ enumerator*), 68
esp_tls_proto_ver_t::ESP_TLS_VER_TLS_1_3 (*C++ enumerator*), 68
esp_tls_proto_ver_t::ESP_TLS_VER_TLS_MAX (*C++ enumerator*), 68
esp_tls_role (*C++ enum*), 68
esp_tls_role::ESP_TLS_CLIENT (*C++ enumerator*), 68
esp_tls_role::ESP_TLS_SERVER (*C++ enumerator*), 68
esp_tls_role_t (*C++ type*), 67
esp_tls_server_session_create (*C++ function*), 61
esp_tls_server_session_delete (*C++ function*), 62
esp_tls_set_conn_sockfd (*C++ function*), 59
esp_tls_set_conn_state (*C++ function*), 60
esp_tls_set_global_ca_store (*C++ function*), 60
esp_tls_t (*C++ type*), 67
esp_unregister_shutdown_handler (*C++ function*), 1639
esp_vfs_close (*C++ function*), 1329
esp_vfs_dev_uart_port_set_rx_line_endings (*C++ function*), 1338
esp_vfs_dev_uart_port_set_tx_line_endings (*C++ function*), 1338
esp_vfs_dev_uart_register (*C++ function*), 1338
esp_vfs_dev_uart_set_rx_line_endings (*C++ function*), 1338
esp_vfs_dev_uart_set_tx_line_endings (*C++ function*), 1338
esp_vfs_dev_uart_use_driver (*C++ function*), 1338
esp_vfs_dev_uart_use_nonblocking (*C++ function*), 1338
esp_vfs_dump_fds (*C++ function*), 1332
ESP_VFS_EVENTD_CONFIG_DEFAULT (*C macro*), 1341
esp_vfs_eventfd_config_t (*C++ struct*), 1341
esp_vfs_eventfd_config_t::max_fds (*C++ member*), 1341
esp_vfs_eventfd_register (*C++ function*), 1341
esp_vfs_eventfd_unregister (*C++ function*), 1341
esp_vfs_fat_conf_t (*C++ struct*), 1263
esp_vfs_fat_conf_t::base_path (*C++ member*), 1263
esp_vfs_fat_conf_t::fat_drive (*C++ member*), 1263
esp_vfs_fat_conf_t::max_files (*C++ member*), 1263
esp_vfs_fat_create_contiguous_file (*C++ function*), 1262
esp_vfs_fat_info (*C++ function*), 1262
esp_vfs_fat_mount_config_t (*C++ struct*), 1263

esp_vfs_fat_mount_config_t::allocation_unit_size (C++ member), 1263
esp_vfs_fat_mount_config_t::disk_status_check_enable (C++ member), 1263
esp_vfs_fat_mount_config_t::format_if_mount_failed (C++ member), 1263
esp_vfs_fat_mount_config_t::max_files (C++ member), 1263
esp_vfs_fat_mount_config_t::use_one_fat (C++ member), 1263
esp_vfs_fat_register_cfg (C++ function), 1257
esp_vfs_fat_sdcard_format (C++ function), 1259
esp_vfs_fat_sdcard_format_cfg (C++ function), 1259
esp_vfs_fat_sdcard_unmount (C++ function), 1259
esp_vfs_fat_sdmmc_mount (C++ function), 1257
esp_vfs_fat_sdmmc_mount_config_t (C++ type), 1264
esp_vfs_fat_sdmmc_unmount (C++ function), 1259
esp_vfs_fat_sdspi_mount (C++ function), 1258
esp_vfs_fat_spiflash_format_cfg_rw_wl (C++ function), 1260
esp_vfs_fat_spiflash_format_rw_wl (C++ function), 1261
esp_vfs_fat_spiflash_mount_ro (C++ function), 1261
esp_vfs_fat_spiflash_mount_rw_wl (C++ function), 1260
esp_vfs_fat_spiflash_unmount_ro (C++ function), 1262
esp_vfs_fat_spiflash_unmount_rw_wl (C++ function), 1260
esp_vfs_fat_test_contiguous_file (C++ function), 1262
esp_vfs_fat_unregister_path (C++ function), 1257
ESP_VFS_FLAG_CONTEXT_PTR (C macro), 1337
ESP_VFS_FLAG_DEFAULT (C macro), 1337
ESP_VFS_FLAG_READONLY_FS (C macro), 1337
esp_vfs_fstat (C++ function), 1329
esp_vfs_id_t (C++ type), 1338
esp_vfs_l2tap_eth_filter (C++ function), 251
esp_vfs_l2tap_intf_register (C++ function), 251
esp_vfs_l2tap_intf_unregister (C++ function), 251
esp_vfs_link (C++ function), 1329
esp_vfs_lseek (C++ function), 1329
esp_vfs_open (C++ function), 1329
ESP_VFS_PATH_MAX (C macro), 1337
esp_vfs_pread (C++ function), 1331
esp_vfs_pwrite (C++ function), 1331
esp_vfs_read (C++ function), 1329
esp_vfs_register (C++ function), 1329
esp_vfs_register_fd (C++ function), 1330
esp_vfs_register_fd_range (C++ function), 1329
esp_vfs_register_fd_with_local_fd (C++ function), 1330
esp_vfs_register_with_id (C++ function), 1330
esp_vfs_rename (C++ function), 1329
esp_vfs_select (C++ function), 1331
esp_vfs_select_sem_t (C++ struct), 1332
esp_vfs_select_sem_t::is_sem_local (C++ member), 1332
esp_vfs_select_sem_t::sem (C++ member), 1332
esp_vfs_select_triggered (C++ function), 1331
esp_vfs_select_triggered_isr (C++ function), 1331
esp_vfs_spiffs_conf_t (C++ struct), 1324
esp_vfs_spiffs_conf_t::base_path (C++ member), 1324
esp_vfs_spiffs_conf_t::format_if_mount_failed (C++ member), 1324
esp_vfs_spiffs_conf_t::max_files (C++ member), 1324
esp_vfs_spiffs_conf_t::partition_label (C++ member), 1324
esp_vfs_spiffs_register (C++ function), 1323
esp_vfs_spiffs_unregister (C++ function), 1323
esp_vfs_stat (C++ function), 1329
esp_vfs_t (C++ struct), 1332
esp_vfs_t::access (C++ member), 1335
esp_vfs_t::access_p (C++ member), 1335
esp_vfs_t::close (C++ member), 1333
esp_vfs_t::close_p (C++ member), 1333
esp_vfs_t::closedir (C++ member), 1335
esp_vfs_t::closedir_p (C++ member), 1335
esp_vfs_t::end_select (C++ member), 1337
esp_vfs_t::fcntl (C++ member), 1335
esp_vfs_t::fcntl_p (C++ member), 1335
esp_vfs_t::flags (C++ member), 1332
esp_vfs_t::fstat (C++ member), 1333
esp_vfs_t::fstat_p (C++ member), 1333
esp_vfs_t::fsync (C++ member), 1335
esp_vfs_t::fsync_p (C++ member), 1335
esp_vfs_t::ftruncate (C++ member), 1336
esp_vfs_t::ftruncate_p (C++ member), 1336
esp_vfs_t::get_socket_select_semaphore (C++ member), 1337
esp_vfs_t::ioctl (C++ member), 1335
esp_vfs_t::ioctl_p (C++ member), 1335
esp_vfs_t::link (C++ member), 1334
esp_vfs_t::link_p (C++ member), 1334

esp_vfs_t::lseek (*C++ member*), 1333
esp_vfs_t::lseek_p (*C++ member*), 1333
esp_vfs_t::mkdir (*C++ member*), 1335
esp_vfs_t::mkdir_p (*C++ member*), 1335
esp_vfs_t::open (*C++ member*), 1333
esp_vfs_t::open_p (*C++ member*), 1333
esp_vfs_t::opendir (*C++ member*), 1334
esp_vfs_t::opendir_p (*C++ member*), 1334
esp_vfs_t::pread (*C++ member*), 1333
esp_vfs_t::pread_p (*C++ member*), 1333
esp_vfs_t::pwrite (*C++ member*), 1333
esp_vfs_t::pwrite_p (*C++ member*), 1333
esp_vfs_t::read (*C++ member*), 1333
esp_vfs_t::read_p (*C++ member*), 1333
esp_vfs_t::readdir (*C++ member*), 1334
esp_vfs_t::readdir_p (*C++ member*), 1334
esp_vfs_t::readdir_r (*C++ member*), 1334
esp_vfs_t::readdir_r_p (*C++ member*), 1334
esp_vfs_t::rename (*C++ member*), 1334
esp_vfs_t::rename_p (*C++ member*), 1334
esp_vfs_t::rmdir (*C++ member*), 1335
esp_vfs_t::rmdir_p (*C++ member*), 1335
esp_vfs_t::seekdir (*C++ member*), 1334
esp_vfs_t::seekdir_p (*C++ member*), 1334
esp_vfs_t::socket_select (*C++ member*), 1337
esp_vfs_t::start_select (*C++ member*), 1337
esp_vfs_t::stat (*C++ member*), 1333
esp_vfs_t::stat_p (*C++ member*), 1333
esp_vfs_t::stop_socket_select (*C++ member*), 1337
esp_vfs_t::stop_socket_select_isr (*C++ member*), 1337
esp_vfs_t::tcdrain (*C++ member*), 1336
esp_vfs_t::tcdrain_p (*C++ member*), 1336
esp_vfs_t::tcflow (*C++ member*), 1336
esp_vfs_t::tcflow_p (*C++ member*), 1336
esp_vfs_t::tcflush (*C++ member*), 1336
esp_vfs_t::tcflush_p (*C++ member*), 1336
esp_vfs_t::tcgetattr (*C++ member*), 1336
esp_vfs_t::tcgetattr_p (*C++ member*), 1336
esp_vfs_t::tcgetsid (*C++ member*), 1336
esp_vfs_t::tcgetsid_p (*C++ member*), 1336
esp_vfs_t::tcsendbreak (*C++ member*), 1337
esp_vfs_t::tcsendbreak_p (*C++ member*), 1337
esp_vfs_t::tcsetattr (*C++ member*), 1336
esp_vfs_t::tcsetattr_p (*C++ member*), 1336
esp_vfs_t::telldir (*C++ member*), 1334
esp_vfs_t::telldir_p (*C++ member*), 1334
esp_vfs_t::truncate (*C++ member*), 1335
esp_vfs_t::truncate_p (*C++ member*), 1335
esp_vfs_t::unlink (*C++ member*), 1334
esp_vfs_t::unlink_p (*C++ member*), 1334
esp_vfs_t::utime (*C++ member*), 1336
esp_vfs_t::utime_p (*C++ member*), 1336
esp_vfs_t::write (*C++ member*), 1332
esp_vfs_t::write_p (*C++ member*), 1332
esp_vfs_unlink (*C++ function*), 1329
esp_vfs_unregister (*C++ function*), 1330
esp_vfs_unregister_fd (*C++ function*), 1330
esp_vfs_unregister_with_id (*C++ function*), 1330
esp_vfs_usb_serial_jtag_use_driver (*C++ function*), 1339
esp_vfs_usb_serial_jtag_use_nonblocking (*C++ function*), 1339
esp_vfs_utime (*C++ function*), 1329
esp_vfs_write (*C++ function*), 1329
esp_wake_deep_sleep (*C++ function*), 1692
essl_clear_intr (*C++ function*), 105
essl_get_intr (*C++ function*), 105
essl_get_intr_ena (*C++ function*), 105
essl_get_packet (*C++ function*), 104
essl_get_rx_data_size (*C++ function*), 103
essl_get_tx_buffer_num (*C++ function*), 103
essl_handle_t (*C++ type*), 106
essl_init (*C++ function*), 103
essl_read_reg (*C++ function*), 104
essl_reset_cnt (*C++ function*), 103
essl_sdio_config_t (*C++ struct*), 106
essl_sdio_config_t::card (*C++ member*), 106
essl_sdio_config_t::recv_buffer_size (*C++ member*), 106
essl_sdio_deinit_dev (*C++ function*), 106
essl_sdio_init_dev (*C++ function*), 106
essl_send_packet (*C++ function*), 103
essl_send_slave_intr (*C++ function*), 106
essl_set_intr_ena (*C++ function*), 105
essl_spi_config_t (*C++ struct*), 112
essl_spi_config_t::rx_sync_reg (*C++ member*), 112
essl_spi_config_t::spi (*C++ member*), 112
essl_spi_config_t::tx_buf_size (*C++ member*), 112
essl_spi_config_t::tx_sync_reg (*C++ member*), 112
essl_spi_deinit_dev (*C++ function*), 107
essl_spi_get_packet (*C++ function*), 107
essl_spi_init_dev (*C++ function*), 107
essl_spi_rdbuf (*C++ function*), 108
essl_spi_rdbuf_polling (*C++ function*), 109
essl_spi_rddma (*C++ function*), 110
essl_spi_rddma_done (*C++ function*), 111
essl_spi_rddma_seg (*C++ function*), 110
essl_spi_read_reg (*C++ function*), 107
essl_spi_reset_cnt (*C++ function*), 108
essl_spi_send_packet (*C++ function*), 108
essl_spi_wrbuf (*C++ function*), 109
essl_spi_wrbuf_polling (*C++ function*), 110
essl_spi_wrdma (*C++ function*), 111
essl_spi_wrdma_done (*C++ function*), 111
essl_spi_wrdma_seg (*C++ function*), 111
essl_spi_write_reg (*C++ function*), 108

essl_wait_for_ready (C++ function), 103
essl_wait_int (C++ function), 105
essl_write_reg (C++ function), 104
eTaskGetState (C++ function), 1444
eTaskState (C++ enum), 1466
eTaskState::eBlocked (C++ enumerator), 1466
eTaskState::eDeleted (C++ enumerator), 1466
eTaskState::eInvalid (C++ enumerator), 1466
eTaskState::eReady (C++ enumerator), 1466
eTaskState::eRunning (C++ enumerator), 1466
eTaskState::eSuspended (C++ enumerator), 1466
eth_checksum_t (C++ enum), 173
eth_checksum_t::ETH_CHECKSUM_HW (C++ enumerator), 173
eth_checksum_t::ETH_CHECKSUM_SW (C++ enumerator), 173
ETH_CMD_CUSTOM_MAC_CMDS_OFFSET (C macro), 182
ETH_CMD_CUSTOM_PHY_CMDS_OFFSET (C macro), 182
eth_data_interface_t (C++ enum), 172
eth_data_interface_t::EMAC_DATA_INTERFACE_MII (C++ enumerator), 172
eth_data_interface_t::EMAC_DATA_INTERFACE_RMII (C++ enumerator), 172
ETH_DEFAULT_CONFIG (C macro), 179
ETH_DEFAULT_SPI (C macro), 194
eth_duplex_t (C++ enum), 173
eth_duplex_t::ETH_DUPLEX_FULL (C++ enumerator), 173
eth_duplex_t::ETH_DUPLEX_HALF (C++ enumerator), 173
eth_esp32_emac_config_t (C++ struct), 191
eth_esp32_emac_config_t::clock_config (C++ member), 191
eth_esp32_emac_config_t::clock_config_out_in (C++ member), 191
eth_esp32_emac_config_t::dma_burst_len (C++ member), 191
eth_esp32_emac_config_t::emac_dataif_gpio (C++ member), 191
eth_esp32_emac_config_t::interface (C++ member), 191
eth_esp32_emac_config_t::intr_priority (C++ member), 191
eth_esp32_emac_config_t::smi_gpio (C++ member), 191
eth_esp32_emac_config_t::smi_mdc_gpio_num (C++ member), 191
eth_esp32_emac_config_t::smi_mdio_gpio_num (C++ member), 191
ETH_ESP32_EMAC_DEFAULT_CONFIG (C macro), 192
eth_event_t (C++ enum), 182
eth_event_t::ETHERNET_EVENT_CONNECTED (C++ enumerator), 183
eth_event_t::ETHERNET_EVENT_DISCONNECTED (C++ enumerator), 183
eth_event_t::ETHERNET_EVENT_START (C++ enumerator), 182
eth_event_t::ETHERNET_EVENT_STOP (C++ enumerator), 183
eth_link_t (C++ enum), 172
eth_link_t::ETH_LINK_DOWN (C++ enumerator), 172
eth_link_t::ETH_LINK_UP (C++ enumerator), 172
eth_mac_clock_config_t (C++ union), 188
eth_mac_clock_config_t::clock_gpio (C++ member), 189
eth_mac_clock_config_t::clock_mode (C++ member), 189
eth_mac_clock_config_t::mii (C++ member), 189
eth_mac_clock_config_t::rmii (C++ member), 189
eth_mac_config_t (C++ struct), 187
eth_mac_config_t::flags (C++ member), 188
eth_mac_config_t::rx_task_prio (C++ member), 187
eth_mac_config_t::rx_task_stack_size (C++ member), 187
eth_mac_config_t::sw_reset_timeout_ms (C++ member), 187
eth_mac_dataif_gpio_config_t (C++ union), 189
eth_mac_dataif_gpio_config_t::mii (C++ member), 189
eth_mac_dataif_gpio_config_t::rmii (C++ member), 189
ETH_MAC_DEFAULT_CONFIG (C macro), 188
eth_mac_dma_burst_len_t (C++ enum), 173
eth_mac_dma_burst_len_t::ETH_DMA_BURST_LEN_1 (C++ enumerator), 173
eth_mac_dma_burst_len_t::ETH_DMA_BURST_LEN_16 (C++ enumerator), 173
eth_mac_dma_burst_len_t::ETH_DMA_BURST_LEN_2 (C++ enumerator), 173
eth_mac_dma_burst_len_t::ETH_DMA_BURST_LEN_32 (C++ enumerator), 173
eth_mac_dma_burst_len_t::ETH_DMA_BURST_LEN_4 (C++ enumerator), 173
eth_mac_dma_burst_len_t::ETH_DMA_BURST_LEN_8 (C++ enumerator), 173
eth_mac_esp_io_cmd_t (C++ enum), 192
eth_mac_esp_io_cmd_t::ETH_MAC_ESP_CMD_CLEAR_TDES0 (C++ enumerator), 192
eth_mac_esp_io_cmd_t::ETH_MAC_ESP_CMD_PTP_ENABLE (C++ enumerator), 192
eth_mac_esp_io_cmd_t::ETH_MAC_ESP_CMD_SET_TDES0_C (C++ enumerator), 192
ETH_MAC_FLAG_PIN_TO_CORE (C macro), 188
ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE (C macro), 188
eth_mac_mii_gpio_config_t (C++ struct), 189

eth_mac_mii_gpio_config_t::col_in_num (C++ member), 190
eth_mac_mii_gpio_config_t::crs_in_num (C++ member), 190
eth_mac_mii_gpio_config_t::rx_clk_num (C++ member), 190
eth_mac_mii_gpio_config_t::rx_dv_num (C++ member), 190
eth_mac_mii_gpio_config_t::rx_er_num (C++ member), 190
eth_mac_mii_gpio_config_t::rxd0_num (C++ member), 190
eth_mac_mii_gpio_config_t::rxd1_num (C++ member), 190
eth_mac_mii_gpio_config_t::rxd2_num (C++ member), 190
eth_mac_mii_gpio_config_t::rxd3_num (C++ member), 190
eth_mac_mii_gpio_config_t::tx_clk_num (C++ member), 189
eth_mac_mii_gpio_config_t::tx_en_num (C++ member), 189
eth_mac_mii_gpio_config_t::tx_er_num (C++ member), 190
eth_mac_mii_gpio_config_t::txd0_num (C++ member), 189
eth_mac_mii_gpio_config_t::txd1_num (C++ member), 190
eth_mac_mii_gpio_config_t::txd2_num (C++ member), 190
eth_mac_mii_gpio_config_t::txd3_num (C++ member), 190
eth_mac_rmii_gpio_config_t (C++ struct), 190
eth_mac_rmii_gpio_config_t::crs_dv_num (C++ member), 191
eth_mac_rmii_gpio_config_t::rxd0_num (C++ member), 191
eth_mac_rmii_gpio_config_t::rxd1_num (C++ member), 191
eth_mac_rmii_gpio_config_t::tx_en_num (C++ member), 190
eth_mac_rmii_gpio_config_t::txd0_num (C++ member), 191
eth_mac_rmii_gpio_config_t::txd1_num (C++ member), 191
eth_phy_autoneg_cmd_t (C++ enum), 199
eth_phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_DIS (C++ enumerator), 199
eth_phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_EN (C++ enumerator), 199
eth_phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_G_STAT (C++ enumerator), 199
eth_phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_RESTART (C++ enumerator), 199
eth_phy_config_t (C++ struct), 198
eth_phy_config_t::autonego_timeout_ms (C++ member), 199
eth_phy_config_t::phy_addr (C++ member), 199
eth_phy_config_t::reset_gpio_num (C++ member), 199
eth_phy_config_t::reset_timeout_ms (C++ member), 199
ETH_PHY_DEFAULT_CONFIG (C macro), 199
eth_speed_t (C++ enum), 172
eth_speed_t::ETH_SPEED_100M (C++ enumerator), 173
eth_speed_t::ETH_SPEED_10M (C++ enumerator), 172
eth_speed_t::ETH_SPEED_MAX (C++ enumerator), 173
eth_spi_custom_driver_config_t (C++ struct), 193
eth_spi_custom_driver_config_t::config (C++ member), 193
eth_spi_custom_driver_config_t::deinit (C++ member), 193
eth_spi_custom_driver_config_t::init (C++ member), 193
eth_spi_custom_driver_config_t::read (C++ member), 194
eth_spi_custom_driver_config_t::write (C++ member), 194
ETS_INTERNAL_INTR_SOURCE_OFF (C macro), 1628
ETS_INTERNAL_PROFILING_INTR_SOURCE (C macro), 1628
ETS_INTERNAL_SW0_INTR_SOURCE (C macro), 1628
ETS_INTERNAL_SW1_INTR_SOURCE (C macro), 1628
ETS_INTERNAL_TIMER0_INTR_SOURCE (C macro), 1627
ETS_INTERNAL_TIMER1_INTR_SOURCE (C macro), 1628
ETS_INTERNAL_TIMER2_INTR_SOURCE (C macro), 1628
ETS_INTERNAL_UNUSED_INTR_SOURCE (C macro), 1628
EventBits_t (C++ type), 1527
eventfd (C++ function), 1341
EventGroupHandle_t (C++ type), 1527

F

ff_diskio_impl_t (C++ struct), 1254
ff_diskio_impl_t::init (C++ member), 1254
ff_diskio_impl_t::ioctl (C++ member), 1255
ff_diskio_impl_t::read (C++ member), 1254
ff_diskio_impl_t::status (C++ member), 1254
ff_diskio_impl_t::write (C++ member), 1254
ff_diskio_register (C++ function), 1254

ff_diskio_register_raw_partition (C++ function), 1255
ff_diskio_register_sdmmc (C++ function), 1255
ff_diskio_register_wl_partition (C++ function), 1255

G

gpio_config (C++ function), 319
gpio_config_t (C++ struct), 326
gpio_config_t::hys_ctrl_mode (C++ member), 326
gpio_config_t::intr_type (C++ member), 326
gpio_config_t::mode (C++ member), 326
gpio_config_t::pin_bit_mask (C++ member), 326
gpio_config_t::pull_down_en (C++ member), 326
gpio_config_t::pull_up_en (C++ member), 326
gpio_deep_sleep_wakeup_disable (C++ function), 326
gpio_deep_sleep_wakeup_enable (C++ function), 325
gpio_del_glitch_filter (C++ function), 338
gpio_drive_cap_t (C++ enum), 331
gpio_drive_cap_t::GPIO_DRIVE_CAP_0 (C++ enumerator), 331
gpio_drive_cap_t::GPIO_DRIVE_CAP_1 (C++ enumerator), 331
gpio_drive_cap_t::GPIO_DRIVE_CAP_2 (C++ enumerator), 331
gpio_drive_cap_t::GPIO_DRIVE_CAP_3 (C++ enumerator), 331
gpio_drive_cap_t::GPIO_DRIVE_CAP_DEFAULT (C++ enumerator), 331
gpio_drive_cap_t::GPIO_DRIVE_CAP_MAX (C++ enumerator), 331
gpio_dump_io_configuration (C++ function), 326
gpio_etm_event_bind_gpio (C++ function), 312
gpio_etm_event_config_t (C++ struct), 313
gpio_etm_event_config_t::edge (C++ member), 314
gpio_etm_event_config_t::edges (C++ member), 314
gpio_etm_event_edge_t (C++ enum), 314
gpio_etm_event_edge_t::GPIO_ETM_EVENT_EDGE_ANY (C++ enumerator), 314
gpio_etm_event_edge_t::GPIO_ETM_EVENT_EDGE_NEG (C++ enumerator), 314
gpio_etm_event_edge_t::GPIO_ETM_EVENT_EDGE_POS (C++ enumerator), 314
GPIO_ETM_EVENT_EDGE_TYPES (C macro), 314
gpio_etm_task_action_t (C++ enum), 314
gpio_etm_task_action_t::GPIO_ETM_TASK_ACTION_CLR (C++ enumerator), 315
gpio_etm_task_action_t::GPIO_ETM_TASK_ACTION_SET (C++ enumerator), 315
gpio_etm_task_action_t::GPIO_ETM_TASK_ACTION_TOG (C++ enumerator), 315
GPIO_ETM_TASK_ACTION_TYPES (C macro), 314
gpio_etm_task_add_gpio (C++ function), 313
gpio_etm_task_config_t (C++ struct), 314
gpio_etm_task_config_t::action (C++ member), 314
gpio_etm_task_config_t::actions (C++ member), 314
gpio_etm_task_rm_gpio (C++ function), 313
gpio_flex_glitch_filter_config_t (C++ struct), 339
gpio_flex_glitch_filter_config_t::clk_src (C++ member), 339
gpio_flex_glitch_filter_config_t::gpio_num (C++ member), 339
gpio_flex_glitch_filter_config_t::window_thres_ns (C++ member), 339
gpio_flex_glitch_filter_config_t::window_width_ns (C++ member), 339
gpio_force_hold_all (C++ function), 324
gpio_force_unhold_all (C++ function), 324
gpio_get_drive_capability (C++ function), 323
gpio_get_level (C++ function), 320
gpio_glitch_filter_disable (C++ function), 338
gpio_glitch_filter_enable (C++ function), 338
gpio_glitch_filter_handle_t (C++ type), 339
gpio_hold_dis (C++ function), 324
gpio_hold_en (C++ function), 323
gpio_hys_ctrl_mode_t (C++ enum), 331
gpio_hys_ctrl_mode_t::GPIO_HYS_SOFT_DISABLE (C++ enumerator), 332
gpio_hys_ctrl_mode_t::GPIO_HYS_SOFT_ENABLE (C++ enumerator), 332
gpio_install_isr_service (C++ function), 322
gpio_int_type_t (C++ enum), 329
gpio_int_type_t::GPIO_INTR_ANYEDGE (C++ enumerator), 330
gpio_int_type_t::GPIO_INTR_DISABLE (C++ enumerator), 330
gpio_int_type_t::GPIO_INTR_HIGH_LEVEL (C++ enumerator), 330
gpio_int_type_t::GPIO_INTR_LOW_LEVEL (C++ enumerator), 330
gpio_int_type_t::GPIO_INTR_MAX (C++ enumerator), 330
gpio_int_type_t::GPIO_INTR_NEGEDGE (C++ enumerator), 330
gpio_int_type_t::GPIO_INTR_POSEDGE

(C++ *enumerator*), 330
gpio_intr_disable (C++ *function*), 320
gpio_intr_enable (C++ *function*), 319
gpio_iomux_in (C++ *function*), 324
gpio_iomux_out (C++ *function*), 324
GPIO_IS_DEEP_SLEEP_WAKEUP_VALID_GPIO (*C macro*), 327
GPIO_IS_VALID_DIGITAL_IO_PAD (*C macro*), 326
GPIO_IS_VALID_GPIO (*C macro*), 326
GPIO_IS_VALID_OUTPUT_GPIO (*C macro*), 326
gpio_isr_handle_t (C++ *type*), 327
gpio_isr_handler_add (C++ *function*), 322
gpio_isr_handler_remove (C++ *function*), 323
gpio_isr_register (C++ *function*), 321
gpio_isr_t (C++ *type*), 327
gpio_mode_t (C++ *enum*), 330
gpio_mode_t::GPIO_MODE_DISABLE (C++ *enumerator*), 330
gpio_mode_t::GPIO_MODE_INPUT (C++ *enumerator*), 330
gpio_mode_t::GPIO_MODE_INPUT_OUTPUT (C++ *enumerator*), 330
gpio_mode_t::GPIO_MODE_INPUT_OUTPUT_OD (C++ *enumerator*), 330
gpio_mode_t::GPIO_MODE_OUTPUT (C++ *enumerator*), 330
gpio_mode_t::GPIO_MODE_OUTPUT_OD (C++ *enumerator*), 330
gpio_new_etm_event (C++ *function*), 311
gpio_new_etm_task (C++ *function*), 312
gpio_new_flex_glitch_filter (C++ *function*), 337
gpio_new_pin_glitch_filter (C++ *function*), 337
GPIO_PIN_COUNT (*C macro*), 326
gpio_pin_glitch_filter_config_t (C++ *struct*), 339
gpio_pin_glitch_filter_config_t::clk_src (C++ *member*), 339
gpio_pin_glitch_filter_config_t::gpio_num (C++ *member*), 339
GPIO_PIN_REG_0 (*C macro*), 327
GPIO_PIN_REG_1 (*C macro*), 327
GPIO_PIN_REG_10 (*C macro*), 327
GPIO_PIN_REG_11 (*C macro*), 327
GPIO_PIN_REG_12 (*C macro*), 327
GPIO_PIN_REG_13 (*C macro*), 327
GPIO_PIN_REG_14 (*C macro*), 327
GPIO_PIN_REG_15 (*C macro*), 328
GPIO_PIN_REG_16 (*C macro*), 328
GPIO_PIN_REG_17 (*C macro*), 328
GPIO_PIN_REG_18 (*C macro*), 328
GPIO_PIN_REG_19 (*C macro*), 328
GPIO_PIN_REG_2 (*C macro*), 327
GPIO_PIN_REG_20 (*C macro*), 328
GPIO_PIN_REG_21 (*C macro*), 328
GPIO_PIN_REG_22 (*C macro*), 328
GPIO_PIN_REG_23 (*C macro*), 328
GPIO_PIN_REG_24 (*C macro*), 328
GPIO_PIN_REG_25 (*C macro*), 328
GPIO_PIN_REG_26 (*C macro*), 328
GPIO_PIN_REG_27 (*C macro*), 328
GPIO_PIN_REG_28 (*C macro*), 328
GPIO_PIN_REG_29 (*C macro*), 328
GPIO_PIN_REG_3 (*C macro*), 327
GPIO_PIN_REG_30 (*C macro*), 328
GPIO_PIN_REG_31 (*C macro*), 328
GPIO_PIN_REG_32 (*C macro*), 328
GPIO_PIN_REG_33 (*C macro*), 328
GPIO_PIN_REG_34 (*C macro*), 328
GPIO_PIN_REG_35 (*C macro*), 328
GPIO_PIN_REG_36 (*C macro*), 328
GPIO_PIN_REG_37 (*C macro*), 328
GPIO_PIN_REG_38 (*C macro*), 329
GPIO_PIN_REG_39 (*C macro*), 329
GPIO_PIN_REG_4 (*C macro*), 327
GPIO_PIN_REG_40 (*C macro*), 329
GPIO_PIN_REG_41 (*C macro*), 329
GPIO_PIN_REG_42 (*C macro*), 329
GPIO_PIN_REG_43 (*C macro*), 329
GPIO_PIN_REG_44 (*C macro*), 329
GPIO_PIN_REG_45 (*C macro*), 329
GPIO_PIN_REG_46 (*C macro*), 329
GPIO_PIN_REG_47 (*C macro*), 329
GPIO_PIN_REG_48 (*C macro*), 329
GPIO_PIN_REG_49 (*C macro*), 329
GPIO_PIN_REG_5 (*C macro*), 327
GPIO_PIN_REG_50 (*C macro*), 329
GPIO_PIN_REG_51 (*C macro*), 329
GPIO_PIN_REG_52 (*C macro*), 329
GPIO_PIN_REG_53 (*C macro*), 329
GPIO_PIN_REG_54 (*C macro*), 329
GPIO_PIN_REG_6 (*C macro*), 327
GPIO_PIN_REG_7 (*C macro*), 327
GPIO_PIN_REG_8 (*C macro*), 327
GPIO_PIN_REG_9 (*C macro*), 327
gpio_port_t (C++ *enum*), 329
gpio_port_t::GPIO_PORT_0 (C++ *enumerator*), 329
gpio_port_t::GPIO_PORT_MAX (C++ *enumerator*), 329
gpio_pull_mode_t (C++ *enum*), 331
gpio_pull_mode_t::GPIO_FLOATING (C++ *enumerator*), 331
gpio_pull_mode_t::GPIO_PULLDOWN_ONLY (C++ *enumerator*), 331
gpio_pull_mode_t::GPIO_PULLUP_ONLY (C++ *enumerator*), 331
gpio_pull_mode_t::GPIO_PULLUP_PULLDOWN (C++ *enumerator*), 331
gpio_pulldown_dis (C++ *function*), 322
gpio_pulldown_en (C++ *function*), 322
gpio_pulldown_t (C++ *enum*), 331
gpio_pulldown_t::GPIO_PULLDOWN_DISABLE (C++ *enumerator*), 331

gpio_pulldown_t::GPIO_PULLDOWN_ENABLE (C++ enumerator), 331
gpio_pullup_dis (C++ function), 322
gpio_pullup_en (C++ function), 322
gpio_pullup_t (C++ enum), 330
gpio_pullup_t::GPIO_PULLUP_DISABLE (C++ enumerator), 330
gpio_pullup_t::GPIO_PULLUP_ENABLE (C++ enumerator), 330
gpio_reset_pin (C++ function), 319
gpio_set_direction (C++ function), 320
gpio_set_drive_capability (C++ function), 323
gpio_set_intr_type (C++ function), 319
gpio_set_level (C++ function), 320
gpio_set_pull_mode (C++ function), 321
gpio_sleep_sel_dis (C++ function), 325
gpio_sleep_sel_en (C++ function), 325
gpio_sleep_set_direction (C++ function), 325
gpio_sleep_set_pull_mode (C++ function), 325
gpio_uninstall_isr_service (C++ function), 322
gpio_wakeup_disable (C++ function), 321
gpio_wakeup_enable (C++ function), 321
gptimer_alarm_cb_t (C++ type), 353
gptimer_alarm_config_t (C++ struct), 351
gptimer_alarm_config_t::alarm_count (C++ member), 351
gptimer_alarm_config_t::auto_reload_on_alarm (C++ member), 351
gptimer_alarm_config_t::flags (C++ member), 351
gptimer_alarm_config_t::reload_count (C++ member), 351
gptimer_alarm_event_data_t (C++ struct), 353
gptimer_alarm_event_data_t::alarm_value (C++ member), 353
gptimer_alarm_event_data_t::count_value (C++ member), 353
gptimer_clock_source_t (C++ type), 353
gptimer_config_t (C++ struct), 350
gptimer_config_t::backup_before_sleep (C++ member), 350
gptimer_config_t::clk_src (C++ member), 350
gptimer_config_t::direction (C++ member), 350
gptimer_config_t::flags (C++ member), 351
gptimer_config_t::intr_priority (C++ member), 350
gptimer_config_t::intr_shared (C++ member), 350
gptimer_config_t::resolution_hz (C++ member), 350
gptimer_count_direction_t (C++ enum), 354
gptimer_count_direction_t::GPTIMER_COUNT_DOWN (C++ enumerator), 354
gptimer_count_direction_t::GPTIMER_COUNT_UP (C++ enumerator), 354
gptimer_del_timer (C++ function), 345
gptimer_disable (C++ function), 349
gptimer_enable (C++ function), 348
gptimer_etm_event_config_t (C++ struct), 352
gptimer_etm_event_config_t::event_type (C++ member), 352
gptimer_etm_event_type_t (C++ enum), 354
gptimer_etm_event_type_t::GPTIMER_ETM_EVENT_ALARM (C++ enumerator), 354
gptimer_etm_event_type_t::GPTIMER_ETM_EVENT_MAX (C++ enumerator), 354
gptimer_etm_task_config_t (C++ struct), 352
gptimer_etm_task_config_t::task_type (C++ member), 352
gptimer_etm_task_type_t (C++ enum), 354
gptimer_etm_task_type_t::GPTIMER_ETM_TASK_CAPTURE (C++ enumerator), 354
gptimer_etm_task_type_t::GPTIMER_ETM_TASK_EN_ALAR (C++ enumerator), 354
gptimer_etm_task_type_t::GPTIMER_ETM_TASK_MAX (C++ enumerator), 354
gptimer_etm_task_type_t::GPTIMER_ETM_TASK_RELOAD (C++ enumerator), 354
gptimer_etm_task_type_t::GPTIMER_ETM_TASK_START_C (C++ enumerator), 354
gptimer_etm_task_type_t::GPTIMER_ETM_TASK_STOP_CO (C++ enumerator), 354
gptimer_event_callbacks_t (C++ struct), 351
gptimer_event_callbacks_t::on_alarm (C++ member), 351
gptimer_get_captured_count (C++ function), 347
gptimer_get_raw_count (C++ function), 346
gptimer_get_resolution (C++ function), 347
gptimer_handle_t (C++ type), 353
gptimer_new_etm_event (C++ function), 352
gptimer_new_etm_task (C++ function), 352
gptimer_new_timer (C++ function), 345
gptimer_register_event_callbacks (C++ function), 347
gptimer_set_alarm_action (C++ function), 348
gptimer_set_raw_count (C++ function), 346
gptimer_start (C++ function), 349
gptimer_stop (C++ function), 350

H

heap_caps_add_region (C++ function), 1582
heap_caps_add_region_with_caps (C++ function), 1583
heap_caps_aligned_alloc (C++ function), 1575

heap_caps_aligned_calloc (C++ function), 1575
heap_caps_aligned_free (C++ function), 1575
heap_caps_calloc (C++ function), 1576
heap_caps_calloc_prefer (C++ function), 1579
heap_caps_check_integrity (C++ function), 1577
heap_caps_check_integrity_addr (C++ function), 1578
heap_caps_check_integrity_all (C++ function), 1577
heap_caps_dump (C++ function), 1579
heap_caps_dump_all (C++ function), 1579
heap_caps_enable_nonos_stack_heaps (C++ function), 1582
heap_caps_free (C++ function), 1575
heap_caps_get_allocated_size (C++ function), 1579
heap_caps_get_free_size (C++ function), 1576
heap_caps_get_info (C++ function), 1577
heap_caps_get_largest_free_block (C++ function), 1576
heap_caps_get_minimum_free_size (C++ function), 1576
heap_caps_get_total_size (C++ function), 1576
heap_caps_init (C++ function), 1582
heap_caps_malloc (C++ function), 1575
heap_caps_malloc_extmem_enable (C++ function), 1578
heap_caps_malloc_prefer (C++ function), 1578
heap_caps_monitor_local_minimum_free_size_start (C++ function), 1577
heap_caps_monitor_local_minimum_free_size_stop (C++ function), 1577
heap_caps_print_heap_info (C++ function), 1577
heap_caps_realloc (C++ function), 1575
heap_caps_realloc_prefer (C++ function), 1578
heap_caps_register_failed_alloc_callback (C++ function), 1574
heap_caps_walk (C++ function), 1579
heap_caps_walk_all (C++ function), 1579
heap_caps_walker_cb_t (C++ type), 1582
HEAP_IRAM_ATTR (C macro), 1580
heap_trace_dump (C++ function), 1607
heap_trace_dump_caps (C++ function), 1607
heap_trace_get (C++ function), 1607
heap_trace_get_count (C++ function), 1607
heap_trace_init_standalone (C++ function), 1606
heap_trace_init_tohost (C++ function), 1606
heap_trace_mode_t (C++ enum), 1609
heap_trace_mode_t::HEAP_TRACE_ALL (C++ enumerator), 1609
heap_trace_mode_t::HEAP_TRACE_LEAKS (C++ enumerator), 1609
heap_trace_record_t (C++ struct), 1608
heap_trace_record_t (C++ type), 1609
heap_trace_record_t::address (C++ member), 1608
heap_trace_record_t::alloced_by (C++ member), 1608
heap_trace_record_t::ccount (C++ member), 1608
heap_trace_record_t::freed_by (C++ member), 1608
heap_trace_record_t::size (C++ member), 1608
heap_trace_resume (C++ function), 1607
heap_trace_start (C++ function), 1606
heap_trace_stop (C++ function), 1607
heap_trace_summary (C++ function), 1608
heap_trace_summary_t (C++ struct), 1608
heap_trace_summary_t::capacity (C++ member), 1608
heap_trace_summary_t::count (C++ member), 1608
heap_trace_summary_t::has_overflowed (C++ member), 1609
heap_trace_summary_t::high_water_mark (C++ member), 1609
heap_trace_summary_t::mode (C++ member), 1608
heap_trace_summary_t::total_allocations (C++ member), 1608
heap_trace_summary_t::total_frees (C++ member), 1608
hmac_key_id_t (C++ enum), 363
hmac_key_id_t::HMAC_KEY0 (C++ enumerator), 363
hmac_key_id_t::HMAC_KEY1 (C++ enumerator), 363
hmac_key_id_t::HMAC_KEY2 (C++ enumerator), 363
hmac_key_id_t::HMAC_KEY3 (C++ enumerator), 363
hmac_key_id_t::HMAC_KEY4 (C++ enumerator), 363
hmac_key_id_t::HMAC_KEY5 (C++ enumerator), 363
hmac_key_id_t::HMAC_KEY_MAX (C++ enumerator), 363
HTTP_ANY (C macro), 138
http_client_init_cb_t (C++ type), 1414
http_event_handle_cb (C++ type), 86
HTTPD_200 (C macro), 138
HTTPD_204 (C macro), 138
HTTPD_207 (C macro), 138
HTTPD_400 (C macro), 138
HTTPD_404 (C macro), 138
HTTPD_408 (C macro), 138

HTTPD_500 (*C macro*), 138
httpd_close_func_t (*C++ type*), 140
httpd_config (*C++ struct*), 134
httpd_config::backlog_conn (*C++ member*), 135
httpd_config::close_fn (*C++ member*), 136
httpd_config::core_id (*C++ member*), 134
httpd_config::ctrl_port (*C++ member*), 134
httpd_config::enable_so_linger (*C++ member*), 135
httpd_config::global_transport_ctx (*C++ member*), 135
httpd_config::global_transport_ctx_free_fn (*C++ member*), 135
httpd_config::global_user_ctx (*C++ member*), 135
httpd_config::global_user_ctx_free_fn (*C++ member*), 135
httpd_config::keep_alive_count (*C++ member*), 136
httpd_config::keep_alive_enable (*C++ member*), 135
httpd_config::keep_alive_idle (*C++ member*), 135
httpd_config::keep_alive_interval (*C++ member*), 136
httpd_config::linger_timeout (*C++ member*), 135
httpd_config::lru_purge_enable (*C++ member*), 135
httpd_config::max_open_sockets (*C++ member*), 134
httpd_config::max_resp_headers (*C++ member*), 135
httpd_config::max_uri_handlers (*C++ member*), 135
httpd_config::open_fn (*C++ member*), 136
httpd_config::recv_wait_timeout (*C++ member*), 135
httpd_config::send_wait_timeout (*C++ member*), 135
httpd_config::server_port (*C++ member*), 134
httpd_config::stack_size (*C++ member*), 134
httpd_config::task_caps (*C++ member*), 134
httpd_config::task_priority (*C++ member*), 134
httpd_config::uri_match_fn (*C++ member*), 136
httpd_config_t (*C++ type*), 140
HTTPD_DEFAULT_CONFIG (*C macro*), 139
httpd_err_code_t (*C++ enum*), 142
httpd_err_code_t::HTTPD_400_BAD_REQUEST (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_401_UNAUTHORIZED (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_403_FORBIDDEN (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_404_NOT_FOUND (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_405_METHOD_NOT_ALLOWED (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_408_REQ_TIMEOUT (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_411_LENGTH_REQUIRED (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_414_URI_TOO_LONG (*C++ enumerator*), 143
httpd_err_code_t::HTTPD_431_REQ_HDR_FIELDS_TOO_LA (*C++ enumerator*), 143
httpd_err_code_t::HTTPD_500_INTERNAL_SERVER_ERROR (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_501_METHOD_NOT_IMPLEMENTE (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_505_VERSION_NOT_SUPPORTED (*C++ enumerator*), 142
httpd_err_code_t::HTTPD_ERR_CODE_MAX (*C++ enumerator*), 143
httpd_err_handler_func_t (*C++ type*), 141
httpd_free_ctx_fn_t (*C++ type*), 139
httpd_get_client_list (*C++ function*), 133
httpd_get_global_transport_ctx (*C++ function*), 133
httpd_get_global_user_ctx (*C++ function*), 132
httpd_handle_t (*C++ type*), 139
HTTPD_MAX_REQ_HDR_LEN (*C macro*), 138
HTTPD_MAX_URI_LEN (*C macro*), 138
httpd_method_t (*C++ type*), 139
httpd_open_func_t (*C++ type*), 140
httpd_pending_func_t (*C++ type*), 141
httpd_query_key_value (*C++ function*), 124
httpd_queue_work (*C++ function*), 131
httpd_recv_func_t (*C++ type*), 141
httpd_register_err_handler (*C++ function*), 131
httpd_register_uri_handler (*C++ function*), 119
httpd_req (*C++ struct*), 136
httpd_req::aux (*C++ member*), 137
httpd_req::content_len (*C++ member*), 137
httpd_req::free_ctx (*C++ member*), 137
httpd_req::handle (*C++ member*), 136
httpd_req::ignore_sess_ctx_changes (*C++ member*), 137
httpd_req::method (*C++ member*), 136
httpd_req::sess_ctx (*C++ member*), 137
httpd_req::uri (*C++ member*), 137
httpd_req::user_ctx (*C++ member*), 137
httpd_req_async_handler_begin (*C++ function*), 121
httpd_req_async_handler_complete (*C++ function*), 121
httpd_req_get_cookie_val (*C++ function*), 124

httpd_req_get_hdr_value_len (C++ function), 123
httpd_req_get_hdr_value_str (C++ function), 123
httpd_req_get_url_query_len (C++ function), 123
httpd_req_get_url_query_str (C++ function), 124
httpd_req_recv (C++ function), 122
httpd_req_t (C++ type), 140
httpd_req_to_sockfd (C++ function), 122
httpd_resp_send (C++ function), 125
httpd_resp_send_404 (C++ function), 129
httpd_resp_send_408 (C++ function), 129
httpd_resp_send_500 (C++ function), 129
httpd_resp_send_chunk (C++ function), 126
httpd_resp_send_custom_err (C++ function), 128
httpd_resp_send_err (C++ function), 128
httpd_resp_sendstr (C++ function), 126
httpd_resp_sendstr_chunk (C++ function), 126
httpd_resp_set_hdr (C++ function), 127
httpd_resp_set_status (C++ function), 127
httpd_resp_set_type (C++ function), 127
HTTPD_RESP_USE_STRLEN (C macro), 139
httpd_send (C++ function), 130
httpd_send_func_t (C++ type), 140
httpd_sess_get_ctx (C++ function), 132
httpd_sess_get_transport_ctx (C++ function), 132
httpd_sess_set_ctx (C++ function), 132
httpd_sess_set_pending_override (C++ function), 121
httpd_sess_set_recv_override (C++ function), 120
httpd_sess_set_send_override (C++ function), 120
httpd_sess_set_transport_ctx (C++ function), 132
httpd_sess_trigger_close (C++ function), 133
httpd_sess_update_lru_counter (C++ function), 133
HTTPD_SOCK_ERR_FAIL (C macro), 138
HTTPD_SOCK_ERR_INVALID (C macro), 138
HTTPD_SOCK_ERR_TIMEOUT (C macro), 138
httpd_socket_recv (C++ function), 131
httpd_socket_send (C++ function), 130
httpd_ssl_config (C++ struct), 145
httpd_ssl_config::alpn_protos (C++ member), 146
httpd_ssl_config::cacert_len (C++ member), 146
httpd_ssl_config::cacert_pem (C++ member), 146
httpd_ssl_config::cert_select_cb (C++ member), 146
httpd_ssl_config::ecdsa_key_efuse_blk (C++ member), 146
httpd_ssl_config::httpd (C++ member), 145
httpd_ssl_config::port_insecure (C++ member), 146
httpd_ssl_config::port_secure (C++ member), 146
httpd_ssl_config::prvtkey_len (C++ member), 146
httpd_ssl_config::prvtkey_pem (C++ member), 146
httpd_ssl_config::servercert (C++ member), 145
httpd_ssl_config::servercert_len (C++ member), 146
httpd_ssl_config::session_tickets (C++ member), 146
httpd_ssl_config::ssl_userdata (C++ member), 146
httpd_ssl_config::transport_mode (C++ member), 146
httpd_ssl_config::use_ecdsa_peripheral (C++ member), 146
httpd_ssl_config::use_secure_element (C++ member), 146
httpd_ssl_config::user_cb (C++ member), 146
HTTPD_SSL_CONFIG_DEFAULT (C macro), 147
httpd_ssl_config_t (C++ type), 147
httpd_ssl_start (C++ function), 145
httpd_ssl_stop (C++ function), 145
httpd_ssl_transport_mode_t (C++ enum), 148
httpd_ssl_transport_mode_t::HTTPD_SSL_TRANSPORT_I (C++ enumerator), 148
httpd_ssl_transport_mode_t::HTTPD_SSL_TRANSPORT_S (C++ enumerator), 148
httpd_ssl_user_cb_state_t (C++ enum), 148
httpd_ssl_user_cb_state_t::HTTPD_SSL_USER_CB_SESS (C++ enumerator), 148
httpd_ssl_user_cb_state_t::HTTPD_SSL_USER_CB_SESS (C++ enumerator), 148
httpd_start (C++ function), 118
httpd_stop (C++ function), 118
HTTPD_TYPE_JSON (C macro), 138
HTTPD_TYPE_OCTET (C macro), 139
HTTPD_TYPE_TEXT (C macro), 138
httpd_unregister_uri (C++ function), 120
httpd_unregister_uri_handler (C++ function), 120
httpd_uri (C++ struct), 137
httpd_uri::handler (C++ member), 138
httpd_uri::method (C++ member), 137
httpd_uri::uri (C++ member), 137
httpd_uri::user_ctx (C++ member), 138
httpd_uri_match_func_t (C++ type), 140
httpd_uri_match_wildcard (C++ function), 125

httpd_uri_t (*C++ type*), 140
httpd_work_fn_t (*C++ type*), 142
HttpStatus_Code (*C++ enum*), 89
HttpStatus_Code::HttpStatus_BadRequest (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_Forbidden (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_Found (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_InternalError (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_MovedPermanently (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_MultipleChoices (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_NotFound (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_Ok (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_PermanentRedirect (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_SeeOther (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_TemporaryRedirect (*C++ enumerator*), 89
HttpStatus_Code::HttpStatus_Unauthorized (*C++ enumerator*), 89

I

i2c_ack_type_t (*C++ enum*), 394
i2c_ack_type_t::I2C_MASTER_ACK (*C++ enumerator*), 394
i2c_ack_type_t::I2C_MASTER_ACK_MAX (*C++ enumerator*), 394
i2c_ack_type_t::I2C_MASTER_LAST_NACK (*C++ enumerator*), 394
i2c_ack_type_t::I2C_MASTER_NACK (*C++ enumerator*), 394
i2c_addr_bit_len_t (*C++ enum*), 392
i2c_addr_bit_len_t::I2C_ADDR_BIT_LEN_10 (*C++ enumerator*), 393
i2c_addr_bit_len_t::I2C_ADDR_BIT_LEN_7 (*C++ enumerator*), 393
i2c_addr_mode_t (*C++ enum*), 393
i2c_addr_mode_t::I2C_ADDR_BIT_10 (*C++ enumerator*), 393
i2c_addr_mode_t::I2C_ADDR_BIT_7 (*C++ enumerator*), 393
i2c_addr_mode_t::I2C_ADDR_BIT_MAX (*C++ enumerator*), 394
i2c_clock_source_t (*C++ type*), 392
i2c_del_master_bus (*C++ function*), 380
i2c_del_slave_device (*C++ function*), 386
i2c_device_config_t (*C++ struct*), 384
i2c_device_config_t::dev_addr_length (*C++ member*), 385
i2c_device_config_t::device_address (*C++ member*), 385
i2c_device_config_t::disable_ack_check (*C++ member*), 385
i2c_device_config_t::flags (*C++ member*), 385
i2c_device_config_t::scl_speed_hz (*C++ member*), 385
i2c_device_config_t::scl_wait_us (*C++ member*), 385
i2c_hal_clk_config_t (*C++ struct*), 391
i2c_hal_clk_config_t::clkm_div (*C++ member*), 391
i2c_hal_clk_config_t::hold (*C++ member*), 392
i2c_hal_clk_config_t::scl_high (*C++ member*), 392
i2c_hal_clk_config_t::scl_low (*C++ member*), 391
i2c_hal_clk_config_t::scl_wait_high (*C++ member*), 392
i2c_hal_clk_config_t::sda_hold (*C++ member*), 392
i2c_hal_clk_config_t::sda_sample (*C++ member*), 392
i2c_hal_clk_config_t::setup (*C++ member*), 392
i2c_hal_clk_config_t::tout (*C++ member*), 392
i2c_master_bus_add_device (*C++ function*), 380
i2c_master_bus_config_t (*C++ struct*), 384
i2c_master_bus_config_t::clk_source (*C++ member*), 384
i2c_master_bus_config_t::enable_internal_pullup (*C++ member*), 384
i2c_master_bus_config_t::flags (*C++ member*), 384
i2c_master_bus_config_t::glitch_ignore_cnt (*C++ member*), 384
i2c_master_bus_config_t::i2c_port (*C++ member*), 384
i2c_master_bus_config_t::intr_priority (*C++ member*), 384
i2c_master_bus_config_t::lp_source_clk (*C++ member*), 384
i2c_master_bus_config_t::scl_io_num (*C++ member*), 384
i2c_master_bus_config_t::sda_io_num (*C++ member*), 384
i2c_master_bus_config_t::trans_queue_depth (*C++ member*), 384
i2c_master_bus_handle_t (*C++ type*), 390
i2c_master_bus_reset (*C++ function*), 383
i2c_master_bus_rm_device (*C++ function*), 380
i2c_master_bus_wait_all_done (*C++ function*), 383
i2c_master_callback_t (*C++ type*), 390
i2c_master_dev_handle_t (*C++ type*), 390

i2c_master_event_callbacks_t (*C++ struct*), 385
i2c_master_event_callbacks_t::on_trans_done (*C++ member*), 385
i2c_master_event_data_t (*C++ struct*), 389
i2c_master_event_data_t::event (*C++ member*), 389
i2c_master_event_t (*C++ enum*), 391
i2c_master_event_t::I2C_EVENT_ALIVE (*C++ enumerator*), 391
i2c_master_event_t::I2C_EVENT_DONE (*C++ enumerator*), 391
i2c_master_event_t::I2C_EVENT_NACK (*C++ enumerator*), 391
i2c_master_event_t::I2C_EVENT_TIMEOUT (*C++ enumerator*), 391
i2c_master_get_bus_handle (*C++ function*), 383
i2c_master_probe (*C++ function*), 382
i2c_master_receive (*C++ function*), 381
i2c_master_register_event_callbacks (*C++ function*), 383
i2c_master_status_t (*C++ enum*), 390
i2c_master_status_t::I2C_STATUS_ACK_ERROR (*C++ enumerator*), 391
i2c_master_status_t::I2C_STATUS_DONE (*C++ enumerator*), 391
i2c_master_status_t::I2C_STATUS_IDLE (*C++ enumerator*), 391
i2c_master_status_t::I2C_STATUS_READ (*C++ enumerator*), 390
i2c_master_status_t::I2C_STATUS_START (*C++ enumerator*), 390
i2c_master_status_t::I2C_STATUS_STOP (*C++ enumerator*), 390
i2c_master_status_t::I2C_STATUS_TIMEOUT (*C++ enumerator*), 391
i2c_master_status_t::I2C_STATUS_WRITE (*C++ enumerator*), 390
i2c_master_transmit (*C++ function*), 381
i2c_master_transmit_receive (*C++ function*), 381
i2c_mode_t (*C++ enum*), 393
i2c_mode_t::I2C_MODE_MASTER (*C++ enumerator*), 393
i2c_mode_t::I2C_MODE_MAX (*C++ enumerator*), 393
i2c_mode_t::I2C_MODE_SLAVE (*C++ enumerator*), 393
i2c_new_master_bus (*C++ function*), 380
i2c_new_slave_device (*C++ function*), 386
i2c_port_num_t (*C++ type*), 390
i2c_port_t (*C++ enum*), 392
i2c_port_t::I2C_NUM_0 (*C++ enumerator*), 392
i2c_port_t::I2C_NUM_1 (*C++ enumerator*), 392
i2c_port_t::I2C_NUM_MAX (*C++ enumerator*), 392
i2c_port_t::LP_I2C_NUM_0 (*C++ enumerator*), 392
i2c_rw_t (*C++ enum*), 393
i2c_rw_t::I2C_MASTER_READ (*C++ enumerator*), 393
i2c_rw_t::I2C_MASTER_WRITE (*C++ enumerator*), 393
i2c_slave_config_t (*C++ struct*), 388
i2c_slave_config_t::access_ram_en (*C++ member*), 388
i2c_slave_config_t::addr_bit_len (*C++ member*), 388
i2c_slave_config_t::broadcast_en (*C++ member*), 388
i2c_slave_config_t::clk_source (*C++ member*), 388
i2c_slave_config_t::flags (*C++ member*), 389
i2c_slave_config_t::i2c_port (*C++ member*), 388
i2c_slave_config_t::intr_priority (*C++ member*), 388
i2c_slave_config_t::scl_io_num (*C++ member*), 388
i2c_slave_config_t::sda_io_num (*C++ member*), 388
i2c_slave_config_t::send_buf_depth (*C++ member*), 388
i2c_slave_config_t::slave_addr (*C++ member*), 388
i2c_slave_config_t::slave_unmatch_en (*C++ member*), 389
i2c_slave_dev_handle_t (*C++ type*), 390
i2c_slave_event_callbacks_t (*C++ struct*), 389
i2c_slave_event_callbacks_t::on_recv_done (*C++ member*), 389
i2c_slave_read_ram (*C++ function*), 387
i2c_slave_receive (*C++ function*), 386
i2c_slave_received_callback_t (*C++ type*), 390
i2c_slave_register_event_callbacks (*C++ function*), 387
i2c_slave_rx_done_event_data_t (*C++ struct*), 389
i2c_slave_rx_done_event_data_t::buffer (*C++ member*), 390
i2c_slave_stretch_cause_t (*C++ enum*), 394
i2c_slave_stretch_cause_t::I2C_SLAVE_STRETCH_CAUS (*C++ enumerator*), 394
i2c_slave_stretch_cause_t::I2C_SLAVE_STRETCH_CAUS (*C++ enumerator*), 394
i2c_slave_stretch_cause_t::I2C_SLAVE_STRETCH_CAUS (*C++ enumerator*), 394
i2c_slave_stretch_cause_t::I2C_SLAVE_STRETCH_CAUS (*C++ enumerator*), 394
i2c_slave_transmit (*C++ function*), 386
i2c_slave_write_ram (*C++ function*), 387
i2c_trans_mode_t (*C++ enum*), 393

i2c_trans_mode_t::I2C_DATA_MODE_LSB_FIRST (C++ enumerator), 393
i2c_trans_mode_t::I2C_DATA_MODE_MAX (C++ enumerator), 393
i2c_trans_mode_t::I2C_DATA_MODE_MSB_FIRST (C++ enumerator), 393
i2s_chan_config_t (C++ struct), 433
i2s_chan_config_t::auto_clear (C++ member), 434
i2s_chan_config_t::auto_clear_after_cb (C++ member), 434
i2s_chan_config_t::auto_clear_before_cb (C++ member), 434
i2s_chan_config_t::dma_desc_num (C++ member), 434
i2s_chan_config_t::dma_frame_num (C++ member), 434
i2s_chan_config_t::id (C++ member), 433
i2s_chan_config_t::intr_priority (C++ member), 434
i2s_chan_config_t::role (C++ member), 433
i2s_chan_handle_t (C++ type), 435
i2s_chan_info_t (C++ struct), 434
i2s_chan_info_t::dir (C++ member), 434
i2s_chan_info_t::id (C++ member), 434
i2s_chan_info_t::mode (C++ member), 434
i2s_chan_info_t::pair_chan (C++ member), 434
i2s_chan_info_t::role (C++ member), 434
i2s_chan_info_t::total_dma_buf_size (C++ member), 434
I2S_CHANNEL_DEFAULT_CONFIG (C macro), 435
i2s_channel_disable (C++ function), 431
i2s_channel_enable (C++ function), 430
i2s_channel_get_info (C++ function), 430
i2s_channel_init_pdm_rx_mode (C++ function), 415
i2s_channel_init_pdm_tx_mode (C++ function), 417
i2s_channel_init_std_mode (C++ function), 410
i2s_channel_init_tdm_mode (C++ function), 424
i2s_channel_preload_data (C++ function), 431
i2s_channel_read (C++ function), 432
i2s_channel_reconfig_pdm_rx_clock (C++ function), 416
i2s_channel_reconfig_pdm_rx_gpio (C++ function), 416
i2s_channel_reconfig_pdm_rx_slot (C++ function), 416
i2s_channel_reconfig_pdm_tx_clock (C++ function), 417
i2s_channel_reconfig_pdm_tx_gpio (C++ function), 418
i2s_channel_reconfig_pdm_tx_slot (C++ function), 417
i2s_channel_reconfig_std_clock (C++ function), 411
i2s_channel_reconfig_std_gpio (C++ function), 411
i2s_channel_reconfig_std_slot (C++ function), 411
i2s_channel_reconfig_tdm_clock (C++ function), 424
i2s_channel_reconfig_tdm_gpio (C++ function), 425
i2s_channel_reconfig_tdm_slot (C++ function), 425
i2s_channel_register_event_callback (C++ function), 432
i2s_channel_write (C++ function), 431
i2s_clock_src_t (C++ type), 437
i2s_comm_mode_t (C++ enum), 436
i2s_comm_mode_t::I2S_COMM_MODE_NONE (C++ enumerator), 436
i2s_comm_mode_t::I2S_COMM_MODE_PDM (C++ enumerator), 436
i2s_comm_mode_t::I2S_COMM_MODE_STD (C++ enumerator), 436
i2s_comm_mode_t::I2S_COMM_MODE_TDM (C++ enumerator), 436
i2s_data_bit_width_t (C++ enum), 438
i2s_data_bit_width_t::I2S_DATA_BIT_WIDTH_16BIT (C++ enumerator), 438
i2s_data_bit_width_t::I2S_DATA_BIT_WIDTH_24BIT (C++ enumerator), 438
i2s_data_bit_width_t::I2S_DATA_BIT_WIDTH_32BIT (C++ enumerator), 438
i2s_data_bit_width_t::I2S_DATA_BIT_WIDTH_8BIT (C++ enumerator), 438
i2s_del_channel (C++ function), 430
i2s_dir_t (C++ enum), 437
i2s_dir_t::I2S_DIR_RX (C++ enumerator), 438
i2s_dir_t::I2S_DIR_TX (C++ enumerator), 438
i2s_event_callbacks_t (C++ struct), 433
i2s_event_callbacks_t::on_recv (C++ member), 433
i2s_event_callbacks_t::on_recv_q_ovf (C++ member), 433
i2s_event_callbacks_t::on_send_q_ovf (C++ member), 433
i2s_event_callbacks_t::on_sent (C++ member), 433
i2s_event_data_t (C++ struct), 435
i2s_event_data_t::data (C++ member), 435
i2s_event_data_t::dma_buf (C++ member), 435
i2s_event_data_t::size (C++ member), 435
I2S_GPIO_UNUSED (C macro), 435
i2s_isr_callback_t (C++ type), 435
i2s_mclk_multiple_t (C++ enum), 436
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_1024 (C++ enumerator), 437
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_1152

(C++ enumerator), 437
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_128 (C++ enumerator), 436
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_192 (C++ enumerator), 436
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_256 (C++ enumerator), 437
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_384 (C++ enumerator), 437
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_512 (C++ enumerator), 437
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_576 (C++ enumerator), 437
i2s_mclk_multiple_t::I2S_MCLK_MULTIPLE_768 (C++ enumerator), 437
i2s_new_channel (C++ function), 429
i2s_pcm_compress_t (C++ enum), 439
i2s_pcm_compress_t::I2S_PCM_A_COMPRESS (C++ enumerator), 439
i2s_pcm_compress_t::I2S_PCM_A_DECOMPRESS (C++ enumerator), 439
i2s_pcm_compress_t::I2S_PCM_DISABLE (C++ enumerator), 439
i2s_pcm_compress_t::I2S_PCM_U_COMPRESS (C++ enumerator), 439
i2s_pcm_compress_t::I2S_PCM_U_DECOMPRESS (C++ enumerator), 439
i2s_pdm_dsr_t (C++ enum), 439
i2s_pdm_dsr_t::I2S_PDM_DSR_16S (C++ enumerator), 439
i2s_pdm_dsr_t::I2S_PDM_DSR_8S (C++ enumerator), 439
i2s_pdm_dsr_t::I2S_PDM_DSR_MAX (C++ enumerator), 439
i2s_pdm_rx_clk_config_t (C++ struct), 419
i2s_pdm_rx_clk_config_t::bclk_div (C++ member), 419
i2s_pdm_rx_clk_config_t::clk_src (C++ member), 419
i2s_pdm_rx_clk_config_t::dn_sample_mode (C++ member), 419
i2s_pdm_rx_clk_config_t::mclk_multiple (C++ member), 419
i2s_pdm_rx_clk_config_t::sample_rate_hz (C++ member), 419
I2S_PDM_RX_CLK_DEFAULT_CONFIG (C macro), 423
i2s_pdm_rx_config_t (C++ struct), 420
i2s_pdm_rx_config_t::clk_cfg (C++ member), 420
i2s_pdm_rx_config_t::gpio_cfg (C++ member), 420
i2s_pdm_rx_config_t::slot_cfg (C++ member), 420
i2s_pdm_rx_gpio_config_t (C++ struct), 419
i2s_pdm_rx_gpio_config_t::clk (C++ member), 419
i2s_pdm_rx_gpio_config_t::clk_inv (C++ member), 420
i2s_pdm_rx_gpio_config_t::din (C++ member), 420
i2s_pdm_rx_gpio_config_t::dins (C++ member), 420
i2s_pdm_rx_gpio_config_t::invert_flags (C++ member), 420
i2s_pdm_rx_slot_config_t (C++ struct), 418
i2s_pdm_rx_slot_config_t::amplify_num (C++ member), 419
i2s_pdm_rx_slot_config_t::data_bit_width (C++ member), 418
i2s_pdm_rx_slot_config_t::hp_cut_off_freq_hz (C++ member), 419
i2s_pdm_rx_slot_config_t::hp_en (C++ member), 419
i2s_pdm_rx_slot_config_t::slot_bit_width (C++ member), 418
i2s_pdm_rx_slot_config_t::slot_mask (C++ member), 419
i2s_pdm_rx_slot_config_t::slot_mode (C++ member), 419
I2S_PDM_RX_SLOT_DEFAULT_CONFIG (C macro), 422
i2s_pdm_sig_scale_t (C++ enum), 439
i2s_pdm_sig_scale_t::I2S_PDM_SIG_SCALING_DIV_2 (C++ enumerator), 439
i2s_pdm_sig_scale_t::I2S_PDM_SIG_SCALING_MUL_1 (C++ enumerator), 439
i2s_pdm_sig_scale_t::I2S_PDM_SIG_SCALING_MUL_2 (C++ enumerator), 439
i2s_pdm_sig_scale_t::I2S_PDM_SIG_SCALING_MUL_4 (C++ enumerator), 440
i2s_pdm_slot_mask_t (C++ enum), 440
i2s_pdm_slot_mask_t::I2S_PDM_LINE_SLOT_ALL (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_RX_LINE0_SLOT_LEFT (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_RX_LINE0_SLOT_RIGHT (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_RX_LINE1_SLOT_LEFT (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_RX_LINE1_SLOT_RIGHT (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_RX_LINE2_SLOT_LEFT (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_RX_LINE2_SLOT_RIGHT (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_RX_LINE3_SLOT_LEFT (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_RX_LINE3_SLOT_RIGHT (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_BOTH (C++ enumerator), 441
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_LEFT (C++ enumerator), 440
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_RIGHT (C++ enumerator), 440

i2s_pdm_tx_clk_config_t (C++ struct), 421
i2s_pdm_tx_clk_config_t::bclk_div (C++ member), 421
i2s_pdm_tx_clk_config_t::clk_src (C++ member), 421
i2s_pdm_tx_clk_config_t::mclk_multiple (C++ member), 421
i2s_pdm_tx_clk_config_t::sample_rate_hz (C++ member), 421
i2s_pdm_tx_clk_config_t::up_sample_fp (C++ member), 421
i2s_pdm_tx_clk_config_t::up_sample_fs (C++ member), 421
I2S_PDM_TX_CLK_DAC_DEFAULT_CONFIG (C macro), 423
I2S_PDM_TX_CLK_DEFAULT_CONFIG (C macro), 423
i2s_pdm_tx_config_t (C++ struct), 422
i2s_pdm_tx_config_t::clk_cfg (C++ member), 422
i2s_pdm_tx_config_t::gpio_cfg (C++ member), 422
i2s_pdm_tx_config_t::slot_cfg (C++ member), 422
i2s_pdm_tx_gpio_config_t (C++ struct), 422
i2s_pdm_tx_gpio_config_t::clk (C++ member), 422
i2s_pdm_tx_gpio_config_t::clk_inv (C++ member), 422
i2s_pdm_tx_gpio_config_t::dout (C++ member), 422
i2s_pdm_tx_gpio_config_t::dout2 (C++ member), 422
i2s_pdm_tx_gpio_config_t::invert_flags (C++ member), 422
i2s_pdm_tx_line_mode_t (C++ enum), 440
i2s_pdm_tx_line_mode_t::I2S_PDM_TX_ONE_LINE_CODEC (C++ enumerator), 440
i2s_pdm_tx_line_mode_t::I2S_PDM_TX_ONE_LINE_DAC (C++ enumerator), 440
i2s_pdm_tx_line_mode_t::I2S_PDM_TX_TWO_LINE_DAC (C++ enumerator), 440
i2s_pdm_tx_slot_config_t (C++ struct), 420
i2s_pdm_tx_slot_config_t::data_bit_width (C++ member), 420
i2s_pdm_tx_slot_config_t::hp_cut_off_freq_hz (C++ member), 421
i2s_pdm_tx_slot_config_t::hp_en (C++ member), 421
i2s_pdm_tx_slot_config_t::hp_scale (C++ member), 421
i2s_pdm_tx_slot_config_t::line_mode (C++ member), 421
i2s_pdm_tx_slot_config_t::lp_scale (C++ member), 421
i2s_pdm_tx_slot_config_t::sd_dither (C++ member), 421
i2s_pdm_tx_slot_config_t::sd_dither2 (C++ member), 421
i2s_pdm_tx_slot_config_t::sd_prescale (C++ member), 420
i2s_pdm_tx_slot_config_t::sd_scale (C++ member), 420
i2s_pdm_tx_slot_config_t::sinc_scale (C++ member), 421
i2s_pdm_tx_slot_config_t::slot_bit_width (C++ member), 420
i2s_pdm_tx_slot_config_t::slot_mode (C++ member), 420
I2S_PDM_TX_SLOT_DAC_DEFAULT_CONFIG (C macro), 423
I2S_PDM_TX_SLOT_DEFAULT_CONFIG (C macro), 423
i2s_port_t (C++ enum), 436
i2s_port_t::I2S_NUM_0 (C++ enumerator), 436
i2s_port_t::I2S_NUM_1 (C++ enumerator), 436
i2s_port_t::I2S_NUM_2 (C++ enumerator), 436
i2s_port_t::I2S_NUM_AUTO (C++ enumerator), 436
i2s_role_t (C++ enum), 438
i2s_role_t::I2S_ROLE_MASTER (C++ enumerator), 438
i2s_role_t::I2S_ROLE_SLAVE (C++ enumerator), 438
i2s_slot_bit_width_t (C++ enum), 438
i2s_slot_bit_width_t::I2S_SLOT_BIT_WIDTH_16BIT (C++ enumerator), 438
i2s_slot_bit_width_t::I2S_SLOT_BIT_WIDTH_24BIT (C++ enumerator), 438
i2s_slot_bit_width_t::I2S_SLOT_BIT_WIDTH_32BIT (C++ enumerator), 439
i2s_slot_bit_width_t::I2S_SLOT_BIT_WIDTH_8BIT (C++ enumerator), 438
i2s_slot_bit_width_t::I2S_SLOT_BIT_WIDTH_AUTO (C++ enumerator), 438
i2s_slot_mode_t (C++ enum), 437
i2s_slot_mode_t::I2S_SLOT_MODE_MONO (C++ enumerator), 437
i2s_slot_mode_t::I2S_SLOT_MODE_STEREO (C++ enumerator), 437
i2s_std_clk_config_t (C++ struct), 413
i2s_std_clk_config_t::clk_src (C++ member), 413
i2s_std_clk_config_t::ext_clk_freq_hz (C++ member), 413
i2s_std_clk_config_t::mclk_multiple (C++ member), 413
i2s_std_clk_config_t::sample_rate_hz (C++ member), 413
I2S_STD_CLK_DEFAULT_CONFIG (C macro), 415
i2s_std_config_t (C++ struct), 414
i2s_std_config_t::clk_cfg (C++ member), 414
i2s_std_config_t::gpio_cfg (C++ member), 414
i2s_std_config_t::slot_cfg (C++ member),

414
i2s_std_gpio_config_t (C++ struct), 413
i2s_std_gpio_config_t::bclk (C++ member), 413
i2s_std_gpio_config_t::bclk_inv (C++ member), 414
i2s_std_gpio_config_t::din (C++ member), 413
i2s_std_gpio_config_t::dout (C++ member), 413
i2s_std_gpio_config_t::invert_flags (C++ member), 414
i2s_std_gpio_config_t::mclk (C++ member), 413
i2s_std_gpio_config_t::mclk_inv (C++ member), 414
i2s_std_gpio_config_t::ws (C++ member), 413
i2s_std_gpio_config_t::ws_inv (C++ member), 414
I2S_STD_MSB_SLOT_DEFAULT_CONFIG (C macro), 415
I2S_STD_PCM_SLOT_DEFAULT_CONFIG (C macro), 414
I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG (C macro), 414
i2s_std_slot_config_t (C++ struct), 412
i2s_std_slot_config_t::big_endian (C++ member), 413
i2s_std_slot_config_t::bit_order_lsb (C++ member), 413
i2s_std_slot_config_t::bit_shift (C++ member), 412
i2s_std_slot_config_t::data_bit_width (C++ member), 412
i2s_std_slot_config_t::left_align (C++ member), 412
i2s_std_slot_config_t::slot_bit_width (C++ member), 412
i2s_std_slot_config_t::slot_mask (C++ member), 412
i2s_std_slot_config_t::slot_mode (C++ member), 412
i2s_std_slot_config_t::ws_pol (C++ member), 412
i2s_std_slot_config_t::ws_width (C++ member), 412
i2s_std_slot_mask_t (C++ enum), 440
i2s_std_slot_mask_t::I2S_STD_SLOT_BOTH (C++ enumerator), 440
i2s_std_slot_mask_t::I2S_STD_SLOT_LEFT (C++ enumerator), 440
i2s_std_slot_mask_t::I2S_STD_SLOT_RIGHT (C++ enumerator), 440
I2S_TDM_AUTO_SLOT_NUM (C macro), 428
I2S_TDM_AUTO_WS_WIDTH (C macro), 428
i2s_tdm_clk_config_t (C++ struct), 426
i2s_tdm_clk_config_t::bclk_div (C++ member), 427
i2s_tdm_clk_config_t::clk_src (C++ member), 427
i2s_tdm_clk_config_t::ext_clk_freq_hz (C++ member), 427
i2s_tdm_clk_config_t::mclk_multiple (C++ member), 427
i2s_tdm_clk_config_t::sample_rate_hz (C++ member), 427
I2S_TDM_CLK_DEFAULT_CONFIG (C macro), 429
i2s_tdm_config_t (C++ struct), 428
i2s_tdm_config_t::clk_cfg (C++ member), 428
i2s_tdm_config_t::gpio_cfg (C++ member), 428
i2s_tdm_config_t::slot_cfg (C++ member), 428
i2s_tdm_gpio_config_t (C++ struct), 427
i2s_tdm_gpio_config_t::bclk (C++ member), 427
i2s_tdm_gpio_config_t::bclk_inv (C++ member), 427
i2s_tdm_gpio_config_t::din (C++ member), 427
i2s_tdm_gpio_config_t::dout (C++ member), 427
i2s_tdm_gpio_config_t::invert_flags (C++ member), 428
i2s_tdm_gpio_config_t::mclk (C++ member), 427
i2s_tdm_gpio_config_t::mclk_inv (C++ member), 427
i2s_tdm_gpio_config_t::ws (C++ member), 427
i2s_tdm_gpio_config_t::ws_inv (C++ member), 427
I2S_TDM_MSB_SLOT_DEFAULT_CONFIG (C macro), 428
I2S_TDM_PCM_LONG_SLOT_DEFAULT_CONFIG (C macro), 428
I2S_TDM_PCM_SHORT_SLOT_DEFAULT_CONFIG (C macro), 428
I2S_TDM_PHILIPS_SLOT_DEFAULT_CONFIG (C macro), 428
i2s_tdm_slot_config_t (C++ struct), 425
i2s_tdm_slot_config_t::big_endian (C++ member), 426
i2s_tdm_slot_config_t::bit_order_lsb (C++ member), 426
i2s_tdm_slot_config_t::bit_shift (C++ member), 426
i2s_tdm_slot_config_t::data_bit_width (C++ member), 426
i2s_tdm_slot_config_t::left_align (C++ member), 426
i2s_tdm_slot_config_t::skip_mask (C++ member), 426
i2s_tdm_slot_config_t::slot_bit_width

(C++ member), 426
i2s_tdm_slot_config_t::slot_mask (C++ member), 426
i2s_tdm_slot_config_t::slot_mode (C++ member), 426
i2s_tdm_slot_config_t::total_slot (C++ member), 426
i2s_tdm_slot_config_t::ws_pol (C++ member), 426
i2s_tdm_slot_config_t::ws_width (C++ member), 426
i2s_tdm_slot_mask_t (C++ enum), 441
i2s_tdm_slot_mask_t::I2S_TDM_SLOT0 (C++ enumerator), 441
i2s_tdm_slot_mask_t::I2S_TDM_SLOT1 (C++ enumerator), 441
i2s_tdm_slot_mask_t::I2S_TDM_SLOT10 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT11 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT12 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT13 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT14 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT15 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT2 (C++ enumerator), 441
i2s_tdm_slot_mask_t::I2S_TDM_SLOT3 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT4 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT5 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT6 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT7 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT8 (C++ enumerator), 442
i2s_tdm_slot_mask_t::I2S_TDM_SLOT9 (C++ enumerator), 442
intr_handle_t (C++ type), 1623
intr_handler_t (C++ type), 1623
IP2STR (C macro), 250
IP4ADDR_STRLEN_MAX (C macro), 250
ip_event_add_ip6_t (C++ struct), 241
ip_event_add_ip6_t::addr (C++ member), 241
ip_event_add_ip6_t::preferred (C++ member), 241
ip_event_ap_staipassigned_t (C++ struct), 241
ip_event_ap_staipassigned_t::esp_netif (C++ member), 241
ip_event_ap_staipassigned_t::ip (C++ member), 241
ip_event_ap_staipassigned_t::mac (C++ member), 241
ip_event_got_ip6_t (C++ struct), 241
ip_event_got_ip6_t::esp_netif (C++ member), 241
ip_event_got_ip6_t::ip6_info (C++ member), 241
ip_event_got_ip6_t::ip_index (C++ member), 241
ip_event_got_ip_t (C++ struct), 240
ip_event_got_ip_t::esp_netif (C++ member), 241
ip_event_got_ip_t::ip_changed (C++ member), 241
ip_event_got_ip_t::ip_info (C++ member), 241
ip_event_t (C++ enum), 247
ip_event_t::IP_EVENT_AP_STAIPASSIGNED (C++ enumerator), 247
ip_event_t::IP_EVENT_ETH_GOT_IP (C++ enumerator), 247
ip_event_t::IP_EVENT_ETH_LOST_IP (C++ enumerator), 247
ip_event_t::IP_EVENT_GOT_IP6 (C++ enumerator), 247
ip_event_t::IP_EVENT_PPP_GOT_IP (C++ enumerator), 247
ip_event_t::IP_EVENT_PPP_LOST_IP (C++ enumerator), 247
ip_event_t::IP_EVENT_STA_GOT_IP (C++ enumerator), 247
ip_event_t::IP_EVENT_STA_LOST_IP (C++ enumerator), 247
IPSTR (C macro), 249
IPV62STR (C macro), 250
IPV6STR (C macro), 250
isp_af_ctlr_t (C++ type), 451
isp_af_result_t (C++ struct), 450
isp_af_result_t::definition (C++ member), 450
isp_af_result_t::luminance (C++ member), 450
isp_awb_ctlr_t (C++ type), 451
isp_awb_stat_result_t (C++ struct), 450
isp_awb_stat_result_t::sum_b (C++ member), 451
isp_awb_stat_result_t::sum_g (C++ member), 450
isp_awb_stat_result_t::sum_r (C++ member), 450
isp_awb_stat_result_t::white_patch_num (C++ member), 450
isp_float_range_t (C++ struct), 450
isp_float_range_t::max (C++ member), 450
isp_float_range_t::min (C++ member), 450
isp_proc_handle_t (C++ type), 451
isp_u32_range_t (C++ struct), 449

isp_u32_range_t::max (*C++ member*), 450
isp_u32_range_t::min (*C++ member*), 450

J

jpeg_alloc_decoder_mem (*C++ function*), 740
jpeg_alloc_encoder_mem (*C++ function*), 742
jpeg_codec_mode_t (*C++ enum*), 747
jpeg_codec_mode_t::JPEG_CODEC_DECODER (*C++ enumerator*), 747
jpeg_codec_mode_t::JPEG_CODEC_ENCODER (*C++ enumerator*), 747
jpeg_component_factor_t (*C++ struct*), 746
jpeg_component_factor_t::horizontal (*C++ member*), 746
jpeg_component_factor_t::vertical (*C++ member*), 746
JPEG_COMPONENT_NUMBER_MAX (*C macro*), 746
jpeg_dec_buffer_alloc_direction_t (*C++ enum*), 745
jpeg_dec_buffer_alloc_direction_t::JPEG_DEC_ALLOC_INPUT_BUFFER (*C++ enumerator*), 745
jpeg_dec_buffer_alloc_direction_t::JPEG_DEC_ALLOC_OUTPUT_BUFFER (*C++ enumerator*), 745
jpeg_dec_output_format_t (*C++ enum*), 744
jpeg_dec_output_format_t::JPEG_DECODE_OUT_FORMAT_GRAY (*C++ enumerator*), 744
jpeg_dec_output_format_t::JPEG_DECODE_OUT_FORMAT_RGB565 (*C++ enumerator*), 744
jpeg_dec_output_format_t::JPEG_DECODE_OUT_FORMAT_RGB888 (*C++ enumerator*), 744
jpeg_dec_output_format_t::JPEG_DECODE_OUT_FORMAT_YUV420 (*C++ enumerator*), 744
jpeg_dec_output_format_t::JPEG_DECODE_OUT_FORMAT_YUV422 (*C++ enumerator*), 744
jpeg_dec_output_format_t::JPEG_DECODE_OUT_FORMAT_YUV444 (*C++ enumerator*), 744
jpeg_dec_rgb_element_order_t (*C++ enum*), 744
jpeg_dec_rgb_element_order_t::JPEG_DEC_RGB_ELEMENT_ORDER_BGR (*C++ enumerator*), 744
jpeg_dec_rgb_element_order_t::JPEG_DEC_RGB_ELEMENT_ORDER_RGB (*C++ enumerator*), 744
jpeg_decode_cfg_t (*C++ struct*), 740
jpeg_decode_cfg_t::conv_std (*C++ member*), 740
jpeg_decode_cfg_t::output_format (*C++ member*), 740
jpeg_decode_cfg_t::rgb_order (*C++ member*), 740
jpeg_decode_engine_cfg_t (*C++ struct*), 740
jpeg_decode_engine_cfg_t::intr_priority (*C++ member*), 740
jpeg_decode_engine_cfg_t::timeout_ms (*C++ member*), 741
jpeg_decode_memory_alloc_cfg_t (*C++ struct*), 741
jpeg_decode_memory_alloc_cfg_t::buffer_direction (*C++ member*), 741
jpeg_decode_picture_info_t (*C++ struct*), 741
jpeg_decode_picture_info_t::height (*C++ member*), 741
jpeg_decode_picture_info_t::sample_method (*C++ member*), 741
jpeg_decode_picture_info_t::width (*C++ member*), 741
jpeg_decoder_get_info (*C++ function*), 739
jpeg_decoder_handle_t (*C++ type*), 744
jpeg_decoder_process (*C++ function*), 739
jpeg_del_decoder_engine (*C++ function*), 740
jpeg_del_encoder_engine (*C++ function*), 742
JPEG_DOWN_SAMPLING_NUM (*C macro*), 746
jpeg_down_sampling_type_t (*C++ enum*), 747
jpeg_down_sampling_type_t::JPEG_DOWN_SAMPLING_GRA (*C++ enumerator*), 747
jpeg_down_sampling_type_t::JPEG_DOWN_SAMPLING_YUV (*C++ enumerator*), 747
jpeg_down_sampling_type_t::JPEG_DOWN_SAMPLING_YUV (*C++ enumerator*), 747
jpeg_down_sampling_type_t::JPEG_DOWN_SAMPLING_YUV (*C++ enumerator*), 747
jpeg_enc_buffer_alloc_direction_t (*C++ enum*), 745
jpeg_enc_buffer_alloc_direction_t::JPEG_ENC_ALLOC (*C++ enumerator*), 745
jpeg_enc_buffer_alloc_direction_t::JPEG_ENC_ALLOC (*C++ enumerator*), 745
jpeg_enc_input_format_t (*C++ enum*), 745
jpeg_enc_input_format_t::JPEG_ENCODE_IN_FORMAT_GR (*C++ enumerator*), 745
jpeg_enc_input_format_t::JPEG_ENCODE_IN_FORMAT_RG (*C++ enumerator*), 745
jpeg_enc_input_format_t::JPEG_ENCODE_IN_FORMAT_RG (*C++ enumerator*), 745
jpeg_enc_input_format_t::JPEG_ENCODE_IN_FORMAT_YU (*C++ enumerator*), 745
jpeg_enc_src_type_t (*C++ enum*), 747
jpeg_enc_src_type_t::JPEG_ENC_SRC_GRAY (*C++ enumerator*), 748
jpeg_enc_src_type_t::JPEG_ENC_SRC_RGB565 (*C++ enumerator*), 748
jpeg_enc_src_type_t::JPEG_ENC_SRC_RGB888 (*C++ enumerator*), 747
jpeg_enc_src_type_t::JPEG_ENC_SRC_YUV422 (*C++ enumerator*), 747
jpeg_encode_cfg_t (*C++ struct*), 742
jpeg_encode_cfg_t::height (*C++ member*), 742
jpeg_encode_cfg_t::image_quality (*C++ member*), 743
jpeg_encode_cfg_t::src_type (*C++ member*), 743
jpeg_encode_cfg_t::sub_sample (*C++ member*), 743
jpeg_encode_cfg_t::width (*C++ member*), 743

jpeg_encode_engine_cfg_t (C++ struct), 743
jpeg_encode_engine_cfg_t::intr_priority (C++ member), 743
jpeg_encode_engine_cfg_t::timeout_ms (C++ member), 743
jpeg_encode_memory_alloc_cfg_t (C++ struct), 743
jpeg_encode_memory_alloc_cfg_t::buffer_direction (C++ member), 743
jpeg_encoder_handle_t (C++ type), 744
jpeg_encoder_process (C++ function), 742
JPEG_HUFFMAN_AC_VALUE_TABLE_LEN (C macro), 746
JPEG_HUFFMAN_BITS_LEN_TABLE_LEN (C macro), 746
JPEG_HUFFMAN_DC_VALUE_TABLE_LEN (C macro), 746
jpeg_huffman_table_info_t (C++ union), 745
jpeg_huffman_table_info_t::id (C++ member), 746
jpeg_huffman_table_info_t::info (C++ member), 746
jpeg_huffman_table_info_t::type (C++ member), 746
jpeg_huffman_table_info_t::[anonymous] (C++ member), 746
jpeg_new_decoder_engine (C++ function), 739
jpeg_new_encoder_engine (C++ function), 741
JPEG_QUANTIZATION_TABLE_LEN (C macro), 746
jpeg_sample_mode_t (C++ enum), 747
jpeg_sample_mode_t::JPEG_SAMPLE_MODE_YUV420 (C++ enumerator), 747
jpeg_sample_mode_t::JPEG_SAMPLE_MODE_YUV422 (C++ enumerator), 747
jpeg_sample_mode_t::JPEG_SAMPLE_MODE_YUV444 (C++ enumerator), 747
jpeg_yuv_rgb_conv_std_t (C++ enum), 744
jpeg_yuv_rgb_conv_std_t::JPEG_YUV_RGB_CONV_STD_BT601 (C++ enumerator), 744
jpeg_yuv_rgb_conv_std_t::JPEG_YUV_RGB_CONV_STD_BT709 (C++ enumerator), 744

L

l2tap_ioctl_opt_t (C++ enum), 252
l2tap_ioctl_opt_t::L2TAP_G_DEVICE_DRV_HNDL (C++ enumerator), 252
l2tap_ioctl_opt_t::L2TAP_G_INTF_DEVICE (C++ enumerator), 252
l2tap_ioctl_opt_t::L2TAP_G_RCV_FILTER (C++ enumerator), 252
l2tap_ioctl_opt_t::L2TAP_S_DEVICE_DRV_HNDL (C++ enumerator), 252
l2tap_ioctl_opt_t::L2TAP_S_INTF_DEVICE (C++ enumerator), 252
l2tap_ioctl_opt_t::L2TAP_S_RCV_FILTER (C++ enumerator), 252
l2tap_iodriver_handle (C++ type), 252
L2TAP_VFS_CONFIG_DEFAULT (C macro), 252
l2tap_vfs_config_t (C++ struct), 252
l2tap_vfs_config_t::base_path (C++ member), 252
L2TAP_VFS_DEFAULT_PATH (C macro), 252
lcd_clock_source_t (C++ type), 474
lcd_color_format_t (C++ enum), 475
lcd_color_format_t::LCD_COLOR_FMT_RGB565 (C++ enumerator), 475
lcd_color_format_t::LCD_COLOR_FMT_RGB666 (C++ enumerator), 475
lcd_color_format_t::LCD_COLOR_FMT_RGB888 (C++ enumerator), 475
lcd_color_format_t::LCD_COLOR_FMT_YUV422 (C++ enumerator), 475
lcd_color_range_t (C++ enum), 475
lcd_color_range_t::LCD_COLOR_RANGE_FULL (C++ enumerator), 475
lcd_color_range_t::LCD_COLOR_RANGE_LIMIT (C++ enumerator), 475
lcd_color_rgb_pixel_format_t (C++ enum), 475
lcd_color_rgb_pixel_format_t::LCD_COLOR_PIXEL_FOR (C++ enumerator), 475
lcd_color_rgb_pixel_format_t::LCD_COLOR_PIXEL_FOR (C++ enumerator), 475
lcd_color_rgb_pixel_format_t::LCD_COLOR_PIXEL_FOR (C++ enumerator), 475
lcd_color_space_t (C++ enum), 474
lcd_color_space_t::LCD_COLOR_SPACE_RGB (C++ enumerator), 474
lcd_color_space_t::LCD_COLOR_SPACE_YUV (C++ enumerator), 474
lcd_rgb_data_endian_t (C++ enum), 474
lcd_rgb_data_endian_t::LCD_RGB_DATA_ENDIAN_BIG (C++ enumerator), 474
lcd_rgb_data_endian_t::LCD_RGB_DATA_ENDIAN_LITTLE (C++ enumerator), 474
lcd_rgb_element_order_t (C++ enum), 478
lcd_rgb_element_order_t::LCD_RGB_ELEMENT_ORDER_BG (C++ enumerator), 478
lcd_rgb_element_order_t::LCD_RGB_ELEMENT_ORDER_RG (C++ enumerator), 478
lcd_yuv422_pack_order_t (C++ enum), 476
lcd_yuv422_pack_order_t::LCD_YUV422_PACK_ORDER_UY (C++ enumerator), 476
lcd_yuv422_pack_order_t::LCD_YUV422_PACK_ORDER_VY (C++ enumerator), 476
lcd_yuv422_pack_order_t::LCD_YUV422_PACK_ORDER_YU (C++ enumerator), 476
lcd_yuv422_pack_order_t::LCD_YUV422_PACK_ORDER_YV (C++ enumerator), 476
lcd_yuv_conv_std_t (C++ enum), 476
lcd_yuv_conv_std_t::LCD_YUV_CONV_STD_BT601 (C++ enumerator), 476
lcd_yuv_conv_std_t::LCD_YUV_CONV_STD_BT709 (C++ enumerator), 476
lcd_yuv_sample_t (C++ enum), 475

lcd_yuv_sample_t::LCD_YUV_SAMPLE_411 (C++ enumerator), 476
lcd_yuv_sample_t::LCD_YUV_SAMPLE_420 (C++ enumerator), 475
lcd_yuv_sample_t::LCD_YUV_SAMPLE_422 (C++ enumerator), 475
ledc_bind_channel_timer (C++ function), 495
ledc_cb_event_t (C++ enum), 505
ledc_cb_event_t::LEDC_FADE_END_EVT (C++ enumerator), 505
ledc_cb_param_t (C++ struct), 503
ledc_cb_param_t::channel (C++ member), 503
ledc_cb_param_t::duty (C++ member), 503
ledc_cb_param_t::event (C++ member), 503
ledc_cb_param_t::speed_mode (C++ member), 503
ledc_cb_register (C++ function), 499
ledc_cb_t (C++ type), 505
ledc_cbs_t (C++ struct), 503
ledc_cbs_t::fade_cb (C++ member), 503
ledc_channel_config (C++ function), 490
ledc_channel_config_t (C++ struct), 501
ledc_channel_config_t::channel (C++ member), 502
ledc_channel_config_t::duty (C++ member), 502
ledc_channel_config_t::flags (C++ member), 502
ledc_channel_config_t::gpio_num (C++ member), 502
ledc_channel_config_t::hpoint (C++ member), 502
ledc_channel_config_t::intr_type (C++ member), 502
ledc_channel_config_t::output_invert (C++ member), 502
ledc_channel_config_t::speed_mode (C++ member), 502
ledc_channel_config_t::timer_sel (C++ member), 502
ledc_channel_t (C++ enum), 507
ledc_channel_t::LEDC_CHANNEL_0 (C++ enumerator), 507
ledc_channel_t::LEDC_CHANNEL_1 (C++ enumerator), 507
ledc_channel_t::LEDC_CHANNEL_2 (C++ enumerator), 507
ledc_channel_t::LEDC_CHANNEL_3 (C++ enumerator), 507
ledc_channel_t::LEDC_CHANNEL_4 (C++ enumerator), 507
ledc_channel_t::LEDC_CHANNEL_5 (C++ enumerator), 508
ledc_channel_t::LEDC_CHANNEL_6 (C++ enumerator), 508
ledc_channel_t::LEDC_CHANNEL_7 (C++ enumerator), 508
ledc_channel_t::LEDC_CHANNEL_MAX (C++ enumerator), 508
ledc_clk_cfg_t (C++ type), 506
ledc_clk_src_t (C++ enum), 507
ledc_clk_src_t::LEDC_SCLK (C++ enumerator), 507
ledc_duty_direction_t (C++ enum), 506
ledc_duty_direction_t::LEDC_DUTY_DIR_DECREASE (C++ enumerator), 506
ledc_duty_direction_t::LEDC_DUTY_DIR_INCREASE (C++ enumerator), 506
ledc_duty_direction_t::LEDC_DUTY_DIR_MAX (C++ enumerator), 506
LEDC_ERR_DUTY (C macro), 505
LEDC_ERR_VAL (C macro), 505
ledc_fade_func_install (C++ function), 496
ledc_fade_func_uninstall (C++ function), 496
ledc_fade_mode_t (C++ enum), 509
ledc_fade_mode_t::LEDC_FADE_MAX (C++ enumerator), 509
ledc_fade_mode_t::LEDC_FADE_NO_WAIT (C++ enumerator), 509
ledc_fade_mode_t::LEDC_FADE_WAIT_DONE (C++ enumerator), 509
ledc_fade_param_config_t (C++ struct), 503
ledc_fade_param_config_t::cycle_num (C++ member), 505
ledc_fade_param_config_t::dir (C++ member), 505
ledc_fade_param_config_t::scale (C++ member), 505
ledc_fade_param_config_t::step_num (C++ member), 505
ledc_fade_start (C++ function), 496
ledc_fade_stop (C++ function), 497
ledc_fill_multi_fade_param_list (C++ function), 500
ledc_find_suitable_duty_resolution (C++ function), 490
ledc_get_duty (C++ function), 493
ledc_get_freq (C++ function), 492
ledc_get_hpoint (C++ function), 492
ledc_intr_type_t (C++ enum), 506
ledc_intr_type_t::LEDC_INTR_DISABLE (C++ enumerator), 506
ledc_intr_type_t::LEDC_INTR_FADE_END (C++ enumerator), 506
ledc_intr_type_t::LEDC_INTR_MAX (C++ enumerator), 506
ledc_isr_handle_t (C++ type), 505
ledc_isr_register (C++ function), 494
ledc_mode_t (C++ enum), 506
ledc_mode_t::LEDC_LOW_SPEED_MODE (C++ enumerator), 506
ledc_mode_t::LEDC_SPEED_MODE_MAX (C++ enumerator), 506
ledc_read_fade_param (C++ function), 501

ledc_set_duty (*C++ function*), 493
ledc_set_duty_and_update (*C++ function*), 497
ledc_set_duty_with_hpoint (*C++ function*), 492
ledc_set_fade (*C++ function*), 493
ledc_set_fade_step_and_start (*C++ function*), 498
ledc_set_fade_time_and_start (*C++ function*), 498
ledc_set_fade_with_step (*C++ function*), 495
ledc_set_fade_with_time (*C++ function*), 496
ledc_set_freq (*C++ function*), 492
ledc_set_multi_fade (*C++ function*), 499
ledc_set_multi_fade_and_start (*C++ function*), 500
ledc_set_pin (*C++ function*), 491
ledc_slow_clk_sel_t (*C++ enum*), 506
ledc_slow_clk_sel_t::LEDC_SLOW_CLK_PLL_DIV (*C++ enumerator*), 506
ledc_slow_clk_sel_t::LEDC_SLOW_CLK_RC_FAST (*C++ enumerator*), 506
ledc_slow_clk_sel_t::LEDC_SLOW_CLK_RTC8M (*C++ enumerator*), 507
ledc_slow_clk_sel_t::LEDC_SLOW_CLK_XTAL (*C++ enumerator*), 506
ledc_stop (*C++ function*), 491
ledc_timer_bit_t (*C++ enum*), 508
ledc_timer_bit_t::LEDC_TIMER_10_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_11_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_12_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_13_BIT (*C++ enumerator*), 509
ledc_timer_bit_t::LEDC_TIMER_14_BIT (*C++ enumerator*), 509
ledc_timer_bit_t::LEDC_TIMER_15_BIT (*C++ enumerator*), 509
ledc_timer_bit_t::LEDC_TIMER_16_BIT (*C++ enumerator*), 509
ledc_timer_bit_t::LEDC_TIMER_17_BIT (*C++ enumerator*), 509
ledc_timer_bit_t::LEDC_TIMER_18_BIT (*C++ enumerator*), 509
ledc_timer_bit_t::LEDC_TIMER_19_BIT (*C++ enumerator*), 509
ledc_timer_bit_t::LEDC_TIMER_1_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_20_BIT (*C++ enumerator*), 509
ledc_timer_bit_t::LEDC_TIMER_2_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_3_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_4_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_5_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_6_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_7_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_8_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_9_BIT (*C++ enumerator*), 508
ledc_timer_bit_t::LEDC_TIMER_BIT_MAX (*C++ enumerator*), 509
ledc_timer_config (*C++ function*), 490
ledc_timer_config_t (*C++ struct*), 502
ledc_timer_config_t::clk_cfg (*C++ member*), 502
ledc_timer_config_t::deconfigure (*C++ member*), 503
ledc_timer_config_t::duty_resolution (*C++ member*), 502
ledc_timer_config_t::freq_hz (*C++ member*), 502
ledc_timer_config_t::speed_mode (*C++ member*), 502
ledc_timer_config_t::timer_num (*C++ member*), 502
ledc_timer_pause (*C++ function*), 494
ledc_timer_resume (*C++ function*), 495
ledc_timer_rst (*C++ function*), 494
ledc_timer_set (*C++ function*), 494
ledc_timer_t (*C++ enum*), 507
ledc_timer_t::LEDC_TIMER_0 (*C++ enumerator*), 507
ledc_timer_t::LEDC_TIMER_1 (*C++ enumerator*), 507
ledc_timer_t::LEDC_TIMER_2 (*C++ enumerator*), 507
ledc_timer_t::LEDC_TIMER_3 (*C++ enumerator*), 507
ledc_timer_t::LEDC_TIMER_MAX (*C++ enumerator*), 507
ledc_update_duty (*C++ function*), 490
linenoiseCompletions (*C++ type*), 1374
lp_core_etm_event_config_t (*C++ struct*), 1742
lp_core_etm_event_config_t::event_type (*C++ member*), 1742
lp_core_etm_event_type_t (*C++ enum*), 1742
lp_core_etm_event_type_t::LP_CORE_EVENT_ERR_INTR (*C++ enumerator*), 1742
lp_core_etm_event_type_t::LP_CORE_EVENT_MAX (*C++ enumerator*), 1742
lp_core_etm_event_type_t::LP_CORE_EVENT_START_INT (*C++ enumerator*), 1742
lp_core_etm_task_config_t (*C++ struct*), 1742
lp_core_etm_task_config_t::task_type (*C++ member*), 1742

lp_core_etm_task_type_t (C++ enum), 1742
lp_core_etm_task_type_t::LP_CORE_TASK_MAX (C++ enumerator), 1743
lp_core_etm_task_type_t::LP_CORE_TASK_WAKEUP_CPU (C++ enumerator), 1742
lp_core_i2c_cfg_t (C++ struct), 1736
lp_core_i2c_cfg_t::i2c_pin_cfg (C++ member), 1736
lp_core_i2c_cfg_t::i2c_src_clk (C++ member), 1736
lp_core_i2c_cfg_t::i2c_timing_cfg (C++ member), 1736
LP_CORE_I2C_DEFAULT_CONFIG (C macro), 1736
lp_core_i2c_master_init (C++ function), 1735
lp_core_i2c_master_read_from_device (C++ function), 1747
lp_core_i2c_master_set_ack_check_en (C++ function), 1748
lp_core_i2c_master_write_read_device (C++ function), 1748
lp_core_i2c_master_write_to_device (C++ function), 1747
lp_core_i2c_pin_cfg_t (C++ struct), 1735
lp_core_i2c_pin_cfg_t::scl_io_num (C++ member), 1735
lp_core_i2c_pin_cfg_t::scl_pullup_en (C++ member), 1735
lp_core_i2c_pin_cfg_t::sda_io_num (C++ member), 1735
lp_core_i2c_pin_cfg_t::sda_pullup_en (C++ member), 1735
lp_core_i2c_timing_cfg_t (C++ struct), 1735
lp_core_i2c_timing_cfg_t::clk_speed_hz (C++ member), 1736
LP_CORE_ISR_ATTR (C macro), 1750
lp_core_lp_spi_bus_add_device (C++ function), 1738
lp_core_lp_spi_bus_initialize (C++ function), 1738
lp_core_lp_spi_master_transfer (C++ function), 1750
lp_core_lp_spi_slave_initialize (C++ function), 1739
lp_core_lp_spi_slave_transfer (C++ function), 1751
lp_core_new_etm_event (C++ function), 1741
lp_core_new_etm_task (C++ function), 1741
lp_core_print_char (C++ function), 1750
lp_core_printf (C++ function), 1749
lp_core_uart_cfg_t (C++ struct), 1737
lp_core_uart_cfg_t::lp_uart_source_clk (C++ member), 1738
lp_core_uart_cfg_t::uart_pin_cfg (C++ member), 1737
lp_core_uart_cfg_t::uart_proto_cfg (C++ member), 1738
LP_CORE_UART_DEFAULT_CONFIG (C macro), 1738
lp_core_uart_init (C++ function), 1736
lp_core_uart_pin_cfg_t (C++ struct), 1737
lp_core_uart_pin_cfg_t::cts_io_num (C++ member), 1737
lp_core_uart_pin_cfg_t::rts_io_num (C++ member), 1737
lp_core_uart_pin_cfg_t::rx_io_num (C++ member), 1737
lp_core_uart_pin_cfg_t::tx_io_num (C++ member), 1737
lp_core_uart_proto_cfg_t (C++ struct), 1737
lp_core_uart_proto_cfg_t::baud_rate (C++ member), 1737
lp_core_uart_proto_cfg_t::data_bits (C++ member), 1737
lp_core_uart_proto_cfg_t::flow_ctrl (C++ member), 1737
lp_core_uart_proto_cfg_t::parity (C++ member), 1737
lp_core_uart_proto_cfg_t::rx_flow_ctrl_thresh (C++ member), 1737
lp_core_uart_proto_cfg_t::stop_bits (C++ member), 1737
lp_core_uart_read_bytes (C++ function), 1749
lp_core_uart_tx_chars (C++ function), 1749
lp_core_uart_write_bytes (C++ function), 1749
lp_gpio_connect_in_signal (C++ function), 336
lp_gpio_connect_out_signal (C++ function), 336
lp_i2c_clock_source_t (C++ type), 392
LP_I2C_DEFAULT_GPIO_CONFIG (C macro), 1736
LP_I2C_DEFAULT_SRC_CLK (C macro), 1736
LP_I2C_FAST_MODE_CONFIG (C macro), 1736
LP_I2C_STANDARD_MODE_CONFIG (C macro), 1736
lp_io_intr_type_t (C++ enum), 1746
lp_io_intr_type_t::LP_IO_INTR_ANYEDGE (C++ enumerator), 1747
lp_io_intr_type_t::LP_IO_INTR_DISABLE (C++ enumerator), 1746
lp_io_intr_type_t::LP_IO_INTR_HIGH_LEVEL (C++ enumerator), 1747
lp_io_intr_type_t::LP_IO_INTR_LOW_LEVEL (C++ enumerator), 1747
lp_io_intr_type_t::LP_IO_INTR_NEGEDGE (C++ enumerator), 1747
lp_io_intr_type_t::LP_IO_INTR_POSEDGE (C++ enumerator), 1746
lp_io_num_t (C++ enum), 1745
lp_io_num_t::LP_IO_NUM_0 (C++ enumerator), 1745
lp_io_num_t::LP_IO_NUM_1 (C++ enumerator), 1745
lp_io_num_t::LP_IO_NUM_10 (C++ enumera-

tor), 1746
lp_io_num_t::LP_IO_NUM_11 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_12 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_13 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_14 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_15 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_2 (C++ enumerator), 1745
lp_io_num_t::LP_IO_NUM_3 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_4 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_5 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_6 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_7 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_8 (C++ enumerator), 1746
lp_io_num_t::LP_IO_NUM_9 (C++ enumerator), 1746
lp_spi_bus_config_t (C++ struct), 1739
lp_spi_bus_config_t::miso_io_num (C++ member), 1739
lp_spi_bus_config_t::mosi_io_num (C++ member), 1739
lp_spi_bus_config_t::sclk_io_num (C++ member), 1739
lp_spi_bus_t (C++ type), 1752
LP_SPI_DEVICE_3WIRE (C macro), 1740
LP_SPI_DEVICE_BIT_LSBFIRST (C macro), 1740
lp_spi_device_config_t (C++ struct), 1739
lp_spi_device_config_t::clock_speed_hz (C++ member), 1739
lp_spi_device_config_t::cs_ena_posttrans (C++ member), 1740
lp_spi_device_config_t::cs_ena_pretrans (C++ member), 1740
lp_spi_device_config_t::cs_io_num (C++ member), 1739
lp_spi_device_config_t::duty_cycle (C++ member), 1740
lp_spi_device_config_t::flags (C++ member), 1740
lp_spi_device_config_t::spi_mode (C++ member), 1739
LP_SPI_DEVICE_CS_ACTIVE_HIGH (C macro), 1740
LP_SPI_DEVICE_HALF_DUPLEX (C macro), 1741
LP_SPI_DEVICE_RXBIT_LSBFIRST (C macro), 1740
LP_SPI_DEVICE_TXBIT_LSBFIRST (C macro), 1740
lp_spi_host_t (C++ type), 1741
lp_spi_slave_config_t (C++ struct), 1740
lp_spi_slave_config_t::cs_io_num (C++ member), 1740
lp_spi_slave_config_t::flags (C++ member), 1740
lp_spi_slave_config_t::spi_mode (C++ member), 1740
lp_spi_transaction_t (C++ struct), 1751
lp_spi_transaction_t::address (C++ member), 1751
lp_spi_transaction_t::address_bits (C++ member), 1751
lp_spi_transaction_t::bus (C++ member), 1751
lp_spi_transaction_t::command (C++ member), 1751
lp_spi_transaction_t::command_bits (C++ member), 1751
lp_spi_transaction_t::dummy_bits (C++ member), 1751
lp_spi_transaction_t::rx_buffer (C++ member), 1751
lp_spi_transaction_t::rx_length (C++ member), 1751
lp_spi_transaction_t::tx_buffer (C++ member), 1751
lp_spi_transaction_t::tx_length (C++ member), 1751
LP_UART_DEFAULT_CLOCK_CONFIG (C macro), 1738
LP_UART_DEFAULT_CTS_GPIO_NUM (C macro), 1738
LP_UART_DEFAULT_GPIO_CONFIG (C macro), 1738
LP_UART_DEFAULT_PROTO_CONFIG (C macro), 1738
LP_UART_DEFAULT_RTS_GPIO_NUM (C macro), 1738
LP_UART_DEFAULT_RX_GPIO_NUM (C macro), 1738
LP_UART_DEFAULT_TX_GPIO_NUM (C macro), 1738
lp_uart_sclk_t (C++ type), 824

M

MAC2STR (C macro), 1644
MACSTR (C macro), 1644
MALLOC_CAP_32BIT (C macro), 1580
MALLOC_CAP_8BIT (C macro), 1580
MALLOC_CAP_CACHE_ALIGNED (C macro), 1581
MALLOC_CAP_DEFAULT (C macro), 1581
MALLOC_CAP_DMA (C macro), 1580
MALLOC_CAP_DMA_DESC_AHB (C macro), 1581
MALLOC_CAP_DMA_DESC_AXI (C macro), 1581
MALLOC_CAP_EXEC (C macro), 1580
MALLOC_CAP_INTERNAL (C macro), 1581

MALLOC_CAP_INVALID (*C macro*), 1581
MALLOC_CAP_IRAM_8BIT (*C macro*), 1581
MALLOC_CAP_PID2 (*C macro*), 1580
MALLOC_CAP_PID3 (*C macro*), 1580
MALLOC_CAP_PID4 (*C macro*), 1580
MALLOC_CAP_PID5 (*C macro*), 1581
MALLOC_CAP_PID6 (*C macro*), 1581
MALLOC_CAP_PID7 (*C macro*), 1581
MALLOC_CAP_RETENTION (*C macro*), 1581
MALLOC_CAP_RTCRAM (*C macro*), 1581
MALLOC_CAP_SPIRAM (*C macro*), 1581
MALLOC_CAP_TCM (*C macro*), 1581
MAX_FDS (*C macro*), 1337
mcpwm_brake_config_t (*C++ struct*), 543
mcpwm_brake_config_t::brake_mode (*C++ member*), 543
mcpwm_brake_config_t::cbc_recover_on_tez (*C++ member*), 543
mcpwm_brake_config_t::cbc_recover_on_tez (*C++ member*), 543
mcpwm_brake_config_t::fault (*C++ member*), 543
mcpwm_brake_config_t::flags (*C++ member*), 543
mcpwm_brake_event_cb_t (*C++ type*), 567
mcpwm_brake_event_data_t (*C++ struct*), 565
mcpwm_cap_channel_handle_t (*C++ type*), 567
mcpwm_cap_timer_handle_t (*C++ type*), 566
mcpwm_capture_channel_config_t (*C++ struct*), 563
mcpwm_capture_channel_config_t::extra_capture_config (*C++ struct*), 563
mcpwm_capture_channel_config_t::extra_capture_config.invert_cap_signal (*C++ member*), 564
mcpwm_capture_channel_config_t::extra_capture_config.pull_down (*C++ member*), 564
mcpwm_capture_channel_config_t::extra_capture_config.pull_up (*C++ member*), 564
mcpwm_capture_channel_config_t::extra_capture_config.sync_phase (*C++ member*), 563
mcpwm_capture_channel_config_t::extra_capture_config.sync_phase_config_t (*C++ struct*), 563
mcpwm_capture_channel_config_t::extra_capture_config.sync_phase_config_t::count_value (*C++ member*), 564
mcpwm_capture_channel_config_t::flags (*C++ member*), 563
mcpwm_capture_channel_config_t::gpio_num (*C++ member*), 563
mcpwm_capture_channel_config_t::intr_priority (*C++ member*), 563
mcpwm_capture_channel_config_t::prescale (*C++ member*), 563
mcpwm_capture_channel_disable (*C++ function*), 561
mcpwm_capture_channel_enable (*C++ function*), 561
mcpwm_capture_channel_register_event_callbacks (*C++ function*), 562
mcpwm_capture_channel_trigger_soft_catch (*C++ function*), 562
mcpwm_capture_clock_source_t (*C++ type*), 568
mcpwm_capture_edge_t (*C++ enum*), 570
mcpwm_capture_edge_t::MCPWM_CAP_EDGE_NEG (*C++ enumerator*), 570
mcpwm_capture_edge_t::MCPWM_CAP_EDGE_POS (*C++ enumerator*), 570
mcpwm_capture_event_callbacks_t (*C++ struct*), 564
mcpwm_capture_event_callbacks_t::on_cap (*C++ member*), 564
mcpwm_capture_event_cb_t (*C++ type*), 567
mcpwm_capture_event_data_t (*C++ struct*), 566
mcpwm_capture_event_data_t::cap_edge (*C++ member*), 566
mcpwm_capture_event_data_t::cap_value (*C++ member*), 566
mcpwm_capture_timer_config_t (*C++ struct*), 562
mcpwm_capture_timer_config_t::clk_src (*C++ member*), 562
mcpwm_capture_timer_config_t::group_id (*C++ member*), 562
mcpwm_capture_timer_config_t::resolution_hz (*C++ member*), 563
mcpwm_capture_timer_disable (*C++ function*), 559
mcpwm_capture_timer_enable (*C++ function*), 559
mcpwm_capture_timer_register_event_callbacks (*C++ function*), 560
mcpwm_capture_timer_set_phase_on_sync (*C++ function*), 560
mcpwm_capture_timer_start (*C++ function*), 560
mcpwm_capture_timer_stop (*C++ function*), 560
mcpwm_capture_timer_sync_phase_config_t (*C++ struct*), 563
mcpwm_capture_timer_sync_phase_config_t::count_value (*C++ member*), 563
mcpwm_capture_timer_sync_phase_config_t::direction (*C++ member*), 563
mcpwm_capture_timer_sync_phase_config_t::sync_src (*C++ member*), 563
mcpwm_carrier_clock_source_t (*C++ type*), 568
mcpwm_carrier_config_t (*C++ struct*), 544
mcpwm_carrier_config_t::clk_src (*C++ member*), 544
mcpwm_carrier_config_t::duty_cycle (*C++ member*), 544
mcpwm_carrier_config_t::first_pulse_duration_us

(C++ member), 544
mcpwm_carrier_config_t::flags (C++ member), 544
mcpwm_carrier_config_t::frequency_hz (C++ member), 544
mcpwm_carrier_config_t::invert_after_modulate (C++ member), 544
mcpwm_carrier_config_t::invert_before_modulate (C++ member), 544
mcpwm_cmpr_etm_event_config_t (C++ struct), 565
mcpwm_cmpr_etm_event_config_t::event_type (C++ member), 565
mcpwm_cmpr_handle_t (C++ type), 566
mcpwm_comparator_config_t (C++ struct), 546
mcpwm_comparator_config_t::flags (C++ member), 546
mcpwm_comparator_config_t::intr_priority (C++ member), 546
mcpwm_comparator_config_t::update_cmp_on_sync (C++ member), 546
mcpwm_comparator_config_t::update_cmp_on_tez (C++ member), 546
mcpwm_comparator_config_t::update_cmp_on_tep (C++ member), 546
mcpwm_comparator_etm_event_type_t (C++ enum), 570
mcpwm_comparator_etm_event_type_t::MCPWM_COMPARATOR_ETM_EVENT_EQUAL (C++ enumerator), 570
mcpwm_comparator_etm_event_type_t::MCPWM_COMPARATOR_ETM_EVENT_MAX (C++ enumerator), 570
mcpwm_comparator_event_callbacks_t (C++ struct), 547
mcpwm_comparator_event_callbacks_t::on_reach (C++ member), 547
mcpwm_comparator_new_etm_event (C++ function), 564
mcpwm_comparator_register_event_callbacks (C++ function), 545
mcpwm_comparator_set_compare_value (C++ function), 546
mcpwm_compare_event_cb_t (C++ type), 567
mcpwm_compare_event_data_t (C++ struct), 566
mcpwm_compare_event_data_t::compare_ticks (C++ member), 566
mcpwm_compare_event_data_t::direction (C++ member), 566
mcpwm_dead_time_config_t (C++ struct), 553
mcpwm_dead_time_config_t::flags (C++ member), 553
mcpwm_dead_time_config_t::invert_output (C++ member), 553
mcpwm_dead_time_config_t::negedge_delay_ticks (C++ member), 553
mcpwm_dead_time_config_t::posedge_delay_ticks (C++ member), 553
mcpwm_del_capture_channel (C++ function), 561
mcpwm_del_capture_timer (C++ function), 559
mcpwm_del_comparator (C++ function), 545
mcpwm_del_fault (C++ function), 554
mcpwm_del_generator (C++ function), 547
mcpwm_del_operator (C++ function), 541
mcpwm_del_sync_src (C++ function), 557
mcpwm_del_timer (C++ function), 537
mcpwm_event_comparator_config_t (C++ struct), 546
mcpwm_fault_event_callbacks_t (C++ struct), 556
mcpwm_fault_event_callbacks_t::on_fault_enter (C++ member), 556
mcpwm_fault_event_callbacks_t::on_fault_exit (C++ member), 556
mcpwm_fault_event_cb_t (C++ type), 567
mcpwm_fault_event_data_t (C++ struct), 566
mcpwm_fault_handle_t (C++ type), 566
mcpwm_fault_register_event_callbacks (C++ function), 555
MCPWM_GEN_BRAKE_EVENT_ACTION (C macro), 553
MCPWM_GEN_BRAKE_EVENT_ACTION_END (C macro), 553
mcpwm_gen_brake_event_action_t (C++ struct), 552
mcpwm_gen_brake_event_action_t::action (C++ member), 552
mcpwm_gen_brake_event_action_t::brake_mode (C++ member), 552
mcpwm_gen_brake_event_action_t::direction (C++ member), 552
MCPWM_GEN_COMPARE_EVENT_ACTION (C macro), 553
MCPWM_GEN_COMPARE_EVENT_ACTION_END (C macro), 553
mcpwm_gen_compare_event_action_t (C++ struct), 552
mcpwm_gen_compare_event_action_t::action (C++ member), 552
mcpwm_gen_compare_event_action_t::comparator (C++ member), 552
mcpwm_gen_compare_event_action_t::direction (C++ member), 552
MCPWM_GEN_FAULT_EVENT_ACTION (C macro), 554
mcpwm_gen_fault_event_action_t (C++ struct), 552
mcpwm_gen_fault_event_action_t::action (C++ member), 553
mcpwm_gen_fault_event_action_t::direction (C++ member), 552
mcpwm_gen_fault_event_action_t::fault (C++ member), 552
mcpwm_gen_handle_t (C++ type), 566
MCPWM_GEN_SYNC_EVENT_ACTION (C macro), 554

mcpwm_gen_sync_event_action_t (C++ struct), 553
mcpwm_gen_sync_event_action_t::action (C++ member), 553
mcpwm_gen_sync_event_action_t::direction (C++ member), 553
mcpwm_gen_sync_event_action_t::sync (C++ member), 553
MCPWM_GEN_TIMER_EVENT_ACTION (C macro), 553
MCPWM_GEN_TIMER_EVENT_ACTION_END (C macro), 553
mcpwm_gen_timer_event_action_t (C++ struct), 551
mcpwm_gen_timer_event_action_t::action (C++ member), 552
mcpwm_gen_timer_event_action_t::direction (C++ member), 551
mcpwm_gen_timer_event_action_t::event (C++ member), 552
mcpwm_generator_action_t (C++ enum), 569
mcpwm_generator_action_t::MCPWM_GEN_ACTION_HIGH (C++ enumerator), 569
mcpwm_generator_action_t::MCPWM_GEN_ACTION_KEEP (C++ enumerator), 569
mcpwm_generator_action_t::MCPWM_GEN_ACTION_LOW (C++ enumerator), 569
mcpwm_generator_action_t::MCPWM_GEN_ACTION_TOGGLE (C++ enumerator), 569
mcpwm_generator_config_t (C++ struct), 551
mcpwm_generator_config_t::flags (C++ member), 551
mcpwm_generator_config_t::gen_gpio_num (C++ member), 551
mcpwm_generator_config_t::invert_pwm (C++ member), 551
mcpwm_generator_config_t::io_loop_back (C++ member), 551
mcpwm_generator_config_t::io_od_mode (C++ member), 551
mcpwm_generator_config_t::pull_down (C++ member), 551
mcpwm_generator_config_t::pull_up (C++ member), 551
mcpwm_generator_set_action_on_brake_event (C++ function), 549
mcpwm_generator_set_action_on_compare_event (C++ function), 549
mcpwm_generator_set_action_on_fault_event (C++ function), 550
mcpwm_generator_set_action_on_sync_event (C++ function), 550
mcpwm_generator_set_action_on_timer_event (C++ function), 548
mcpwm_generator_set_actions_on_brake_event (C++ function), 550
mcpwm_generator_set_actions_on_compare_event (C++ function), 549
mcpwm_generator_set_actions_on_timer_event (C++ function), 548
mcpwm_generator_set_dead_time (C++ function), 550
mcpwm_generator_set_force_level (C++ function), 548
mcpwm_gpio_fault_config_t (C++ struct), 555
mcpwm_gpio_fault_config_t::active_level (C++ member), 555
mcpwm_gpio_fault_config_t::flags (C++ member), 556
mcpwm_gpio_fault_config_t::gpio_num (C++ member), 555
mcpwm_gpio_fault_config_t::group_id (C++ member), 555
mcpwm_gpio_fault_config_t::intr_priority (C++ member), 555
mcpwm_gpio_fault_config_t::io_loop_back (C++ member), 555
mcpwm_gpio_fault_config_t::pull_down (C++ member), 556
mcpwm_gpio_fault_config_t::pull_up (C++ member), 556
mcpwm_gpio_sync_src_config_t (C++ struct), 558
mcpwm_gpio_sync_src_config_t::active_neg (C++ member), 558
mcpwm_gpio_sync_src_config_t::flags (C++ member), 558
mcpwm_gpio_sync_src_config_t::gpio_num (C++ member), 558
mcpwm_gpio_sync_src_config_t::group_id (C++ member), 558
mcpwm_gpio_sync_src_config_t::io_loop_back (C++ member), 558
mcpwm_gpio_sync_src_config_t::pull_down (C++ member), 558
mcpwm_gpio_sync_src_config_t::pull_up (C++ member), 558
mcpwm_new_capture_channel (C++ function), 560
mcpwm_new_capture_timer (C++ function), 559
mcpwm_new_comparator (C++ function), 545
mcpwm_new_event_comparator (C++ function), 545
mcpwm_new_generator (C++ function), 547
mcpwm_new_gpio_fault (C++ function), 554
mcpwm_new_gpio_sync_src (C++ function), 557
mcpwm_new_operator (C++ function), 541
mcpwm_new_soft_fault (C++ function), 554
mcpwm_new_soft_sync_src (C++ function), 557
mcpwm_new_timer (C++ function), 537
mcpwm_new_timer_sync_src (C++ function), 556
mcpwm_oper_handle_t (C++ type), 566
mcpwm_operator_apply_carrier (C++ function), 542
mcpwm_operator_brake_mode_t (C++ enum),

569
mcpwm_operator_brake_mode_t::MCPWM_OPERATOR_BRAKE_MODE_CBC (C++ enumerator), 569
mcpwm_operator_brake_mode_t::MCPWM_OPERATOR_BRAKE_MODE_INVALID (C++ enumerator), 570
mcpwm_operator_brake_mode_t::MCPWM_OPERATOR_BRAKE_MODE_OST (C++ enumerator), 569
mcpwm_operator_config_t (C++ struct), 542
mcpwm_operator_config_t::flags (C++ member), 543
mcpwm_operator_config_t::group_id (C++ member), 543
mcpwm_operator_config_t::intr_priority (C++ member), 543
mcpwm_operator_config_t::update_dead_time_on_empty (C++ member), 543
mcpwm_operator_config_t::update_dead_time_on_sync (C++ member), 543
mcpwm_operator_config_t::update_dead_time_on_tez (C++ member), 543
mcpwm_operator_config_t::update_gen_action_on_empty (C++ member), 543
mcpwm_operator_config_t::update_gen_action_on_sync (C++ member), 543
mcpwm_operator_config_t::update_gen_action_on_tez (C++ member), 543
mcpwm_operator_connect_timer (C++ function), 541
mcpwm_operator_event_callbacks_t (C++ struct), 543
mcpwm_operator_event_callbacks_t::on_brake_cbc (C++ member), 544
mcpwm_operator_event_callbacks_t::on_brake_ost (C++ member), 544
mcpwm_operator_recover_from_fault (C++ function), 542
mcpwm_operator_register_event_callbacks (C++ function), 542
mcpwm_operator_set_brake_on_fault (C++ function), 541
mcpwm_soft_fault_activate (C++ function), 555
mcpwm_soft_fault_config_t (C++ struct), 556
mcpwm_soft_sync_activate (C++ function), 557
mcpwm_soft_sync_config_t (C++ struct), 558
mcpwm_sync_handle_t (C++ type), 566
mcpwm_timer_clock_source_t (C++ type), 568
mcpwm_timer_config_t (C++ struct), 539
mcpwm_timer_config_t::clk_src (C++ member), 540
mcpwm_timer_config_t::count_mode (C++ member), 540
mcpwm_timer_config_t::flags (C++ member), 540
mcpwm_timer_config_t::group_id (C++ member), 540
mcpwm_timer_config_t::intr_priority (C++ member), 540
mcpwm_timer_config_t::period_ticks (C++ member), 540
mcpwm_timer_config_t::resolution_hz (C++ member), 540
mcpwm_timer_config_t::update_period_on_empty (C++ member), 540
mcpwm_timer_config_t::update_period_on_sync (C++ member), 540
mcpwm_timer_count_mode_t (C++ enum), 568
mcpwm_timer_count_mode_t::MCPWM_TIMER_COUNT_MODE_ (C++ enumerator), 568
mcpwm_timer_count_mode_t::MCPWM_TIMER_COUNT_MODE_ (C++ enumerator), 568
mcpwm_timer_count_mode_t::MCPWM_TIMER_COUNT_MODE_ (C++ enumerator), 568
mcpwm_timer_count_mode_t::MCPWM_TIMER_COUNT_MODE_ (C++ enumerator), 569
mcpwm_timer_direction_t (C++ enum), 568
mcpwm_timer_direction_t::MCPWM_TIMER_DIRECTION_DO (C++ enumerator), 568
mcpwm_timer_direction_t::MCPWM_TIMER_DIRECTION_UP (C++ enumerator), 568
mcpwm_timer_disable (C++ function), 538
mcpwm_timer_enable (C++ function), 538
mcpwm_timer_event_callbacks_t (C++ struct), 539
mcpwm_timer_event_callbacks_t::on_empty (C++ member), 539
mcpwm_timer_event_callbacks_t::on_full (C++ member), 539
mcpwm_timer_event_callbacks_t::on_stop (C++ member), 539
mcpwm_timer_event_cb_t (C++ type), 567
mcpwm_timer_event_data_t (C++ struct), 565
mcpwm_timer_event_data_t::count_value (C++ member), 565
mcpwm_timer_event_data_t::direction (C++ member), 565
mcpwm_timer_event_t (C++ enum), 568
mcpwm_timer_event_t::MCPWM_TIMER_EVENT_EMPTY (C++ enumerator), 568
mcpwm_timer_event_t::MCPWM_TIMER_EVENT_FULL (C++ enumerator), 568
mcpwm_timer_event_t::MCPWM_TIMER_EVENT_INVALID (C++ enumerator), 568
mcpwm_timer_handle_t (C++ type), 566
mcpwm_timer_register_event_callbacks (C++ function), 538
mcpwm_timer_set_period (C++ function), 537
mcpwm_timer_set_phase_on_sync (C++ function), 539
mcpwm_timer_start_stop (C++ function), 538
mcpwm_timer_start_stop_cmd_t (C++ enum), 569
mcpwm_timer_start_stop_cmd_t::MCPWM_TIMER_START_N (C++ enumerator), 569
mcpwm_timer_start_stop_cmd_t::MCPWM_TIMER_START_S

(C++ *enumerator*), 569
mcpwm_timer_start_stop_cmd_t::MCPWM_TIMER_START_STOP_FULL (C++ *enumerator*), 569
mcpwm_timer_start_stop_cmd_t::MCPWM_TIMER_START_STOP_EMPTY (C++ *enumerator*), 569
mcpwm_timer_start_stop_cmd_t::MCPWM_TIMER_STOP_FULL (C++ *enumerator*), 569
mcpwm_timer_sync_phase_config_t (C++ *struct*), 540
mcpwm_timer_sync_phase_config_t::count_value (C++ *member*), 540
mcpwm_timer_sync_phase_config_t::direction (C++ *member*), 540
mcpwm_timer_sync_phase_config_t::sync_src (C++ *member*), 540
mcpwm_timer_sync_src_config_t (C++ *struct*), 558
mcpwm_timer_sync_src_config_t::flags (C++ *member*), 558
mcpwm_timer_sync_src_config_t::propagate_input_sync (C++ *member*), 558
mcpwm_timer_sync_src_config_t::timer_event (C++ *member*), 558
MessageBufferHandle_t (C++ *type*), 1545
MQTT_ERROR_TYPE_ESP_TLS (C *macro*), 49
multi_heap_aligned_alloc (C++ *function*), 1584
multi_heap_aligned_alloc_offs (C++ *function*), 1586
multi_heap_aligned_free (C++ *function*), 1584
multi_heap_check (C++ *function*), 1585
multi_heap_dump (C++ *function*), 1585
multi_heap_free (C++ *function*), 1584
multi_heap_free_size (C++ *function*), 1585
multi_heap_get_allocated_size (C++ *function*), 1585
multi_heap_get_info (C++ *function*), 1586
multi_heap_handle_t (C++ *type*), 1587
multi_heap_info_t (C++ *struct*), 1587
multi_heap_info_t::allocated_blocks (C++ *member*), 1587
multi_heap_info_t::free_blocks (C++ *member*), 1587
multi_heap_info_t::largest_free_block (C++ *member*), 1587
multi_heap_info_t::minimum_free_bytes (C++ *member*), 1587
multi_heap_info_t::total_allocated_bytes (C++ *member*), 1587
multi_heap_info_t::total_blocks (C++ *member*), 1587
multi_heap_info_t::total_free_bytes (C++ *member*), 1587
multi_heap_malloc (C++ *function*), 1584
multi_heap_minimum_free_size (C++ *function*), 1586
multi_heap_realloc (C++ *function*), 1584
multi_heap_register (C++ *function*), 1585
multi_heap_restore_minimum_free_bytes (C++ *function*), 1586
multi_heap_set_lock (C++ *function*), 1585
multi_heap_walk (C++ *function*), 1586
multi_heap_walker_cb_t (C++ *type*), 1587

N

nvs_close (C++ *function*), 1284
nvs_commit (C++ *function*), 1284
NVS_DEFAULT_PART_NAME (C *macro*), 1289
nvs_entry_find (C++ *function*), 1285
nvs_entry_find_in_handle (C++ *function*), 1286
nvs_entry_info (C++ *function*), 1287
nvs_entry_info_t (C++ *struct*), 1287
nvs_entry_info_t::key (C++ *member*), 1287
nvs_entry_info_t::namespace_name (C++ *member*), 1287
nvs_entry_info_t::type (C++ *member*), 1287
nvs_entry_next (C++ *function*), 1286
nvs_erase_all (C++ *function*), 1283
nvs_erase_key (C++ *function*), 1283
nvs_find_key (C++ *function*), 1283
nvs_flash_deinit (C++ *function*), 1274
nvs_flash_deinit_partition (C++ *function*), 1274
nvs_flash_erase (C++ *function*), 1274
nvs_flash_erase_partition (C++ *function*), 1275
nvs_flash_erase_partition_ptr (C++ *function*), 1275
nvs_flash_generate_keys (C++ *function*), 1276
nvs_flash_generate_keys_t (C++ *type*), 1278
nvs_flash_generate_keys_v2 (C++ *function*), 1276
nvs_flash_get_default_security_scheme (C++ *function*), 1276
nvs_flash_init (C++ *function*), 1273
nvs_flash_init_partition (C++ *function*), 1274
nvs_flash_init_partition_ptr (C++ *function*), 1274
nvs_flash_read_cfg_t (C++ *type*), 1278
nvs_flash_read_security_cfg (C++ *function*), 1276
nvs_flash_read_security_cfg_v2 (C++ *function*), 1277
nvs_flash_register_security_scheme (C++ *function*), 1276
nvs_flash_secure_init (C++ *function*), 1275
nvs_flash_secure_init_partition (C++ *function*), 1275
nvs_get_blob (C++ *function*), 1281
nvs_get_i16 (C++ *function*), 1280

nvs_get_i32 (C++ function), 1280
nvs_get_i64 (C++ function), 1280
nvs_get_i8 (C++ function), 1279
nvs_get_stats (C++ function), 1284
nvs_get_str (C++ function), 1280
nvs_get_u16 (C++ function), 1280
nvs_get_u32 (C++ function), 1280
nvs_get_u64 (C++ function), 1280
nvs_get_u8 (C++ function), 1280
nvs_get_used_entry_count (C++ function), 1285
NVS_GUARD_SYSVIEW_MACRO_EXPANSION_POP (C macro), 1290
NVS_GUARD_SYSVIEW_MACRO_EXPANSION_PUSH (C macro), 1290
nvs_handle (C++ type), 1290
nvs_handle_t (C++ type), 1290
nvs_iterator_t (C++ type), 1290
NVS_KEY_NAME_MAX_SIZE (C macro), 1289
NVS_KEY_SIZE (C macro), 1277
NVS_NS_NAME_MAX_SIZE (C macro), 1289
nvs_open (C++ function), 1281
nvs_open_from_partition (C++ function), 1282
nvs_open_mode (C++ type), 1290
nvs_open_mode_t (C++ enum), 1290
nvs_open_mode_t::NVS_READONLY (C++ enumerator), 1290
nvs_open_mode_t::NVS_READWRITE (C++ enumerator), 1290
NVS_PART_NAME_MAX_SIZE (C macro), 1289
nvs_release_iterator (C++ function), 1287
nvs_sec_cfg_t (C++ struct), 1277
nvs_sec_cfg_t::eky (C++ member), 1277
nvs_sec_cfg_t::tky (C++ member), 1277
nvs_sec_config_flash_enc_t (C++ struct), 1296
nvs_sec_config_flash_enc_t::nvs_keys_part (C++ member), 1296
nvs_sec_config_hmac_t (C++ struct), 1296
nvs_sec_config_hmac_t::hmac_key_id (C++ member), 1296
NVS_SEC_PROVIDER_CFG_FLASH_ENC_DEFAULT (C macro), 1296
NVS_SEC_PROVIDER_CFG_HMAC_DEFAULT (C macro), 1296
nvs_sec_provider_deregister (C++ function), 1295
nvs_sec_provider_register_flash_enc (C++ function), 1295
nvs_sec_provider_register_hmac (C++ function), 1295
nvs_sec_scheme_id_t (C++ enum), 1297
nvs_sec_scheme_id_t::NVS_SEC_SCHEME_FLASH_ENC (C++ enumerator), 1297
nvs_sec_scheme_id_t::NVS_SEC_SCHEME_HMAC (C++ enumerator), 1297
nvs_sec_scheme_id_t::NVS_SEC_SCHEME_MAX (C++ enumerator), 1297
nvs_sec_scheme_t (C++ struct), 1277
nvs_sec_scheme_t::nvs_flash_key_gen (C++ member), 1277
nvs_sec_scheme_t::nvs_flash_read_cfg (C++ member), 1277
nvs_sec_scheme_t::scheme_data (C++ member), 1277
nvs_sec_scheme_t::scheme_id (C++ member), 1277
nvs_set_blob (C++ function), 1282
nvs_set_i16 (C++ function), 1278
nvs_set_i32 (C++ function), 1279
nvs_set_i64 (C++ function), 1279
nvs_set_i8 (C++ function), 1278
nvs_set_str (C++ function), 1279
nvs_set_u16 (C++ function), 1278
nvs_set_u32 (C++ function), 1279
nvs_set_u64 (C++ function), 1279
nvs_set_u8 (C++ function), 1278
nvs_stats_t (C++ struct), 1287
nvs_stats_t::available_entries (C++ member), 1287
nvs_stats_t::free_entries (C++ member), 1287
nvs_stats_t::namespace_count (C++ member), 1288
nvs_stats_t::total_entries (C++ member), 1288
nvs_stats_t::used_entries (C++ member), 1287
nvs_type_t (C++ enum), 1290
nvs_type_t::NVS_TYPE_ANY (C++ enumerator), 1291
nvs_type_t::NVS_TYPE_BLOB (C++ enumerator), 1291
nvs_type_t::NVS_TYPE_I16 (C++ enumerator), 1290
nvs_type_t::NVS_TYPE_I32 (C++ enumerator), 1290
nvs_type_t::NVS_TYPE_I64 (C++ enumerator), 1291
nvs_type_t::NVS_TYPE_I8 (C++ enumerator), 1290
nvs_type_t::NVS_TYPE_STR (C++ enumerator), 1291
nvs_type_t::NVS_TYPE_U16 (C++ enumerator), 1290
nvs_type_t::NVS_TYPE_U32 (C++ enumerator), 1290
nvs_type_t::NVS_TYPE_U64 (C++ enumerator), 1291
nvs_type_t::NVS_TYPE_U8 (C++ enumerator), 1290

O

OTA_SIZE_UNKNOWN (C macro), 1665
OTA_WITH_SEQUENTIAL_WRITES (C macro), 1665

P

parlio_bit_pack_order_t (*C++ enum*), 585
parlio_bit_pack_order_t::PARLIO_BIT_PACK_ORDER_LSB (*C++ enumerator*), 585
parlio_bit_pack_order_t::PARLIO_BIT_PACK_ORDER_MSB (*C++ enumerator*), 585
parlio_clock_source_t (*C++ type*), 584
parlio_del_rx_delimiter (*C++ function*), 577
parlio_del_rx_unit (*C++ function*), 576
parlio_del_tx_unit (*C++ function*), 571
parlio_new_rx_level_delimiter (*C++ function*), 576
parlio_new_rx_pulse_delimiter (*C++ function*), 576
parlio_new_rx_soft_delimiter (*C++ function*), 577
parlio_new_rx_unit (*C++ function*), 576
parlio_new_tx_unit (*C++ function*), 571
parlio_receive_config_t (*C++ struct*), 582
parlio_receive_config_t::delimiter (*C++ member*), 582
parlio_receive_config_t::flags (*C++ member*), 582
parlio_receive_config_t::indirect_mount (*C++ member*), 582
parlio_receive_config_t::partial_rx_en (*C++ member*), 582
parlio_rx_callback_t (*C++ type*), 583
parlio_rx_delimiter_handle_t (*C++ type*), 584
parlio_rx_event_callbacks_t (*C++ struct*), 583
parlio_rx_event_callbacks_t::on_partial_receive (*C++ member*), 583
parlio_rx_event_callbacks_t::on_receive_done (*C++ member*), 583
parlio_rx_event_callbacks_t::on_timeout (*C++ member*), 583
parlio_rx_event_data_t (*C++ struct*), 583
parlio_rx_event_data_t::data (*C++ member*), 583
parlio_rx_event_data_t::delimiter (*C++ member*), 583
parlio_rx_event_data_t::recv_bytes (*C++ member*), 583
parlio_rx_level_delimiter_config_t (*C++ struct*), 580
parlio_rx_level_delimiter_config_t::active_low_en (*C++ member*), 581
parlio_rx_level_delimiter_config_t::bit_pack_order (*C++ member*), 580
parlio_rx_level_delimiter_config_t::eof_data_len (*C++ member*), 581
parlio_rx_level_delimiter_config_t::flags (*C++ member*), 581
parlio_rx_level_delimiter_config_t::sample_edge (*C++ member*), 580
parlio_rx_level_delimiter_config_t::timeout_ticks (*C++ member*), 581
parlio_rx_level_delimiter_config_t::valid_sig_lin (*C++ member*), 580
parlio_rx_pulse_delimiter_config_t (*C++ struct*), 581
parlio_rx_pulse_delimiter_config_t::bit_pack_orde (*C++ member*), 581
parlio_rx_pulse_delimiter_config_t::end_bit_inclu (*C++ member*), 581
parlio_rx_pulse_delimiter_config_t::eof_data_len (*C++ member*), 581
parlio_rx_pulse_delimiter_config_t::flags (*C++ member*), 582
parlio_rx_pulse_delimiter_config_t::has_end_pulse (*C++ member*), 581
parlio_rx_pulse_delimiter_config_t::pulse_invert (*C++ member*), 582
parlio_rx_pulse_delimiter_config_t::sample_edge (*C++ member*), 581
parlio_rx_pulse_delimiter_config_t::start_bit_inc (*C++ member*), 581
parlio_rx_pulse_delimiter_config_t::timeout_ticks (*C++ member*), 581
parlio_rx_pulse_delimiter_config_t::valid_sig_lin (*C++ member*), 581
parlio_rx_soft_delimiter_config_t (*C++ struct*), 582
parlio_rx_soft_delimiter_config_t::bit_pack_order (*C++ member*), 582
parlio_rx_soft_delimiter_config_t::eof_data_len (*C++ member*), 582
parlio_rx_soft_delimiter_config_t::sample_edge (*C++ member*), 582
parlio_rx_soft_delimiter_config_t::timeout_ticks (*C++ member*), 582
parlio_rx_soft_delimiter_start_stop (*C++ function*), 577
parlio_rx_unit_config_t (*C++ struct*), 579
parlio_rx_unit_config_t::clk_gate_en (*C++ member*), 580
parlio_rx_unit_config_t::clk_in_gpio_num (*C++ member*), 579
parlio_rx_unit_config_t::clk_out_gpio_num (*C++ member*), 580
parlio_rx_unit_config_t::clk_src (*C++ member*), 579
parlio_rx_unit_config_t::data_gpio_nums (*C++ member*), 580
parlio_rx_unit_config_t::data_width (*C++ member*), 579
parlio_rx_unit_config_t::exp_clk_freq_hz (*C++ member*), 579
parlio_rx_unit_config_t::ext_clk_freq_hz (*C++ member*), 579
parlio_rx_unit_config_t::flags (*C++ member*), 580
parlio_rx_unit_config_t::free_clk (*C++ member*), 580

parlio_rx_unit_config_t::io_loop_back (C++ member), 580
parlio_rx_unit_config_t::io_no_init (C++ member), 580
parlio_rx_unit_config_t::max_recv_size (C++ member), 579
parlio_rx_unit_config_t::trans_queue_depth (C++ member), 579
parlio_rx_unit_config_t::valid_gpio_num (C++ member), 580
parlio_rx_unit_disable (C++ function), 578
parlio_rx_unit_enable (C++ function), 577
parlio_rx_unit_handle_t (C++ type), 584
PARLIO_RX_UNIT_MAX_DATA_WIDTH (C macro), 584
parlio_rx_unit_receive (C++ function), 578
parlio_rx_unit_register_event_callbacks (C++ function), 579
parlio_rx_unit_wait_all_done (C++ function), 578
parlio_sample_edge_t (C++ enum), 584
parlio_sample_edge_t::PARLIO_SAMPLE_EDGE_NEG (C++ enumerator), 584
parlio_sample_edge_t::PARLIO_SAMPLE_EDGE_POS (C++ enumerator), 584
parlio_transmit_config_t (C++ struct), 575
parlio_transmit_config_t::flags (C++ member), 575
parlio_transmit_config_t::idle_value (C++ member), 575
parlio_transmit_config_t::queue_nonblocking (C++ member), 575
parlio_tx_done_callback_t (C++ type), 575
parlio_tx_done_event_data_t (C++ struct), 574
parlio_tx_event_callbacks_t (C++ struct), 574
parlio_tx_event_callbacks_t::on_trans_done (C++ member), 575
parlio_tx_unit_config_t (C++ struct), 573
parlio_tx_unit_config_t::bit_pack_order (C++ member), 574
parlio_tx_unit_config_t::clk_gate_en (C++ member), 574
parlio_tx_unit_config_t::clk_in_gpio_num (C++ member), 573
parlio_tx_unit_config_t::clk_out_gpio_num (C++ member), 574
parlio_tx_unit_config_t::clk_src (C++ member), 573
parlio_tx_unit_config_t::data_gpio_nums (C++ member), 574
parlio_tx_unit_config_t::data_width (C++ member), 573
parlio_tx_unit_config_t::dma_burst_size (C++ member), 574
parlio_tx_unit_config_t::flags (C++ member), 574
parlio_tx_unit_config_t::input_clk_src_freq_hz (C++ member), 573
parlio_tx_unit_config_t::io_loop_back (C++ member), 574
parlio_tx_unit_config_t::max_transfer_size (C++ member), 574
parlio_tx_unit_config_t::output_clk_freq_hz (C++ member), 573
parlio_tx_unit_config_t::sample_edge (C++ member), 574
parlio_tx_unit_config_t::trans_queue_depth (C++ member), 574
parlio_tx_unit_config_t::valid_gpio_num (C++ member), 574
parlio_tx_unit_disable (C++ function), 572
parlio_tx_unit_enable (C++ function), 571
parlio_tx_unit_handle_t (C++ type), 584
PARLIO_TX_UNIT_MAX_DATA_WIDTH (C macro), 584
parlio_tx_unit_register_event_callbacks (C++ function), 572
parlio_tx_unit_transmit (C++ function), 572
parlio_tx_unit_wait_all_done (C++ function), 573
pcnt_chan_config_t (C++ struct), 598
pcnt_chan_config_t::edge_gpio_num (C++ member), 598
pcnt_chan_config_t::flags (C++ member), 598
pcnt_chan_config_t::invert_edge_input (C++ member), 598
pcnt_chan_config_t::invert_level_input (C++ member), 598
pcnt_chan_config_t::io_loop_back (C++ member), 598
pcnt_chan_config_t::level_gpio_num (C++ member), 598
pcnt_chan_config_t::virt_edge_io_level (C++ member), 598
pcnt_chan_config_t::virt_level_io_level (C++ member), 598
pcnt_channel_edge_action_t (C++ enum), 600
pcnt_channel_edge_action_t::PCNT_CHANNEL_EDGE_ACT (C++ enumerator), 600
pcnt_channel_edge_action_t::PCNT_CHANNEL_EDGE_ACT (C++ enumerator), 600
pcnt_channel_edge_action_t::PCNT_CHANNEL_EDGE_ACT (C++ enumerator), 600
pcnt_channel_handle_t (C++ type), 599
pcnt_channel_level_action_t (C++ enum), 599
pcnt_channel_level_action_t::PCNT_CHANNEL_LEVEL_A (C++ enumerator), 600
pcnt_channel_level_action_t::PCNT_CHANNEL_LEVEL_A (C++ enumerator), 599
pcnt_channel_level_action_t::PCNT_CHANNEL_LEVEL_A (C++ enumerator), 599

pcnt_channel_set_edge_action (C++ function), 596
pcnt_channel_set_level_action (C++ function), 596
pcnt_clear_signal_config_t (C++ struct), 598
pcnt_clear_signal_config_t::clear_signal_gpio_num (C++ member), 599
pcnt_clear_signal_config_t::flags (C++ member), 599
pcnt_clear_signal_config_t::invert_clear_signal (C++ member), 599
pcnt_clear_signal_config_t::io_loop_back (C++ member), 599
pcnt_del_channel (C++ function), 596
pcnt_del_unit (C++ function), 591
pcnt_event_callbacks_t (C++ struct), 597
pcnt_event_callbacks_t::on_reach (C++ member), 597
pcnt_glitch_filter_config_t (C++ struct), 598
pcnt_glitch_filter_config_t::max_glitch_ns (C++ member), 598
pcnt_new_channel (C++ function), 595
pcnt_new_unit (C++ function), 591
pcnt_unit_add_watch_point (C++ function), 595
pcnt_unit_clear_count (C++ function), 594
pcnt_unit_config_t (C++ struct), 597
pcnt_unit_config_t::accum_count (C++ member), 597
pcnt_unit_config_t::flags (C++ member), 598
pcnt_unit_config_t::high_limit (C++ member), 597
pcnt_unit_config_t::intr_priority (C++ member), 597
pcnt_unit_config_t::low_limit (C++ member), 597
pcnt_unit_disable (C++ function), 593
pcnt_unit_enable (C++ function), 592
pcnt_unit_get_count (C++ function), 594
pcnt_unit_handle_t (C++ type), 599
pcnt_unit_register_event_callbacks (C++ function), 595
pcnt_unit_remove_watch_point (C++ function), 595
pcnt_unit_set_clear_signal (C++ function), 592
pcnt_unit_set_glitch_filter (C++ function), 591
pcnt_unit_start (C++ function), 593
pcnt_unit_stop (C++ function), 593
pcnt_unit_zero_cross_mode_t (C++ enum), 600
pcnt_unit_zero_cross_mode_t::PCNT_UNIT_ZERO_CROSS_NEG_ZERO (C++ enumerator), 600
pcnt_unit_zero_cross_mode_t::PCNT_UNIT_ZERO_CROSS_POS_ZERO (C++ enumerator), 600
pcnt_unit_zero_cross_mode_t::PCNT_UNIT_ZERO_CROSS_ZERO (C++ enumerator), 600
pcnt_watch_cb_t (C++ type), 599
pcnt_watch_event_data_t (C++ struct), 597
pcnt_watch_event_data_t::watch_point_value (C++ member), 597
pcnt_watch_event_data_t::zero_cross_mode (C++ member), 597
pcQueueGetName (C++ function), 1474
pcTaskGetName (C++ function), 1449
pcTimerGetName (C++ function), 1508
PendedFunction_t (C++ type), 1518
phy_802_3_t (C++ struct), 205
phy_802_3_t::addr (C++ member), 205
phy_802_3_t::autonego_timeout_ms (C++ member), 205
phy_802_3_t::eth (C++ member), 205
phy_802_3_t::link_status (C++ member), 205
phy_802_3_t::parent (C++ member), 205
phy_802_3_t::reset_gpio_num (C++ member), 205
phy_802_3_t::reset_timeout_ms (C++ member), 205
ppa_alpha_update_mode_t (C++ enum), 613
ppa_alpha_update_mode_t::PPA_ALPHA_FIX_VALUE (C++ enumerator), 613
ppa_alpha_update_mode_t::PPA_ALPHA_INVERT (C++ enumerator), 613
ppa_alpha_update_mode_t::PPA_ALPHA_NO_CHANGE (C++ enumerator), 613
ppa_alpha_update_mode_t::PPA_ALPHA_SCALE (C++ enumerator), 613
ppa_blend_color_mode_t (C++ enum), 612
ppa_blend_color_mode_t::PPA_BLEND_COLOR_MODE_A4 (C++ enumerator), 613
ppa_blend_color_mode_t::PPA_BLEND_COLOR_MODE_A8 (C++ enumerator), 613
ppa_blend_color_mode_t::PPA_BLEND_COLOR_MODE_ARGB (C++ enumerator), 612
ppa_blend_color_mode_t::PPA_BLEND_COLOR_MODE_RGB565 (C++ enumerator), 613
ppa_blend_color_mode_t::PPA_BLEND_COLOR_MODE_RGB888 (C++ enumerator), 613
ppa_blend_oper_config_t (C++ struct), 608
ppa_blend_oper_config_t::bg_alpha_fix_val (C++ member), 609
ppa_blend_oper_config_t::bg_alpha_scale_ratio (C++ member), 609
ppa_blend_oper_config_t::bg_alpha_update_mode (C++ member), 609
ppa_blend_oper_config_t::bg_byte_swap (C++ member), 609
ppa_blend_oper_config_t::bg_ck_en (C++ member), 609

ppa_blend_oper_config_t::bg_ck_rgb_high_thres (C++ member), 609
ppa_blend_oper_config_t::bg_ck_rgb_low_thres (C++ member), 609
ppa_blend_oper_config_t::bg_rgb_swap (C++ member), 608
ppa_blend_oper_config_t::ck_reverse_bg2fg (C++ member), 610
ppa_blend_oper_config_t::ck_rgb_default_val (C++ member), 610
ppa_blend_oper_config_t::fg_alpha_fix_val (C++ member), 609
ppa_blend_oper_config_t::fg_alpha_scale_ratio (C++ member), 609
ppa_blend_oper_config_t::fg_alpha_update_mode (C++ member), 609
ppa_blend_oper_config_t::fg_byte_swap (C++ member), 609
ppa_blend_oper_config_t::fg_ck_en (C++ member), 609
ppa_blend_oper_config_t::fg_ck_rgb_high_thres (C++ member), 610
ppa_blend_oper_config_t::fg_ck_rgb_low_thres (C++ member), 610
ppa_blend_oper_config_t::fg_fix_rgb_val (C++ member), 609
ppa_blend_oper_config_t::fg_rgb_swap (C++ member), 609
ppa_blend_oper_config_t::in_bg (C++ member), 608
ppa_blend_oper_config_t::in_fg (C++ member), 608
ppa_blend_oper_config_t::mode (C++ member), 610
ppa_blend_oper_config_t::out (C++ member), 608
ppa_blend_oper_config_t::user_data (C++ member), 610
ppa_client_config_t (C++ struct), 605
ppa_client_config_t::data_burst_length (C++ member), 605
ppa_client_config_t::max_pending_trans_num (C++ member), 605
ppa_client_config_t::oper_type (C++ member), 605
ppa_client_handle_t (C++ type), 611
ppa_client_register_event_callbacks (C++ function), 604
ppa_color_conv_std_rgb_yuv_t (C++ enum), 613
ppa_color_conv_std_rgb_yuv_t::PPA_COLOR_CONV_STD_RGB_YUV_BT601 (C++ enumerator), 614
ppa_color_conv_std_rgb_yuv_t::PPA_COLOR_CONV_STD_RGB_YUV_BT709 (C++ enumerator), 614
ppa_color_range_t (C++ enum), 614
ppa_color_range_t::PPA_COLOR_RANGE_FULL (C++ enumerator), 614
ppa_color_range_t::PPA_COLOR_RANGE_LIMIT (C++ enumerator), 614
ppa_data_burst_length_t (C++ enum), 614
ppa_data_burst_length_t::PPA_DATA_BURST_LENGTH_128 (C++ enumerator), 614
ppa_data_burst_length_t::PPA_DATA_BURST_LENGTH_16 (C++ enumerator), 614
ppa_data_burst_length_t::PPA_DATA_BURST_LENGTH_32 (C++ enumerator), 614
ppa_data_burst_length_t::PPA_DATA_BURST_LENGTH_64 (C++ enumerator), 614
ppa_data_burst_length_t::PPA_DATA_BURST_LENGTH_8 (C++ enumerator), 614
ppa_do_blend (C++ function), 604
ppa_do_fill (C++ function), 604
ppa_do_scale_rotate_mirror (C++ function), 604
ppa_engine_type_t (C++ enum), 612
ppa_engine_type_t::PPA_ENGINE_TYPE_BLEND (C++ enumerator), 612
ppa_engine_type_t::PPA_ENGINE_TYPE_SRM (C++ enumerator), 612
ppa_event_callback_t (C++ type), 611
ppa_event_callbacks_t (C++ struct), 605
ppa_event_callbacks_t::on_trans_done (C++ member), 605
ppa_event_data_t (C++ struct), 605
ppa_fill_color_mode_t (C++ enum), 613
ppa_fill_color_mode_t::PPA_FILL_COLOR_MODE_ARGB8888 (C++ enumerator), 613
ppa_fill_color_mode_t::PPA_FILL_COLOR_MODE_RGB565 (C++ enumerator), 613
ppa_fill_color_mode_t::PPA_FILL_COLOR_MODE_RGB888 (C++ enumerator), 613
ppa_fill_oper_config_t (C++ struct), 610
ppa_fill_oper_config_t::fill_argb_color (C++ member), 610
ppa_fill_oper_config_t::fill_block_h (C++ member), 610
ppa_fill_oper_config_t::fill_block_w (C++ member), 610
ppa_fill_oper_config_t::mode (C++ member), 610
ppa_fill_oper_config_t::out (C++ member), 610
ppa_fill_oper_config_t::user_data (C++ member), 610
ppa_in_pic_blk_config_t (C++ struct), 605
ppa_in_pic_blk_config_t::blend_cm (C++ member), 606
ppa_in_pic_blk_config_t::block_h (C++ member), 606
ppa_in_pic_blk_config_t::block_offset_x (C++ member), 606
ppa_in_pic_blk_config_t::block_offset_y (C++ member), 606
ppa_in_pic_blk_config_t::block_w (C++ member), 606
ppa_in_pic_blk_config_t::buffer (C++

member), 605
ppa_in_pic_blk_config_t::fill_cm (*C++ member*), 606
ppa_in_pic_blk_config_t::pic_h (*C++ member*), 606
ppa_in_pic_blk_config_t::pic_w (*C++ member*), 605
ppa_in_pic_blk_config_t::srm_cm (*C++ member*), 606
ppa_in_pic_blk_config_t::yuv_range (*C++ member*), 606
ppa_in_pic_blk_config_t::yuv_std (*C++ member*), 606
ppa_operation_t (*C++ enum*), 611
ppa_operation_t::PPA_OPERATION_BLEND (*C++ enumerator*), 611
ppa_operation_t::PPA_OPERATION_FILL (*C++ enumerator*), 611
ppa_operation_t::PPA_OPERATION_INVALID (*C++ enumerator*), 611
ppa_operation_t::PPA_OPERATION_SRM (*C++ enumerator*), 611
ppa_out_pic_blk_config_t (*C++ struct*), 606
ppa_out_pic_blk_config_t::blend_cm (*C++ member*), 607
ppa_out_pic_blk_config_t::block_offset_x (*C++ member*), 607
ppa_out_pic_blk_config_t::block_offset_y (*C++ member*), 607
ppa_out_pic_blk_config_t::buffer (*C++ member*), 606
ppa_out_pic_blk_config_t::buffer_size (*C++ member*), 606
ppa_out_pic_blk_config_t::fill_cm (*C++ member*), 607
ppa_out_pic_blk_config_t::pic_h (*C++ member*), 607
ppa_out_pic_blk_config_t::pic_w (*C++ member*), 606
ppa_out_pic_blk_config_t::srm_cm (*C++ member*), 607
ppa_out_pic_blk_config_t::yuv_range (*C++ member*), 607
ppa_out_pic_blk_config_t::yuv_std (*C++ member*), 607
ppa_register_client (*C++ function*), 603
ppa_srm_color_mode_t (*C++ enum*), 612
ppa_srm_color_mode_t::PPA_SRM_COLOR_MODE_ARGB8888 (*C++ enumerator*), 612
ppa_srm_color_mode_t::PPA_SRM_COLOR_MODE_RGB565 (*C++ enumerator*), 612
ppa_srm_color_mode_t::PPA_SRM_COLOR_MODE_RGB888 (*C++ enumerator*), 612
ppa_srm_color_mode_t::PPA_SRM_COLOR_MODE_YUV420 (*C++ enumerator*), 612
ppa_srm_color_mode_t::PPA_SRM_COLOR_MODE_YUV444 (*C++ enumerator*), 612
ppa_srm_oper_config_t (*C++ struct*), 607
ppa_srm_oper_config_t::alpha_fix_val (*C++ member*), 608
ppa_srm_oper_config_t::alpha_scale_ratio (*C++ member*), 608
ppa_srm_oper_config_t::alpha_update_mode (*C++ member*), 608
ppa_srm_oper_config_t::byte_swap (*C++ member*), 608
ppa_srm_oper_config_t::in (*C++ member*), 607
ppa_srm_oper_config_t::mirror_x (*C++ member*), 608
ppa_srm_oper_config_t::mirror_y (*C++ member*), 608
ppa_srm_oper_config_t::mode (*C++ member*), 608
ppa_srm_oper_config_t::out (*C++ member*), 607
ppa_srm_oper_config_t::rgb_swap (*C++ member*), 608
ppa_srm_oper_config_t::rotation_angle (*C++ member*), 607
ppa_srm_oper_config_t::scale_x (*C++ member*), 607
ppa_srm_oper_config_t::scale_y (*C++ member*), 607
ppa_srm_oper_config_t::user_data (*C++ member*), 608
ppa_srm_rotation_angle_t (*C++ enum*), 612
ppa_srm_rotation_angle_t::PPA_SRM_ROTATION_ANGLE_ (*C++ enumerator*), 612
ppa_srm_rotation_angle_t::PPA_SRM_ROTATION_ANGLE_ (*C++ enumerator*), 612
ppa_srm_rotation_angle_t::PPA_SRM_ROTATION_ANGLE_ (*C++ enumerator*), 612
ppa_srm_rotation_angle_t::PPA_SRM_ROTATION_ANGLE_ (*C++ enumerator*), 612
ppa_trans_mode_t (*C++ enum*), 611
ppa_trans_mode_t::PPA_TRANS_MODE_BLOCKING (*C++ enumerator*), 611
ppa_trans_mode_t::PPA_TRANS_MODE_NON_BLOCKING (*C++ enumerator*), 611
ppa_unregister_client (*C++ function*), 603
print_class_descriptor_cb (*C++ type*), 856
protocomm_add_endpoint (*C++ function*), 1241
protocomm_close_session (*C++ function*), 1242
protocomm_delete (*C++ function*), 1241
protocomm_http_server_config_t (*C++ struct*), 1252
protocomm_http_server_config_t::port (*C++ member*), 1252
protocomm_http_server_config_t::stack_size (*C++ member*), 1252
protocomm_http_server_config_t::task_priority (*C++ member*), 1252
protocomm_httpd_config_data_t (*C++ union*), 1251

protocomm_httpd_config_data_t::config (C++ member), 1251
protocomm_httpd_config_data_t::handle (C++ member), 1251
protocomm_httpd_config_t (C++ struct), 1252
protocomm_httpd_config_t::data (C++ member), 1252
protocomm_httpd_config_t::ext_handle_provided (C++ member), 1252
PROTOCOMM_HTTPD_DEFAULT_CONFIG (C macro), 1252
protocomm_httpd_start (C++ function), 1251
protocomm_httpd_stop (C++ function), 1251
protocomm_new (C++ function), 1241
protocomm_open_session (C++ function), 1242
protocomm_remove_endpoint (C++ function), 1241
protocomm_req_handle (C++ function), 1242
protocomm_req_handler_t (C++ type), 1244
protocomm_security (C++ struct), 1245
protocomm_security1_params (C++ struct), 1245
protocomm_security1_params::data (C++ member), 1245
protocomm_security1_params::len (C++ member), 1245
protocomm_security1_params_t (C++ type), 1246
protocomm_security2_params (C++ struct), 1245
protocomm_security2_params::salt (C++ member), 1245
protocomm_security2_params::salt_len (C++ member), 1245
protocomm_security2_params::verifier (C++ member), 1245
protocomm_security2_params::verifier_len (C++ member), 1245
protocomm_security2_params_t (C++ type), 1246
protocomm_security::cleanup (C++ member), 1246
protocomm_security::close_transport_session (C++ member), 1246
protocomm_security::decrypt (C++ member), 1246
protocomm_security::encrypt (C++ member), 1246
protocomm_security::init (C++ member), 1246
protocomm_security::new_transport_session (C++ member), 1246
protocomm_security::security_req_handler (C++ member), 1246
protocomm_security::ver (C++ member), 1246
protocomm_security_handle_t (C++ type), 1246
protocomm_security_pop_t (C++ type), 1246
protocomm_security_session_event_t (C++ enum), 1247
protocomm_security_session_event_t::PROTOCOMM_SEC (C++ enumerator), 1247
protocomm_security_session_event_t::PROTOCOMM_SEC (C++ enumerator), 1247
protocomm_security_session_event_t::PROTOCOMM_SEC (C++ enumerator), 1247
protocomm_security_t (C++ type), 1246
protocomm_set_security (C++ function), 1243
protocomm_set_version (C++ function), 1244
protocomm_t (C++ type), 1244
protocomm_unset_security (C++ function), 1243
protocomm_unset_version (C++ function), 1244
psk_hint_key_t (C++ type), 67
psk_key_hint (C++ struct), 62
psk_key_hint::hint (C++ member), 62
psk_key_hint::key (C++ member), 62
psk_key_hint::key_size (C++ member), 62
PTHREAD_STACK_MIN (C macro), 1678
pvTaskGetThreadLocalStoragePointer (C++ function), 1450
pvTimerGetTimerID (C++ function), 1505
pxTaskGetStackStart (C++ function), 1567

Q

QueueHandle_t (C++ type), 1486
QueueSetHandle_t (C++ type), 1486
QueueSetMemberHandle_t (C++ type), 1486

R

RingbufferType_t (C++ enum), 1564
RingbufferType_t::RINGBUF_TYPE_ALLOWSPLIT (C++ enumerator), 1564
RingbufferType_t::RINGBUF_TYPE_BYTEBUF (C++ enumerator), 1564
RingbufferType_t::RINGBUF_TYPE_MAX (C++ enumerator), 1564
RingbufferType_t::RINGBUF_TYPE_NOSPLIT (C++ enumerator), 1564
RingbufHandle_t (C++ type), 1563
rmt_alloc_encoder_mem (C++ function), 637
rmt_apply_carrier (C++ function), 635
rmt_bytes_encoder_config_t (C++ struct), 638
rmt_bytes_encoder_config_t::bit0 (C++ member), 638
rmt_bytes_encoder_config_t::bit1 (C++ member), 638
rmt_bytes_encoder_config_t::flags (C++ member), 639
rmt_bytes_encoder_config_t::msb_first (C++ member), 639
rmt_bytes_encoder_update_config (C++ function), 636

rmt_carrier_config_t (C++ struct), 635
rmt_carrier_config_t::always_on (C++ member), 636
rmt_carrier_config_t::duty_cycle (C++ member), 636
rmt_carrier_config_t::flags (C++ member), 636
rmt_carrier_config_t::frequency_hz (C++ member), 636
rmt_carrier_config_t::polarity_active_low (C++ member), 636
rmt_channel_handle_t (C++ type), 641
rmt_clock_source_t (C++ type), 642
rmt_copy_encoder_config_t (C++ struct), 639
rmt_del_channel (C++ function), 635
rmt_del_encoder (C++ function), 637
rmt_del_sync_manager (C++ function), 629
rmt_disable (C++ function), 635
rmt_enable (C++ function), 635
rmt_encode_simple_cb_t (C++ type), 639
rmt_encode_state_t (C++ enum), 640
rmt_encode_state_t::RMT_ENCODING_COMPLETE (C++ enumerator), 640
rmt_encode_state_t::RMT_ENCODING_MEM_FULL (C++ enumerator), 640
rmt_encode_state_t::RMT_ENCODING_RESET (C++ enumerator), 640
rmt_encoder_handle_t (C++ type), 641
rmt_encoder_reset (C++ function), 637
rmt_encoder_t (C++ struct), 638
rmt_encoder_t::del (C++ member), 638
rmt_encoder_t::encode (C++ member), 638
rmt_encoder_t::reset (C++ member), 638
rmt_new_bytes_encoder (C++ function), 636
rmt_new_copy_encoder (C++ function), 637
rmt_new_rx_channel (C++ function), 632
rmt_new_simple_encoder (C++ function), 637
rmt_new_sync_manager (C++ function), 629
rmt_new_tx_channel (C++ function), 627
rmt_receive (C++ function), 632
rmt_receive_config_t (C++ struct), 634
rmt_receive_config_t::extra_rmt_receive_flags (C++ struct), 634
rmt_receive_config_t::extra_rmt_receive_flags (C++ member), 634
rmt_receive_config_t::flags (C++ member), 634
rmt_receive_config_t::signal_range_max_ns (C++ member), 634
rmt_receive_config_t::signal_range_min_ns (C++ member), 634
rmt_rx_channel_config_t (C++ struct), 633
rmt_rx_channel_config_t::clk_src (C++ member), 633
rmt_rx_channel_config_t::flags (C++ member), 634
rmt_rx_channel_config_t::gpio_num (C++ member), 633
rmt_rx_channel_config_t::intr_priority (C++ member), 633
rmt_rx_channel_config_t::invert_in (C++ member), 633
rmt_rx_channel_config_t::io_loop_back (C++ member), 634
rmt_rx_channel_config_t::mem_block_symbols (C++ member), 633
rmt_rx_channel_config_t::resolution_hz (C++ member), 633
rmt_rx_channel_config_t::with_dma (C++ member), 634
rmt_rx_done_callback_t (C++ type), 641
rmt_rx_done_event_data_t (C++ struct), 640
rmt_rx_done_event_data_t::flags (C++ member), 641
rmt_rx_done_event_data_t::is_last (C++ member), 641
rmt_rx_done_event_data_t::num_symbols (C++ member), 641
rmt_rx_done_event_data_t::received_symbols (C++ member), 641
rmt_rx_event_callbacks_t (C++ struct), 633
rmt_rx_event_callbacks_t::on_recv_done (C++ member), 633
rmt_rx_register_event_callbacks (C++ function), 632
rmt_simple_encoder_config_t (C++ struct), 639
rmt_simple_encoder_config_t::arg (C++ member), 639
rmt_simple_encoder_config_t::callback (C++ member), 639
rmt_simple_encoder_config_t::min_chunk_size (C++ member), 639
rmt_symbol_word_t (C++ union), 642
rmt_symbol_word_t::duration0 (C++ member), 642
rmt_symbol_word_t::duration1 (C++ member), 642
rmt_symbol_word_t::level0 (C++ member), 642
rmt_symbol_word_t::level1 (C++ member), 642
rmt_symbol_word_t::val (C++ member), 642
rmt_symbol_word_t::[anonymous] (C++ member), 642
rmt_sync_manager_config_t (C++ struct), 631
rmt_sync_manager_config_t::array_size (C++ member), 631
rmt_sync_manager_config_t::tx_channel_array (C++ member), 631
rmt_sync_manager_handle_t (C++ type), 641
rmt_sync_reset (C++ function), 629
rmt_transmit (C++ function), 627
rmt_transmit_config_t (C++ struct), 631
rmt_transmit_config_t::eot_level (C++ member), 631

rmt_transmit_config_t::flags (C++ member), 631
rmt_transmit_config_t::loop_count (C++ member), 631
rmt_transmit_config_t::queue_nonblocking (C++ member), 631
rmt_tx_channel_config_t (C++ struct), 630
rmt_tx_channel_config_t::clk_src (C++ member), 630
rmt_tx_channel_config_t::flags (C++ member), 631
rmt_tx_channel_config_t::gpio_num (C++ member), 630
rmt_tx_channel_config_t::intr_priority (C++ member), 630
rmt_tx_channel_config_t::invert_out (C++ member), 630
rmt_tx_channel_config_t::io_loop_back (C++ member), 630
rmt_tx_channel_config_t::io_od_mode (C++ member), 630
rmt_tx_channel_config_t::mem_block_symbols (C++ member), 630
rmt_tx_channel_config_t::resolution_hz (C++ member), 630
rmt_tx_channel_config_t::trans_queue_depth (C++ member), 630
rmt_tx_channel_config_t::with_dma (C++ member), 630
rmt_tx_done_callback_t (C++ type), 641
rmt_tx_done_event_data_t (C++ struct), 640
rmt_tx_done_event_data_t::num_symbols (C++ member), 640
rmt_tx_event_callbacks_t (C++ struct), 629
rmt_tx_event_callbacks_t::on_trans_done (C++ member), 630
rmt_tx_register_event_callbacks (C++ function), 628
rmt_tx_wait_all_done (C++ function), 628
rtc_gpio_deinit (C++ function), 332
rtc_gpio_force_hold_dis_all (C++ function), 335
rtc_gpio_force_hold_en_all (C++ function), 335
rtc_gpio_get_drive_capability (C++ function), 334
rtc_gpio_get_level (C++ function), 332
rtc_gpio_hold_dis (C++ function), 335
rtc_gpio_hold_en (C++ function), 334
rtc_gpio_init (C++ function), 332
rtc_gpio_iomux_func_sel (C++ function), 334
RTC_GPIO_IS_VALID_GPIO (C macro), 335
rtc_gpio_is_valid_gpio (C++ function), 332
rtc_gpio_mode_t (C++ enum), 336
rtc_gpio_mode_t::RTC_GPIO_MODE_DISABLED (C++ enumerator), 337
rtc_gpio_mode_t::RTC_GPIO_MODE_INPUT_ONLY (C++ enumerator), 336
rtc_gpio_mode_t::RTC_GPIO_MODE_INPUT_OUTPUT (C++ enumerator), 337
rtc_gpio_mode_t::RTC_GPIO_MODE_INPUT_OUTPUT_OD (C++ enumerator), 337
rtc_gpio_mode_t::RTC_GPIO_MODE_OUTPUT_OD (C++ enumerator), 337
rtc_gpio_mode_t::RTC_GPIO_MODE_OUTPUT_ONLY (C++ enumerator), 336
rtc_gpio_pulldown_dis (C++ function), 334
rtc_gpio_pulldown_en (C++ function), 333
rtc_gpio_pullup_dis (C++ function), 334
rtc_gpio_pullup_en (C++ function), 333
rtc_gpio_set_direction (C++ function), 333
rtc_gpio_set_direction_in_sleep (C++ function), 333
rtc_gpio_set_drive_capability (C++ function), 334
rtc_gpio_set_level (C++ function), 333
rtc_gpio_wakeup_disable (C++ function), 335
rtc_gpio_wakeup_enable (C++ function), 335
rtc_io_number_get (C++ function), 332
RTCIO_OUTPUT_NORMAL (C macro), 1745
RTCIO_OUTPUT_OD (C macro), 1745

S

sdm_channel_disable (C++ function), 660
sdm_channel_enable (C++ function), 659
sdm_channel_handle_t (C++ type), 661
sdm_channel_set_duty (C++ function), 660
sdm_channel_set_pulse_density (C++ function), 660
sdm_clock_source_t (C++ type), 661
sdm_config_t (C++ struct), 661
sdm_config_t::clk_src (C++ member), 661
sdm_config_t::flags (C++ member), 661
sdm_config_t::gpio_num (C++ member), 661
sdm_config_t::invert_out (C++ member), 661
sdm_config_t::io_loop_back (C++ member), 661
sdm_config_t::sample_rate_hz (C++ member), 661
sdm_del_channel (C++ function), 659
sdm_new_channel (C++ function), 659
sdmmc_can_discard (C++ function), 1307
sdmmc_can_trim (C++ function), 1307
sdmmc_card_init (C++ function), 1305
sdmmc_card_print_info (C++ function), 1306
sdmmc_erase_sectors (C++ function), 1306
sdmmc_full_erase (C++ function), 1307
sdmmc_get_status (C++ function), 1306
sdmmc_host_deinit (C++ function), 648
sdmmc_host_do_transaction (C++ function), 647
sdmmc_host_get_dma_info (C++ function), 649
sdmmc_host_get_real_freq (C++ function), 648

sdmmc_host_get_slot_width (C++ function), 646
sdmmc_host_init (C++ function), 646
sdmmc_host_init_slot (C++ function), 646
sdmmc_host_io_int_enable (C++ function), 648
sdmmc_host_io_int_wait (C++ function), 648
sdmmc_host_set_bus_ddr_mode (C++ function), 647
sdmmc_host_set_bus_width (C++ function), 646
sdmmc_host_set_card_clk (C++ function), 647
sdmmc_host_set_cclk_always_on (C++ function), 647
sdmmc_host_set_input_delay (C++ function), 648
sdmmc_io_enable_int (C++ function), 1309
SDMMC_IO_FIXED_ADDR (C macro), 1311
sdmmc_io_get_cis_data (C++ function), 1310
sdmmc_io_print_cis_info (C++ function), 1310
sdmmc_io_read_blocks (C++ function), 1309
sdmmc_io_read_byte (C++ function), 1307
sdmmc_io_read_bytes (C++ function), 1308
sdmmc_io_wait_int (C++ function), 1309
sdmmc_io_write_blocks (C++ function), 1309
sdmmc_io_write_byte (C++ function), 1308
sdmmc_io_write_bytes (C++ function), 1308
sdmmc_mmc_can_sanitize (C++ function), 1307
sdmmc_mmc_sanitize (C++ function), 1307
sdmmc_read_sectors (C++ function), 1306
sdmmc_slot_config_t (C++ struct), 649
sdmmc_slot_config_t::cd (C++ member), 650
sdmmc_slot_config_t::clk (C++ member), 649
sdmmc_slot_config_t::cmd (C++ member), 649
sdmmc_slot_config_t::d0 (C++ member), 649
sdmmc_slot_config_t::d1 (C++ member), 649
sdmmc_slot_config_t::d2 (C++ member), 649
sdmmc_slot_config_t::d3 (C++ member), 649
sdmmc_slot_config_t::d4 (C++ member), 649
sdmmc_slot_config_t::d5 (C++ member), 649
sdmmc_slot_config_t::d6 (C++ member), 650
sdmmc_slot_config_t::d7 (C++ member), 650
sdmmc_slot_config_t::flags (C++ member), 650
sdmmc_slot_config_t::gpio_cd (C++ member), 650
sdmmc_slot_config_t::gpio_wp (C++ member), 650
sdmmc_slot_config_t::width (C++ member), 650
sdmmc_slot_config_t::wp (C++ member), 650
SDMMC_SLOT_FLAG_INTERNAL_PULLUP (C macro), 650
SDMMC_SLOT_FLAG_WP_ACTIVE_HIGH (C macro), 650
sdmmc_write_sectors (C++ function), 1306
SDSPI_DEFAULT_DMA (C macro), 655
SDSPI_DEFAULT_HOST (C macro), 655
sdspi_dev_handle_t (C++ type), 656
SDSPI_DEVICE_CONFIG_DEFAULT (C macro), 656
sdspi_device_config_t (C++ struct), 655
sdspi_device_config_t::gpio_cd (C++ member), 655
sdspi_device_config_t::gpio_cs (C++ member), 655
sdspi_device_config_t::gpio_int (C++ member), 655
sdspi_device_config_t::gpio_wp (C++ member), 655
sdspi_device_config_t::gpio_wp_polarity (C++ member), 655
sdspi_device_config_t::host_id (C++ member), 655
SDSPI_HOST_DEFAULT (C macro), 655
sdspi_host_deinit (C++ function), 654
sdspi_host_do_transaction (C++ function), 653
sdspi_host_get_dma_info (C++ function), 654
sdspi_host_get_real_freq (C++ function), 654
sdspi_host_init (C++ function), 653
sdspi_host_init_device (C++ function), 653
sdspi_host_io_int_enable (C++ function), 654
sdspi_host_io_int_wait (C++ function), 654
sdspi_host_remove_device (C++ function), 653
sdspi_host_set_card_clk (C++ function), 654
SDSPI_IO_ACTIVE_LOW (C macro), 656
SDSPI_SLOT_NO_CD (C macro), 655
SDSPI_SLOT_NO_CS (C macro), 655
SDSPI_SLOT_NO_INT (C macro), 655
SDSPI_SLOT_NO_WP (C macro), 655
SemaphoreHandle_t (C++ type), 1501
semBINARY_SEMAPHORE_QUEUE_LENGTH (C macro), 1486
semGIVE_BLOCK_TIME (C macro), 1487
semSEMAPHORE_QUEUE_ITEM_LENGTH (C macro), 1487
shared_stack_function (C++ type), 1360
shutdown_handler_t (C++ type), 1640
slave_cb_t (C++ type), 730
slave_transaction_cb_t (C++ type), 723
sntp_get_sync_interval (C++ function), 1722
sntp_get_sync_mode (C++ function), 1721
sntp_get_sync_status (C++ function), 1721
sntp_getoperatingmode (C++ function), 1723
sntp_getreachability (C++ function), 1723
sntp_getserver (C++ function), 1723
sntp_getservername (C++ function), 1723
sntp_init (C++ function), 1723
SNTP_OPMODE_POLL (C macro), 1724

sntp_restart (*C++ function*), 1722
sntp_servermode_dhcp (*C++ function*), 1723
sntp_set_sync_interval (*C++ function*), 1722
sntp_set_sync_mode (*C++ function*), 1721
sntp_set_sync_status (*C++ function*), 1722
sntp_set_time_sync_notification_cb (*C++ function*), 1722
sntp_setoperatingmode (*C++ function*), 1723
sntp_setservername (*C++ function*), 1723
sntp_sync_mode_t (*C++ enum*), 1724
sntp_sync_mode_t::SNTP_SYNC_MODE_IMMED (*C++ enumerator*), 1724
sntp_sync_mode_t::SNTP_SYNC_MODE_SMOOTH (*C++ enumerator*), 1724
sntp_sync_status_t (*C++ enum*), 1724
sntp_sync_status_t::SNTP_SYNC_STATUS_COMPLETED (*C++ enumerator*), 1724
sntp_sync_status_t::SNTP_SYNC_STATUS_IN_PROGRESS (*C++ enumerator*), 1724
sntp_sync_status_t::SNTP_SYNC_STATUS_RESET (*C++ enumerator*), 1724
sntp_sync_time (*C++ function*), 1721
sntp_sync_time_cb_t (*C++ type*), 1724
SOC_ADC_ATTEN_NUM (*C macro*), 1700
SOC_ADC_CALIBRATION_V1_SUPPORTED (*C macro*), 1701
SOC_ADC_CHANNEL_NUM (*C macro*), 1700
SOC_ADC_DIG_CTRL_SUPPORTED (*C macro*), 1700
SOC_ADC_DIG_SUPPORTED_UNIT (*C macro*), 1700
SOC_ADC_DIGI_CLKS (*C macro*), 289
SOC_ADC_DIGI_CONTROLLER_NUM (*C macro*), 1700
SOC_ADC_DIGI_DATA_BYTES_PER_CONV (*C macro*), 1701
SOC_ADC_DIGI_IIR_FILTER_NUM (*C macro*), 1701
SOC_ADC_DIGI_MAX_BITWIDTH (*C macro*), 1701
SOC_ADC_DIGI_MIN_BITWIDTH (*C macro*), 1701
SOC_ADC_DIGI_MONITOR_NUM (*C macro*), 1701
SOC_ADC_DIGI_RESULT_BYTES (*C macro*), 1701
SOC_ADC_DMA_SUPPORTED (*C macro*), 1700
SOC_ADC_MAX_CHANNEL_NUM (*C macro*), 1700
SOC_ADC_PATT_LEN_MAX (*C macro*), 1700
SOC_ADC_PERIPH_NUM (*C macro*), 1700
SOC_ADC_RTC_CLKS (*C macro*), 289
SOC_ADC_RTC_CTRL_SUPPORTED (*C macro*), 1700
SOC_ADC_RTC_MAX_BITWIDTH (*C macro*), 1701
SOC_ADC_RTC_MIN_BITWIDTH (*C macro*), 1701
SOC_ADC_SAMPLE_FREQ_THRES_HIGH (*C macro*), 1701
SOC_ADC_SAMPLE_FREQ_THRES_LOW (*C macro*), 1701
SOC_ADC_SHARED_POWER (*C macro*), 1701
SOC_ADC_SUPPORTED (*C macro*), 1697
SOC_AES_GDMA (*C macro*), 1700
SOC_AES_SUPPORT_AES_128 (*C macro*), 1700
SOC_AES_SUPPORT_AES_256 (*C macro*), 1700
SOC_AES_SUPPORT_DMA (*C macro*), 1700
SOC_AES_SUPPORT_GCM (*C macro*), 1700
SOC_AES_SUPPORTED (*C macro*), 1698
SOC_AHB_GDMA_SUPPORTED (*C macro*), 1697
SOC_AHB_GDMA_VERSION (*C macro*), 1702
SOC_ANA_CMPR_CAN_DISTINGUISH_EDGE (*C macro*), 1704
SOC_ANA_CMPR_CLKS (*C macro*), 289
SOC_ANA_CMPR_NUM (*C macro*), 1704
SOC_ANA_CMPR_SUPPORT_ETM (*C macro*), 1704
SOC_ANA_CMPR_SUPPORTED (*C macro*), 1697
SOC_APB_BACKUP_DMA (*C macro*), 1701
SOC_APM_SUPPORTED (*C macro*), 1699
SOC_ASSIST_DEBUG_SUPPORTED (*C macro*), 1699
SOC_ASYNC_MEMCPY_SUPPORTED (*C macro*), 1697
SOC_ASYNCHRONOUS_BUS_ERROR_MODE (*C macro*), 1718
SOC_AXI_DMA_EXT_MEM_ENC_ALIGNMENT (*C macro*), 1703
SOC_AXI_GDMA_SUPPORT_PSRAM (*C macro*), 1703
SOC_AXI_GDMA_SUPPORTED (*C macro*), 1697
SOC_BOD_SUPPORTED (*C macro*), 1699
SOC_BRANCH_PREDICTOR_SUPPORTED (*C macro*), 1702
SOC_BROWNOUT_RESET_SUPPORTED (*C macro*), 1701
SOC_CACHE_FREEZE_SUPPORTED (*C macro*), 1701
SOC_CACHE_INTERNAL_MEM_VIA_L1CACHE (*C macro*), 1701
SOC_CACHE_WRITEBACK_SUPPORTED (*C macro*), 1701
SOC_CAM_CLKS (*C macro*), 289
SOC_CLK_APLL_SUPPORTED (*C macro*), 1717
SOC_CLK_LP_FAST_SUPPORT_LP_PLL (*C macro*), 1717
SOC_CLK_MPLL_SUPPORTED (*C macro*), 1717
SOC_CLK_RC32K_FREQ_APPROX (*C macro*), 288
SOC_CLK_RC32K_SUPPORTED (*C macro*), 1717
SOC_CLK_RC_FAST_FREQ_APPROX (*C macro*), 288
SOC_CLK_RC_FAST_SUPPORT_CALIBRATION (*C macro*), 1717
SOC_CLK_RC_SLOW_FREQ_APPROX (*C macro*), 288
SOC_CLK_TREE_SUPPORTED (*C macro*), 1699
SOC_CLK_XTAL32K_FREQ_APPROX (*C macro*), 288
SOC_CLK_XTAL32K_SUPPORTED (*C macro*), 1717
soc_clkout_sig_id_t (*C++ enum*), 304
soc_clkout_sig_id_t::CLKOUT_SIG_APB (*C++ enumerator*), 305
soc_clkout_sig_id_t::CLKOUT_SIG_CPLL (*C++ enumerator*), 304
soc_clkout_sig_id_t::CLKOUT_SIG_CPU (*C++ enumerator*), 305
soc_clkout_sig_id_t::CLKOUT_SIG_I2S0 (*C++ enumerator*), 305
soc_clkout_sig_id_t::CLKOUT_SIG_I2S1 (*C++ enumerator*), 305

soc_clkout_sig_id_t::CLKOUT_SIG_I2S2 (C++ enumerator), 305
soc_clkout_sig_id_t::CLKOUT_SIG_INVALID (C++ enumerator), 305
soc_clkout_sig_id_t::CLKOUT_SIG_MEM (C++ enumerator), 305
soc_clkout_sig_id_t::CLKOUT_SIG_MPLL (C++ enumerator), 304
soc_clkout_sig_id_t::CLKOUT_SIG_PLL_F80M (C++ enumerator), 305
soc_clkout_sig_id_t::CLKOUT_SIG_RC_32K (C++ enumerator), 304
soc_clkout_sig_id_t::CLKOUT_SIG_RC_FAST (C++ enumerator), 304
soc_clkout_sig_id_t::CLKOUT_SIG_RC_SLOW (C++ enumerator), 304
soc_clkout_sig_id_t::CLKOUT_SIG_SPLL (C++ enumerator), 304
soc_clkout_sig_id_t::CLKOUT_SIG_SYS (C++ enumerator), 305
soc_clkout_sig_id_t::CLKOUT_SIG_XTAL (C++ enumerator), 304
soc_clkout_sig_id_t::CLKOUT_SIG_XTAL32K (C++ enumerator), 305
SOC_CLOCKOUT_SUPPORT_CHANNEL_DIVIDER (C macro), 1704
SOC_COEX_HW_PTI (C macro), 1716
SOC_CPU_BREAKPOINTS_NUM (C macro), 1702
soc_cpu_clk_src_t (C++ enum), 290
soc_cpu_clk_src_t::SOC_CPU_CLK_SRC_CPLL (C++ enumerator), 291
soc_cpu_clk_src_t::SOC_CPU_CLK_SRC_INVALID (C++ enumerator), 291
soc_cpu_clk_src_t::SOC_CPU_CLK_SRC_PLL (C++ enumerator), 291
soc_cpu_clk_src_t::SOC_CPU_CLK_SRC_RC_FAST (C++ enumerator), 291
soc_cpu_clk_src_t::SOC_CPU_CLK_SRC_XTAL (C++ enumerator), 290
SOC_CPU_COPROC_NUM (C macro), 1702
SOC_CPU_CORES_NUM (C macro), 1701
SOC_CPU_HAS_FLEXIBLE_INTC (C macro), 1701
SOC_CPU_HAS_FPU (C macro), 1702
SOC_CPU_HAS_FPU_EXT_ILL_BUG (C macro), 1702
SOC_CPU_HAS_HWLOOP (C macro), 1702
SOC_CPU_HAS_LOCKUP_RESET (C macro), 1702
SOC_CPU_HAS_PIE (C macro), 1702
SOC_CPU_HAS_PMA (C macro), 1702
SOC_CPU_IDRAM_SPLIT_USING_PMP (C macro), 1702
SOC_CPU_IN_TOP_DOMAIN (C macro), 1717
SOC_CPU_INTR_NUM (C macro), 1701
SOC_CPU_PMP_REGION_GRANULARITY (C macro), 1702
SOC_CPU_WATCHPOINT_MAX_REGION_SIZE (C macro), 1702
SOC_CPU_WATCHPOINTS_NUM (C macro), 1702
SOC_DCDC_SUPPORTED (C macro), 1699
SOC_DEDIC_GPIO_IN_CHANNELS_NUM (C macro), 1704
SOC_DEDIC_GPIO_OUT_CHANNELS_NUM (C macro), 1704
SOC_DEDIC_PERIPH_ALWAYS_ENABLE (C macro), 1704
SOC_DEDICATED_GPIO_SUPPORTED (C macro), 1697
SOC_DEEP_SLEEP_SUPPORTED (C macro), 1700
SOC_DIG_SIGN_SUPPORTED (C macro), 1698
SOC_DMA2D_GROUPS (C macro), 1703
SOC_DMA2D_RX_CHANNELS_PER_GROUP (C macro), 1703
SOC_DMA2D_SUPPORTED (C macro), 1697
SOC_DMA2D_TX_CHANNELS_PER_GROUP (C macro), 1703
SOC_DMA_CAN_ACCESS_FLASH (C macro), 1702
SOC_DS_KEY_CHECK_MAX_WAIT_US (C macro), 1702
SOC_DS_KEY_PARAM_MD_IV_LENGTH (C macro), 1702
SOC_DS_SIGNATURE_MAX_BIT_LEN (C macro), 1702
SOC_DW_GDMA_SUPPORTED (C macro), 1697
SOC_ECC_EXTENDED_MODES_SUPPORTED (C macro), 1699
SOC_ECC_SUPPORTED (C macro), 1698
SOC_ECDSA_SUPPORT_DETERMINISTIC_MODE (C macro), 1711
SOC_ECDSA_SUPPORT_EXPORT_PUBKEY (C macro), 1711
SOC_ECDSA_SUPPORTED (C macro), 1699
SOC_ECDSA_USES_MPI (C macro), 1711
SOC_EFUSE_DIS_DIRECT_BOOT (C macro), 1714
SOC_EFUSE_DIS_DOWNLOAD_MSPI (C macro), 1715
SOC_EFUSE_DIS_PAD_JTAG (C macro), 1714
SOC_EFUSE_DIS_USB_JTAG (C macro), 1714
SOC_EFUSE_ECDSA_KEY (C macro), 1715
SOC_EFUSE_KEY_PURPOSE_FIELD (C macro), 1698
SOC_EFUSE_REVOKE_BOOT_KEY_DIGESTS (C macro), 1715
SOC_EFUSE_SECURE_BOOT_KEY_DIGESTS (C macro), 1715
SOC_EFUSE_SOFT_DIS_JTAG (C macro), 1714
SOC_EFUSE_SUPPORTED (C macro), 1698
SOC_EMAC_IEEE_1588_SUPPORT (C macro), 1718
SOC_EMAC_MII_USE_GPIO_MATRIX (C macro), 1718
SOC_EMAC_SUPPORTED (C macro), 1697
SOC_EMAC_USE_MULTI_IO_MUX (C macro), 1718
SOC_ETM_CHANNELS_PER_GROUP (C macro), 1703
SOC_ETM_GROUPS (C macro), 1703
SOC_ETM_SUPPORTED (C macro), 1697
SOC_FLASH_CLKS (C macro), 289
SOC_FLASH_ENC_SUPPORTED (C macro), 1699

SOC_FLASH_ENCRYPTED_XTS_AES_BLOCK_MAX (*C macro*), 1715
SOC_FLASH_ENCRYPTION_XTS_AES (*C macro*), 1715
SOC_FLASH_ENCRYPTION_XTS_AES_128 (*C macro*), 1715
SOC_FLASH_ENCRYPTION_XTS_AES_256 (*C macro*), 1715
SOC_FLASH_ENCRYPTION_XTS_AES_OPTIONS (*C macro*), 1715
SOC_GDMA_NUM_GROUPS_MAX (*C macro*), 1703
SOC_GDMA_PAIRS_PER_GROUP_MAX (*C macro*), 1703
SOC_GDMA_SUPPORT_CRC (*C macro*), 1703
SOC_GDMA_SUPPORT_ETM (*C macro*), 1703
SOC_GDMA_SUPPORTED (*C macro*), 1697
SOC_GLITCH_FILTER_CLKS (*C macro*), 289
SOC_GP_LDO_SUPPORTED (*C macro*), 1700
SOC_GPIO_CLOCKOUT_BY_GPIO_MATRIX (*C macro*), 1704
SOC_GPIO_CLOCKOUT_CHANNEL_NUM (*C macro*), 1704
SOC_GPIO_DEEP_SLEEP_WAKE_SUPPORTED_PIN_CNT (*C macro*), 1704
SOC_GPIO_DEEP_SLEEP_WAKE_VALID_GPIO_MASK (*C macro*), 1704
SOC_GPIO_FLEX_GLITCH_FILTER_NUM (*C macro*), 1703
SOC_GPIO_IN_RANGE_MAX (*C macro*), 1703
SOC_GPIO_OUT_RANGE_MAX (*C macro*), 1704
SOC_GPIO_PIN_COUNT (*C macro*), 1703
SOC_GPIO_PORT (*C macro*), 1703
SOC_GPIO_SUPPORT_DEEPSLEEP_WAKEUP (*C macro*), 1703
SOC_GPIO_SUPPORT_ETM (*C macro*), 1703
SOC_GPIO_SUPPORT_FORCE_HOLD (*C macro*), 1704
SOC_GPIO_SUPPORT_PIN_GLITCH_FILTER (*C macro*), 1703
SOC_GPIO_SUPPORT_PIN_HYS_FILTER (*C macro*), 1703
SOC_GPIO_SUPPORT_RTC_INDEPENDENT (*C macro*), 1703
SOC_GPIO_VALID_DIGITAL_IO_PAD_MASK (*C macro*), 1704
SOC_GPIO_VALID_GPIO_MASK (*C macro*), 1703
SOC_GPIO_VALID_OUTPUT_GPIO_MASK (*C macro*), 1703
SOC_GPSPI_SUPPORTED (*C macro*), 1698
SOC_GPTIMER_CLKS (*C macro*), 288
SOC_GPTIMER_SUPPORTED (*C macro*), 1697
SOC_HMAC_SUPPORTED (*C macro*), 1698
SOC_HP_CPU_HAS_MULTIPLE_CORES (*C macro*), 1702
SOC_HP_I2C_NUM (*C macro*), 1704
SOC_I2C_CLKS (*C macro*), 289
SOC_I2C_CMD_REG_NUM (*C macro*), 1705
SOC_I2C_FIFO_LEN (*C macro*), 1704
SOC_I2C_NUM (*C macro*), 1704
SOC_I2C_SLAVE_SUPPORT_BROADCAST (*C macro*), 1705
SOC_I2C_SLAVE_SUPPORT_I2CRAM_ACCESS (*C macro*), 1705
SOC_I2C_SLAVE_SUPPORT_SLAVE_UNMATCH (*C macro*), 1705
SOC_I2C_SUPPORT_10BIT_ADDR (*C macro*), 1705
SOC_I2C_SUPPORT_HW_CLR_BUS (*C macro*), 1705
SOC_I2C_SUPPORT_HW_FSM_RST (*C macro*), 1705
SOC_I2C_SUPPORT_RTC (*C macro*), 1705
SOC_I2C_SUPPORT_SLAVE (*C macro*), 1705
SOC_I2C_SUPPORT_XTAL (*C macro*), 1705
SOC_I2C_SUPPORTED (*C macro*), 1698
SOC_I2S_CLKS (*C macro*), 288
SOC_I2S_HW_VERSION_2 (*C macro*), 1705
SOC_I2S_NUM (*C macro*), 1705
SOC_I2S_PDM_MAX_RX_LINES (*C macro*), 1706
SOC_I2S_PDM_MAX_TX_LINES (*C macro*), 1706
SOC_I2S_SUPPORTED (*C macro*), 1698
SOC_I2S_SUPPORTS_APLL (*C macro*), 1705
SOC_I2S_SUPPORTS_PCM (*C macro*), 1705
SOC_I2S_SUPPORTS_PDM (*C macro*), 1705
SOC_I2S_SUPPORTS_PDM_RX (*C macro*), 1705
SOC_I2S_SUPPORTS_PDM_RX_HP_FILTER (*C macro*), 1705
SOC_I2S_SUPPORTS_PDM_TX (*C macro*), 1705
SOC_I2S_SUPPORTS_TDM (*C macro*), 1706
SOC_I2S_SUPPORTS_TX_SYNC_CNT (*C macro*), 1705
SOC_I2S_SUPPORTS_XTAL (*C macro*), 1705
SOC_I2S_TDM_FULL_DATA_WIDTH (*C macro*), 1706
SOC_INT_CLIC_SUPPORTED (*C macro*), 1702
SOC_INT_HW_NESTED_SUPPORTED (*C macro*), 1702
SOC_ISP_AF_CTLR_NUMS (*C macro*), 1706
SOC_ISP_AF_WINDOW_NUMS (*C macro*), 1706
SOC_ISP_BF_SUPPORTED (*C macro*), 1706
SOC_ISP_BF_TEMPLATE_X_NUMS (*C macro*), 1706
SOC_ISP_BF_TEMPLATE_Y_NUMS (*C macro*), 1706
SOC_ISP_CCM_DIMENSION (*C macro*), 1706
SOC_ISP_CCM_SUPPORTED (*C macro*), 1706
SOC_ISP_CLKS (*C macro*), 289
SOC_ISP_DVP_CTLR_NUMS (*C macro*), 1706
SOC_ISP_DVP_DATA_WIDTH_MAX (*C macro*), 1706
SOC_ISP_DVP_SUPPORTED (*C macro*), 1706
SOC_ISP_NUMS (*C macro*), 1706
SOC_ISP_SHARE_CSI_BRG (*C macro*), 1706
SOC_ISP_SUPPORTED (*C macro*), 1698
SOC_JPEG_CODEC_SUPPORTED (*C macro*), 1718
SOC_JPEG_DECODE_SUPPORTED (*C macro*), 1718
SOC_JPEG_ENCODE_SUPPORTED (*C macro*), 1718
SOC_KEY_MANAGER_ECDSA_KEY_DEPLOY (*C macro*), 1715
SOC_KEY_MANAGER_FE_KEY_DEPLOY (*C macro*), 1715
SOC_KEY_MANAGER_SUPPORTED (*C macro*), 1699

SOC_LCD_CLKS (*C macro*), 288
SOC_LCD_I80_SUPPORTED (*C macro*), 1708
SOC_LCDCAM_CAM_DATA_WIDTH_MAX (*C macro*), 1718
SOC_LCDCAM_CAM_PERIPH_NUM (*C macro*), 1718
SOC_LCDCAM_CAM_SUPPORT_RGB_YUV_CONV (*C macro*), 1718
SOC_LCDCAM_CAM_SUPPORTED (*C macro*), 1697
SOC_LCDCAM_I80_BUS_WIDTH (*C macro*), 1708
SOC_LCDCAM_I80_LCD_SUPPORTED (*C macro*), 1697
SOC_LCDCAM_I80_NUM_BUSES (*C macro*), 1708
SOC_LCDCAM_RGB_NUM_PANELS (*C macro*), 1708
SOC_LCDCAM_SUPPORTED (*C macro*), 1697
SOC_LEDC_CHANNEL_NUM (*C macro*), 1706
SOC_LEDC_CLKS (*C macro*), 290
SOC_LEDC_FADE_PARAMS_BIT_WIDTH (*C macro*), 1707
SOC_LEDC_GAMMA_CURVE_FADE_RANGE_MAX (*C macro*), 1706
SOC_LEDC_GAMMA_CURVE_FADE_SUPPORTED (*C macro*), 1706
SOC_LEDC_SUPPORT_FADE_STOP (*C macro*), 1707
SOC_LEDC_SUPPORT_PLL_DIV_CLOCK (*C macro*), 1706
SOC_LEDC_SUPPORT_XTAL_CLOCK (*C macro*), 1706
SOC_LEDC_SUPPORTED (*C macro*), 1698
SOC_LEDC_TIMER_BIT_WIDTH (*C macro*), 1706
SOC_LIGHT_SLEEP_SUPPORTED (*C macro*), 1700
SOC_LP_CORE_SUPPORT_ETM (*C macro*), 1718
SOC_LP_CORE_SUPPORTED (*C macro*), 1698
SOC_LP_GPIO_MATRIX_SUPPORTED (*C macro*), 1699
SOC_LP_I2C_CLKS (*C macro*), 289
SOC_LP_I2C_FIFO_LEN (*C macro*), 1705
SOC_LP_I2C_NUM (*C macro*), 1705
SOC_LP_I2C_SUPPORTED (*C macro*), 1699
SOC_LP_I2S_CLKS (*C macro*), 288
SOC_LP_IO_HAS_INDEPENDENT_WAKEUP_SOURCE (*C macro*), 1703
SOC_LP_PERIPHERALS_SUPPORTED (*C macro*), 1699
soc_lp_pll_clk_src_t (*C++ enum*), 292
soc_lp_pll_clk_src_t::SOC_LP_PLL_CLK_SRC_INVALID (*C++ enumerator*), 292
soc_lp_pll_clk_src_t::SOC_LP_PLL_CLK_SRC_RC32K (*C++ enumerator*), 292
soc_lp_pll_clk_src_t::SOC_LP_PLL_CLK_SRC_XTAL32K (*C++ enumerator*), 292
SOC_LP_SPI_MAXIMUM_BUFFER_SIZE (*C macro*), 1712
SOC_LP_SPI_PERIPH_NUM (*C macro*), 1712
SOC_LP_SPI_SUPPORTED (*C macro*), 1699
SOC_LP_TIMER_BIT_WIDTH_HI (*C macro*), 1713
SOC_LP_TIMER_BIT_WIDTH_LO (*C macro*), 1713
SOC_LP_TIMER_SUPPORTED (*C macro*), 1699
SOC_LP_UART_CLKS (*C macro*), 288
SOC_LP_UART_FIFO_LEN (*C macro*), 1715
SOC_MCPWM_CAPTURE_CHANNELS_PER_TIMER (*C macro*), 1709
SOC_MCPWM_CAPTURE_CLK_FROM_GROUP (*C macro*), 1709
SOC_MCPWM_CAPTURE_CLKS (*C macro*), 288
SOC_MCPWM_CAPTURE_TIMERS_PER_GROUP (*C macro*), 1709
SOC_MCPWM_CARRIER_CLKS (*C macro*), 288
SOC_MCPWM_COMPARATORS_PER_OPERATOR (*C macro*), 1709
SOC_MCPWM_EVENT_COMPARATORS_PER_OPERATOR (*C macro*), 1709
SOC_MCPWM_GENERATORS_PER_OPERATOR (*C macro*), 1709
SOC_MCPWM_GPIO_FAULTS_PER_GROUP (*C macro*), 1709
SOC_MCPWM_GPIO_SYNCHROS_PER_GROUP (*C macro*), 1709
SOC_MCPWM_GROUPS (*C macro*), 1708
SOC_MCPWM_OPERATORS_PER_GROUP (*C macro*), 1709
SOC_MCPWM_SUPPORT_ETM (*C macro*), 1709
SOC_MCPWM_SUPPORT_EVENT_COMPARATOR (*C macro*), 1709
SOC_MCPWM_SUPPORTED (*C macro*), 1697
SOC_MCPWM_SWSYNC_CAN_PROPAGATE (*C macro*), 1709
SOC_MCPWM_TIMER_CLKS (*C macro*), 288
SOC_MCPWM_TIMERS_PER_GROUP (*C macro*), 1709
SOC_MCPWM_TRIGGERS_PER_OPERATOR (*C macro*), 1709
SOC_MEM_NON_CONTIGUOUS_SRAM (*C macro*), 1718
SOC_MEM_TCM_SUPPORTED (*C macro*), 1717
SOC_MEMSPI_IS_INDEPENDENT (*C macro*), 1712
SOC_MEMSPI_SRC_FREQ_20M_SUPPORTED (*C macro*), 1713
SOC_MEMSPI_SRC_FREQ_40M_SUPPORTED (*C macro*), 1713
SOC_MEMSPI_SRC_FREQ_80M_SUPPORTED (*C macro*), 1712
SOC_MEMSPI_TIMING_TUNING_BY_DQS (*C macro*), 1712
SOC_MIPI_CSI_PHY_CLKS (*C macro*), 289
SOC_MIPI_CSI_SUPPORTED (*C macro*), 1697
SOC_MIPI_DSI_DPI_CLKS (*C macro*), 289
SOC_MIPI_DSI_PHY_CLKS (*C macro*), 289
SOC_MIPI_DSI_SUPPORTED (*C macro*), 1697
SOC_MMU_DI_VADDR_SHARED (*C macro*), 1707
SOC_MMU_LINEAR_ADDRESS_REGION_NUM (*C macro*), 1707
SOC_MMU_PAGE_SIZE_CONFIGURABLE (*C macro*), 1707
SOC_MMU_PER_EXT_MEM_TARGET (*C macro*), 1707
SOC_MMU_PERIPH_NUM (*C macro*), 1707
SOC_MODEM_CLOCK_IS_INDEPENDENT (*C macro*), 1717

soc_module_clk_t (C++ enum), 292
soc_module_clk_t::SOC_MOD_CLK_APLL (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_CPLL (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_CPU (C++ enumerator), 292
soc_module_clk_t::SOC_MOD_CLK_INVALID (C++ enumerator), 294
soc_module_clk_t::SOC_MOD_CLK_LP_DYN_FAST (C++ enumerator), 294
soc_module_clk_t::SOC_MOD_CLK_LP_PERI (C++ enumerator), 294
soc_module_clk_t::SOC_MOD_CLK_LP_PLL (C++ enumerator), 294
soc_module_clk_t::SOC_MOD_CLK_MPLL (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_PLL_F160M (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_PLL_F20M (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_PLL_F240M (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_PLL_F25M (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_PLL_F80M (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_RC_FAST (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_RTC_FAST (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_RTC_SLOW (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_SPLL (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_XTAL (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_XTAL32K (C++ enumerator), 293
soc_module_clk_t::SOC_MOD_CLK_XTAL_D2 (C++ enumerator), 294
SOC_MPI_MEM_BLOCKS_NUM (C macro), 1710
SOC_MPI_OPERATIONS_NUM (C macro), 1710
SOC_MPI_SUPPORTED (C macro), 1698
SOC_MPU_CONFIGURABLE_REGIONS_SUPPORTED (C macro), 1707
SOC_MPU_MIN_REGION_SIZE (C macro), 1707
SOC_MPU_REGION_RO_SUPPORTED (C macro), 1707
SOC_MPU_REGION_WO_SUPPORTED (C macro), 1707
SOC_MPU_REGIONS_MAX_NUM (C macro), 1707
SOC_MWDT_CLKS (C macro), 290
SOC_MWDT_SUPPORT_SLEEP_RETENTION (C macro), 1714
SOC_MWDT_SUPPORT_XTAL (C macro), 1713
SOC_PARLIO_CLKS (C macro), 290
SOC_PARLIO_GROUPS (C macro), 1709
SOC_PARLIO_RX_CLK_SUPPORT_GATING (C macro), 1710
SOC_PARLIO_RX_CLK_SUPPORT_OUTPUT (C macro), 1710
SOC_PARLIO_RX_UNIT_MAX_DATA_WIDTH (C macro), 1710
SOC_PARLIO_RX_UNITS_PER_GROUP (C macro), 1710
SOC_PARLIO_SUPPORTED (C macro), 1697
SOC_PARLIO_TRANS_BIT_ALIGN (C macro), 1710
SOC_PARLIO_TX_CLK_SUPPORT_GATING (C macro), 1710
SOC_PARLIO_TX_SIZE_BY_DMA (C macro), 1710
SOC_PARLIO_TX_UNIT_MAX_DATA_WIDTH (C macro), 1710
SOC_PARLIO_TX_UNITS_PER_GROUP (C macro), 1709
SOC_PAU_IN_TOP_DOMAIN (C macro), 1717
SOC_PAU_SUPPORTED (C macro), 1699
SOC_PCNT_CHANNELS_PER_UNIT (C macro), 1707
SOC_PCNT_GROUPS (C macro), 1707
SOC_PCNT_SUPPORT_CLEAR_SIGNAL (C macro), 1707
SOC_PCNT_SUPPORT_RUNTIME_THRES_UPDATE (C macro), 1707
SOC_PCNT_SUPPORTED (C macro), 1697
SOC_PCNT_THRES_POINT_PER_UNIT (C macro), 1707
SOC_PCNT_UNITS_PER_GROUP (C macro), 1707
soc_periph_adc_digi_clk_src_t (C++ enum), 302
soc_periph_adc_digi_clk_src_t::ADC_DIGI_CLK_SRC_D (C++ enumerator), 302
soc_periph_adc_digi_clk_src_t::ADC_DIGI_CLK_SRC_P (C++ enumerator), 302
soc_periph_adc_digi_clk_src_t::ADC_DIGI_CLK_SRC_R (C++ enumerator), 302
soc_periph_adc_digi_clk_src_t::ADC_DIGI_CLK_SRC_X (C++ enumerator), 302
soc_periph_adc_rtc_clk_src_t (C++ enum), 302
soc_periph_adc_rtc_clk_src_t::ADC_RTC_CLK_SRC_DEF (C++ enumerator), 302
soc_periph_adc_rtc_clk_src_t::ADC_RTC_CLK_SRC_RC_ (C++ enumerator), 302
soc_periph_ana_cmpr_clk_src_t (C++ enum), 301
soc_periph_ana_cmpr_clk_src_t::ANA_CMPR_CLK_SRC_D (C++ enumerator), 302
soc_periph_ana_cmpr_clk_src_t::ANA_CMPR_CLK_SRC_P (C++ enumerator), 302
soc_periph_ana_cmpr_clk_src_t::ANA_CMPR_CLK_SRC_X (C++ enumerator), 302
soc_periph_cam_clk_src_t (C++ enum), 298
soc_periph_cam_clk_src_t::CAM_CLK_SRC_APLL (C++ enumerator), 298
soc_periph_cam_clk_src_t::CAM_CLK_SRC_DEFAULT (C++ enumerator), 298

soc_periph_cam_clk_src_t::CAM_CLK_SRC_PLL160M (C++ enumerator), 298
soc_periph_cam_clk_src_t::CAM_CLK_SRC_XTAL (C++ enumerator), 298
SOC_PERIPH_CLK_CTRL_SHARED (C macro), 1717
soc_periph_flash_clk_src_t (C++ enum), 300
soc_periph_flash_clk_src_t::FLASH_CLK_SRC_APLL (C++ enumerator), 300
soc_periph_flash_clk_src_t::FLASH_CLK_SRC_DEFAULT (C++ enumerator), 300
soc_periph_flash_clk_src_t::FLASH_CLK_SRC_SPLL (C++ enumerator), 301
soc_periph_flash_clk_src_t::FLASH_CLK_SRC_XTAL (C++ enumerator), 300
soc_periph_glitch_filter_clk_src_t (C++ enum), 301
soc_periph_glitch_filter_clk_src_t::GLITCH_FILTER_CLK_SRC_APLL (C++ enumerator), 301
soc_periph_glitch_filter_clk_src_t::GLITCH_FILTER_CLK_SRC_PLL_F80M (C++ enumerator), 301
soc_periph_glitch_filter_clk_src_t::GLITCH_FILTER_CLK_SRC_XTAL (C++ enumerator), 301
soc_periph_gptimer_clk_src_t (C++ enum), 294
soc_periph_gptimer_clk_src_t::GPTIMER_CLK_SRC_APLL (C++ enumerator), 294
soc_periph_gptimer_clk_src_t::GPTIMER_CLK_SRC_DEFAULT (C++ enumerator), 294
soc_periph_gptimer_clk_src_t::GPTIMER_CLK_SRC_PLL_F80M (C++ enumerator), 294
soc_periph_gptimer_clk_src_t::GPTIMER_CLK_SRC_RC_FAST (C++ enumerator), 294
soc_periph_gptimer_clk_src_t::GPTIMER_CLK_SRC_XTAL (C++ enumerator), 294
soc_periph_i2c_clk_src_t (C++ enum), 299
soc_periph_i2c_clk_src_t::I2C_CLK_SRC_DEFAULT (C++ enumerator), 299
soc_periph_i2c_clk_src_t::I2C_CLK_SRC_RC_FAST (C++ enumerator), 299
soc_periph_i2c_clk_src_t::I2C_CLK_SRC_XTAL (C++ enumerator), 299
soc_periph_i2s_clk_src_t (C++ enum), 297
soc_periph_i2s_clk_src_t::I2S_CLK_SRC_APLL (C++ enumerator), 297
soc_periph_i2s_clk_src_t::I2S_CLK_SRC_DEFAULT (C++ enumerator), 297
soc_periph_i2s_clk_src_t::I2S_CLK_SRC_EXTERNAL (C++ enumerator), 297
soc_periph_i2s_clk_src_t::I2S_CLK_SRC_XTAL (C++ enumerator), 297
soc_periph_isp_clk_src_t (C++ enum), 301
soc_periph_isp_clk_src_t::ISP_CLK_SRC_DEFAULT (C++ enumerator), 301
soc_periph_isp_clk_src_t::ISP_CLK_SRC_PLL160 (C++ enumerator), 301
soc_periph_isp_clk_src_t::ISP_CLK_SRC_PLL240 (C++ enumerator), 301
soc_periph_isp_clk_src_t::ISP_CLK_SRC_XTAL (C++ enumerator), 301
soc_periph_lcd_clk_src_t (C++ enum), 297
soc_periph_lcd_clk_src_t::LCD_CLK_SRC_APLL (C++ enumerator), 298
soc_periph_lcd_clk_src_t::LCD_CLK_SRC_DEFAULT (C++ enumerator), 298
soc_periph_lcd_clk_src_t::LCD_CLK_SRC_PLL160M (C++ enumerator), 297
soc_periph_lcd_clk_src_t::LCD_CLK_SRC_XTAL (C++ enumerator), 297
soc_periph_ledc_clk_src_legacy_t (C++ enum), 303
soc_periph_ledc_clk_src_legacy_t::LEDC_AUTO_CLK (C++ enumerator), 303
soc_periph_ledc_clk_src_legacy_t::LEDC_USE_PLL_DI (C++ enumerator), 303
soc_periph_ledc_clk_src_legacy_t::LEDC_USE_RC_FAS (C++ enumerator), 303
soc_periph_ledc_clk_src_legacy_t::LEDC_USE_XTAL_C (C++ enumerator), 303
soc_periph_lp_i2c_clk_src_t (C++ enum), 299
soc_periph_lp_i2c_clk_src_t::LP_I2C_SCLK_DEFAULT (C++ enumerator), 299
soc_periph_lp_i2c_clk_src_t::LP_I2C_SCLK_LP_FAST (C++ enumerator), 299
soc_periph_lp_i2c_clk_src_t::LP_I2C_SCLK_XTAL_D2 (C++ enumerator), 299
soc_periph_lp_i2s_clk_src_t (C++ enum), 297
soc_periph_lp_i2s_clk_src_t::LP_I2S_CLK_SRC_DEFAU (C++ enumerator), 297
soc_periph_lp_i2s_clk_src_t::LP_I2S_CLK_SRC_LP_PE (C++ enumerator), 297
soc_periph_lp_i2s_clk_src_t::LP_I2S_CLK_SRC_XTAL_ (C++ enumerator), 297
soc_periph_lp_uart_clk_src_t (C++ enum), 296
soc_periph_lp_uart_clk_src_t::LP_UART_SCLK_DEFAUL (C++ enumerator), 296
soc_periph_lp_uart_clk_src_t::LP_UART_SCLK_LP_FAS (C++ enumerator), 296
soc_periph_lp_uart_clk_src_t::LP_UART_SCLK_XTAL_D (C++ enumerator), 296
soc_periph_mcpwm_capture_clk_src_t (C++ enum), 296
soc_periph_mcpwm_capture_clk_src_t::MCPWM_CAPTURE (C++ enumerator), 296
soc_periph_mcpwm_capture_clk_src_t::MCPWM_CAPTURE (C++ enumerator), 296
soc_periph_mcpwm_capture_clk_src_t::MCPWM_CAPTURE (C++ enumerator), 296
soc_periph_mcpwm_carrier_clk_src_t (C++ enum), 296
soc_periph_mcpwm_carrier_clk_src_t::MCPWM_CARRIER (C++ enumerator), 297
soc_periph_mcpwm_carrier_clk_src_t::MCPWM_CARRIER (C++ enumerator), 297
soc_periph_mcpwm_carrier_clk_src_t::MCPWM_CARRIER

(C++ enumerator), 297
soc_periph_mcpwm_timer_clk_src_t (C++ enum), 296
soc_periph_mcpwm_timer_clk_src_t::MCPWM_TIMER_CLK_SRC_DEFAULT (C++ enumerator), 296
soc_periph_mcpwm_timer_clk_src_t::MCPWM_TIMER_CLK_SRC_PLL160M (C++ enumerator), 296
soc_periph_mcpwm_timer_clk_src_t::MCPWM_TIMER_CLK_SRC_XTAL (C++ enumerator), 296
soc_periph_mipi_csi_phy_clk_src_t (C++ enum), 298
soc_periph_mipi_csi_phy_clk_src_t::MIPI_CSI_PHY_CLK_SRC_DEFAULT (C++ enumerator), 298
soc_periph_mipi_csi_phy_clk_src_t::MIPI_CSI_PHY_CLK_SRC_PLL_F20M (C++ enumerator), 298
soc_periph_mipi_csi_phy_clk_src_t::MIPI_CSI_PHY_CLK_SRC_PLL_F25M (C++ enumerator), 298
soc_periph_mipi_csi_phy_clk_src_t::MIPI_CSI_PHY_CLK_SRC_XTAL (C++ enumerator), 298
soc_periph_mipi_dsi_dpi_clk_src_t (C++ enum), 298
soc_periph_mipi_dsi_dpi_clk_src_t::MIPI_DSI_DPI_CLK_SRC_DEFAULT (C++ enumerator), 299
soc_periph_mipi_dsi_dpi_clk_src_t::MIPI_DSI_DPI_CLK_SRC_PLL_F160M (C++ enumerator), 298
soc_periph_mipi_dsi_dpi_clk_src_t::MIPI_DSI_DPI_CLK_SRC_PLL_F240M (C++ enumerator), 299
soc_periph_mipi_dsi_dpi_clk_src_t::MIPI_DSI_DPI_CLK_SRC_XTAL (C++ enumerator), 298
soc_periph_mipi_dsi_phy_clk_src_t (C++ enum), 299
soc_periph_mipi_dsi_phy_clk_src_t::MIPI_DSI_PHY_CLK_SRC_DEFAULT (C++ enumerator), 299
soc_periph_mipi_dsi_phy_clk_src_t::MIPI_DSI_PHY_CLK_SRC_PLL_F20M (C++ enumerator), 299
soc_periph_mipi_dsi_phy_clk_src_t::MIPI_DSI_PHY_CLK_SRC_PLL_F25M (C++ enumerator), 299
soc_periph_mipi_dsi_phy_clk_src_t::MIPI_DSI_PHY_CLK_SRC_RC_FAST (C++ enumerator), 299
soc_periph_mwdt_clk_src_t (C++ enum), 303
soc_periph_mwdt_clk_src_t::MWDT_CLK_SRC_DEFAULT (C++ enumerator), 303
soc_periph_mwdt_clk_src_t::MWDT_CLK_SRC_PLL_F80M (C++ enumerator), 303
soc_periph_mwdt_clk_src_t::MWDT_CLK_SRC_RC_FAST (C++ enumerator), 303
soc_periph_mwdt_clk_src_t::MWDT_CLK_SRC_XTAL (C++ enumerator), 303
soc_periph_parlio_clk_src_t (C++ enum), 303
soc_periph_parlio_clk_src_t::PARLIO_CLK_SRC_DEFAULT (C++ enumerator), 304
soc_periph_parlio_clk_src_t::PARLIO_CLK_SRC_EXTERNAL (C++ enumerator), 303
soc_periph_parlio_clk_src_t::PARLIO_CLK_SRC_PLL160M (C++ enumerator), 303
soc_periph_parlio_clk_src_t::PARLIO_CLK_SRC_RC_FAST (C++ enumerator), 303
soc_periph_parlio_clk_src_t::PARLIO_CLK_SRC_XTAL (C++ enumerator), 303
soc_periph_psram_clk_src_t (C++ enum), 300
soc_periph_psram_clk_src_t::PSRAM_CLK_SRC_CPLL (C++ enumerator), 300
soc_periph_psram_clk_src_t::PSRAM_CLK_SRC_DEFAULT (C++ enumerator), 300
soc_periph_psram_clk_src_t::PSRAM_CLK_SRC_MPLL (C++ enumerator), 300
soc_periph_psram_clk_src_t::PSRAM_CLK_SRC_SPLL (C++ enumerator), 300
soc_periph_psram_clk_src_t::PSRAM_CLK_SRC_XTAL (C++ enumerator), 300
soc_periph_rmt_clk_src_legacy_t (C++ enum), 295
soc_periph_rmt_clk_src_legacy_t::RMT_BASECLK_DEFAULT (C++ enumerator), 295
soc_periph_rmt_clk_src_legacy_t::RMT_BASECLK_PLL_F80M (C++ enumerator), 295
soc_periph_rmt_clk_src_legacy_t::RMT_BASECLK_XTAL (C++ enumerator), 295
soc_periph_rmt_clk_src_t (C++ enum), 295
soc_periph_rmt_clk_src_t::RMT_CLK_SRC_DEFAULT (C++ enumerator), 295
soc_periph_rmt_clk_src_t::RMT_CLK_SRC_PLL_F80M (C++ enumerator), 295
soc_periph_rmt_clk_src_t::RMT_CLK_SRC_RC_FAST (C++ enumerator), 295
soc_periph_rmt_clk_src_t::RMT_CLK_SRC_XTAL (C++ enumerator), 295
soc_periph_sdm_clk_src_t (C++ enum), 301
soc_periph_sdm_clk_src_t::SDM_CLK_SRC_DEFAULT (C++ enumerator), 301
soc_periph_sdm_clk_src_t::SDM_CLK_SRC_PLL_F80M (C++ enumerator), 301
soc_periph_sdm_clk_src_t::SDM_CLK_SRC_XTAL (C++ enumerator), 301
soc_periph_sdmmc_clk_src_t (C++ enum), 304
soc_periph_sdmmc_clk_src_t::SDMMC_CLK_SRC_DEFAULT (C++ enumerator), 304
soc_periph_sdmmc_clk_src_t::SDMMC_CLK_SRC_PLL160M (C++ enumerator), 304
soc_periph_spi_clk_src_t (C++ enum), 300
soc_periph_spi_clk_src_t::SPI_CLK_SRC_DEFAULT (C++ enumerator), 300
soc_periph_spi_clk_src_t::SPI_CLK_SRC_RC_FAST (C++ enumerator), 300
soc_periph_spi_clk_src_t::SPI_CLK_SRC_SPLL (C++ enumerator), 300
soc_periph_spi_clk_src_t::SPI_CLK_SRC_XTAL (C++ enumerator), 300
soc_periph_systimer_clk_src_t (C++ enum), 294
soc_periph_systimer_clk_src_t::SYSTIMER_CLK_SRC_D (C++ enumerator), 294
soc_periph_systimer_clk_src_t::SYSTIMER_CLK_SRC_R

(C++ enumerator), 294
soc_periph_systimer_clk_src_t::SYSTIMER_CLK_SRC_XTAL (C++ enumerator), 294
soc_periph_temperature_sensor_clk_src_t (C++ enum), 304
soc_periph_temperature_sensor_clk_src_t::TEMPERATURE_SENSOR_CLK_SRC_DEFAULT (C++ enumerator), 304
soc_periph_temperature_sensor_clk_src_t::TEMPERATURE_SENSOR_CLK_SRC_XTAL (C++ enumerator), 304
soc_periph_tg_clk_src_legacy_t (C++ enum), 294
soc_periph_tg_clk_src_legacy_t::TIMER_SRC_CLK_DEFAULT (C++ enumerator), 295
soc_periph_tg_clk_src_legacy_t::TIMER_SRC_CLK_F80M (C++ enumerator), 295
soc_periph_tg_clk_src_legacy_t::TIMER_SRC_CLK_XTAL (C++ enumerator), 295
soc_periph_twai_clk_src_t (C++ enum), 302
soc_periph_twai_clk_src_t::TWAI_CLK_SRC_DEFAULT (C++ enumerator), 302
soc_periph_twai_clk_src_t::TWAI_CLK_SRC_RC_FAST (C++ enumerator), 302
soc_periph_twai_clk_src_t::TWAI_CLK_SRC_XTAL (C++ enumerator), 302
soc_periph_uart_clk_src_legacy_t (C++ enum), 295
soc_periph_uart_clk_src_legacy_t::UART_SCLK_DEFAULT (C++ enumerator), 296
soc_periph_uart_clk_src_legacy_t::UART_SCLK_FOSC_DIV (C++ enumerator), 295
soc_periph_uart_clk_src_legacy_t::UART_SCLK_RTC (C++ enumerator), 295
soc_periph_uart_clk_src_legacy_t::UART_SCLK_XTAL (C++ enumerator), 296
SOC_PHY_DIG_REGS_MEM_SIZE (C macro), 1716
SOC_PM_CPU_RETENTION_BY_SW (C macro), 1717
SOC_PM_EXT1_WAKEUP_BY_PMU (C macro), 1716
SOC_PM_PAU_LINK_NUM (C macro), 1717
SOC_PM_SUPPORT_CNNT_PD (C macro), 1716
SOC_PM_SUPPORT_DEEPSLEEP_CHECK_STUB_ONLY (C macro), 1717
SOC_PM_SUPPORT_EXT1_WAKEUP (C macro), 1716
SOC_PM_SUPPORT_EXT1_WAKEUP_MODE_PER_PIN (C macro), 1716
SOC_PM_SUPPORT_RC32K_PD (C macro), 1716
SOC_PM_SUPPORT_RC_FAST_PD (C macro), 1716
SOC_PM_SUPPORT_RTC_PERIPH_PD (C macro), 1717
SOC_PM_SUPPORT_TOP_PD (C macro), 1716
SOC_PM_SUPPORT_TOUCH_SENSOR_WAKEUP (C macro), 1716
SOC_PM_SUPPORT_VDDSDIO_PD (C macro), 1716
SOC_PM_SUPPORT_WIFI_WAKEUP (C macro), 1716
SOC_PM_SUPPORT_XTAL32K_PD (C macro), 1716
SOC_PM_SUPPORTED (C macro), 1700
SOC_PMU_SUPPORTED (C macro), 1699
SOC_PPA_SUPPORTED (C macro), 1700
SOC_PSRAM_CLKS (C macro), 289
SOC_PSRAM_DMA_CAPABLE (C macro), 1699
SOC_PSRAM_XTAL_POWER_MPLL (C macro), 1717
SOC_RMT_CHANNELS_PER_GROUP (C macro), 1708
SOC_RMT_CLKS (C macro), 288
SOC_RMT_GROUPS (C macro), 1707
SOC_RMT_MEM_WORDS_PER_CHANNEL (C macro), 1708
SOC_RMT_RX_CANDIDATES_PER_GROUP (C macro), 1707
SOC_RMT_SUPPORT_DMA (C macro), 1708
SOC_RMT_SUPPORT_RX_DEMODULATION (C macro), 1708
SOC_RMT_SUPPORT_RX_PINGPONG (C macro), 1708
SOC_RMT_SUPPORT_TX_ASYNC_STOP (C macro), 1708
SOC_RMT_SUPPORT_TX_CARRIER_DATA_ONLY (C macro), 1708
SOC_RMT_SUPPORT_TX_LOOP_AUTO_STOP (C macro), 1708
SOC_RMT_SUPPORT_TX_LOOP_COUNT (C macro), 1708
SOC_RMT_SUPPORT_TX_SYNCHRO (C macro), 1708
SOC_RMT_SUPPORT_XTAL (C macro), 1708
SOC_RMT_SUPPORTED (C macro), 1698
SOC_RMT_TX_CANDIDATES_PER_GROUP (C macro), 1707
SOC_RNG_SUPPORTED (C macro), 1700
soc_root_clk_t (C++ enum), 290
soc_root_clk_t::SOC_ROOT_CLK_EXT_OSC_SLOW (C++ enumerator), 290
soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL (C++ enumerator), 290
soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL32K (C++ enumerator), 290
soc_root_clk_t::SOC_ROOT_CLK_INT_RC32K (C++ enumerator), 290
soc_root_clk_t::SOC_ROOT_CLK_INT_RC_FAST (C++ enumerator), 290
soc_root_clk_t::SOC_ROOT_CLK_INT_RC_SLOW (C++ enumerator), 290
SOC_RSA_MAX_BIT_LEN (C macro), 1710
soc_rtc_fast_clk_src_t (C++ enum), 291
soc_rtc_fast_clk_src_t::SOC_RTC_FAST_CLK_SRC_INVA (C++ enumerator), 292
soc_rtc_fast_clk_src_t::SOC_RTC_FAST_CLK_SRC_LP_P (C++ enumerator), 292
soc_rtc_fast_clk_src_t::SOC_RTC_FAST_CLK_SRC_RC_F (C++ enumerator), 291
soc_rtc_fast_clk_src_t::SOC_RTC_FAST_CLK_SRC_XTAL (C++ enumerator), 291
soc_rtc_fast_clk_src_t::SOC_RTC_FAST_CLK_SRC_XTAL (C++ enumerator), 291
SOC_RTC_FAST_MEM_SUPPORTED (C macro), 1698
SOC_RTC_MEM_SUPPORTED (C macro), 1698
soc_rtc_slow_clk_src_t (C++ enum), 291
soc_rtc_slow_clk_src_t::SOC_RTC_SLOW_CLK_SRC_INVA (C++ enumerator), 291

soc_rtc_slow_clk_src_t::SOC_RTC_SLOW_CLK_SRC_RC_SLOW (C++ enumerator), 291
soc_rtc_slow_clk_src_t::SOC_RTC_SLOW_CLK_SRC_XTAL32K (C++ enumerator), 291
soc_rtc_slow_clk_src_t::SOC_RTC_SLOW_CLK_SRC_XTAL32K (C++ enumerator), 291
SOC_RTCIO_HOLD_SUPPORTED (C macro), 1704
SOC_RTCIO_INPUT_OUTPUT_SUPPORTED (C macro), 1704
SOC_RTCIO_PIN_COUNT (C macro), 1704
SOC_RTCIO_WAKE_SUPPORTED (C macro), 1704
SOC_SDM_CHANNELS_PER_GROUP (C macro), 1711
SOC_SDM_CLK_SUPPORT_PLL_F80M (C macro), 1711
SOC_SDM_CLK_SUPPORT_XTAL (C macro), 1711
SOC_SDM_CLKS (C macro), 289
SOC_SDM_GROUPS (C macro), 1711
SOC_SDM_SUPPORTED (C macro), 1698
SOC_SDMMC_CLKS (C macro), 290
SOC_SDMMC_DELAY_PHASE_NUM (C macro), 1710
SOC_SDMMC_HOST_SUPPORTED (C macro), 1699
SOC_SDMMC_IO_POWER_EXTERNAL (C macro), 1710
SOC_SDMMC_NUM_SLOTS (C macro), 1710
SOC_SDMMC_PSRAM_DMA_CAPABLE (C macro), 1710
SOC_SDMMC_USE_GPIO_MATRIX (C macro), 1710
SOC_SDMMC_USE_IOMUX (C macro), 1710
SOC_SECURE_BOOT_SUPPORTED (C macro), 1699
SOC_SECURE_BOOT_V2_ECC (C macro), 1715
SOC_SECURE_BOOT_V2_RSA (C macro), 1715
SOC_SHA_DMA_MAX_BUFFER_SIZE (C macro), 1710
SOC_SHA_GDMA (C macro), 1711
SOC_SHA_SUPPORT_DMA (C macro), 1711
SOC_SHA_SUPPORT_RESUME (C macro), 1711
SOC_SHA_SUPPORT_SHA1 (C macro), 1711
SOC_SHA_SUPPORT_SHA224 (C macro), 1711
SOC_SHA_SUPPORT_SHA256 (C macro), 1711
SOC_SHA_SUPPORT_SHA384 (C macro), 1711
SOC_SHA_SUPPORT_SHA512 (C macro), 1711
SOC_SHA_SUPPORT_SHA512_224 (C macro), 1711
SOC_SHA_SUPPORT_SHA512_256 (C macro), 1711
SOC_SHA_SUPPORT_SHA512_T (C macro), 1711
SOC_SHA_SUPPORTED (C macro), 1698
SOC_SHARED_IDCACHE_SUPPORTED (C macro), 1701
SOC_SPI_CLKS (C macro), 289
SOC_SPI_FLASH_SUPPORTED (C macro), 1699
SOC_SPI_MAX_CS_NUM (C macro), 1711
SOC_SPI_MAX_PRE_DIVIDER (C macro), 1712
SOC_SPI_MAXIMUM_BUFFER_SIZE (C macro), 1711
SOC_SPI_MEM_SUPPORT_AUTO_RESUME (C macro), 1712
SOC_SPI_MEM_SUPPORT_AUTO_SUSPEND (C macro), 1712
SOC_SPI_MEM_SUPPORT_AUTO_WAIT_IDLE (C macro), 1712
SOC_SPI_MEM_SUPPORT_CACHE_32BIT_ADDR_MAP (C macro), 1712
SOC_SPI_MEM_SUPPORT_CHECK_SUS (C macro), 1712
SOC_SPI_MEM_SUPPORT_IDLE_INTR (C macro), 1712
SOC_SPI_MEM_SUPPORT_SW_SUSPEND (C macro), 1712
SOC_SPI_MEM_SUPPORT_TIMING_TUNING (C macro), 1712
SOC_SPI_PERIPH_CS_NUM (C macro), 1711
SOC_SPI_PERIPH_NUM (C macro), 1711
SOC_SPI_PERIPH_SUPPORT_CONTROL_DUMMY_OUT (C macro), 1712
SOC_SPI_PERIPH_SUPPORT_MULTILINE_MODE (C macro), 1712
SOC_SPI_SLAVE_SUPPORT_SEG_TRANS (C macro), 1712
SOC_SPI_SUPPORT_CD_SIG (C macro), 1712
SOC_SPI_SUPPORT_CLK_RC_FAST (C macro), 1712
SOC_SPI_SUPPORT_CLK_SPLL (C macro), 1712
SOC_SPI_SUPPORT_CLK_XTAL (C macro), 1712
SOC_SPI_SUPPORT_DDRCLK (C macro), 1712
SOC_SPI_SUPPORT_OCT (C macro), 1712
SOC_SPI_SUPPORT_SLAVE_HD_VER2 (C macro), 1711
SOC_SPIRAM_SUPPORTED (C macro), 1699
SOC_SPIRAM_XIP_SUPPORTED (C macro), 1713
SOC_SUPPORT_SECURE_BOOT_REVOKE_KEY (C macro), 1715
SOC_SUPPORTS_SECURE_DL_MODE (C macro), 1698
SOC_SYSTIMER_ALARM_MISS_COMPENSATE (C macro), 1713
SOC_SYSTIMER_ALARM_NUM (C macro), 1713
SOC_SYSTIMER_BIT_WIDTH_HI (C macro), 1713
SOC_SYSTIMER_BIT_WIDTH_LO (C macro), 1713
SOC_SYSTIMER_COUNTER_NUM (C macro), 1713
SOC_SYSTIMER_FIXED_DIVIDER (C macro), 1713
SOC_SYSTIMER_INT_LEVEL (C macro), 1713
SOC_SYSTIMER_SUPPORT_ETM (C macro), 1713
SOC_SYSTIMER_SUPPORT_RC_FAST (C macro), 1713
SOC_SYSTIMER_SUPPORTED (C macro), 1698
SOC_TEMP_SENSOR_CLKS (C macro), 290
SOC_TEMP_SENSOR_SUPPORTED (C macro), 1698
SOC_TEMPERATURE_SENSOR_INTR_SUPPORT (C macro), 1717
SOC_TEMPERATURE_SENSOR_LP_PLL_SUPPORT (C macro), 1717
SOC_TEMPERATURE_SENSOR_SUPPORT_ETM (C macro), 1717
SOC_TIMER_GROUP_COUNTER_BIT_WIDTH (C macro), 1713
SOC_TIMER_GROUP_SUPPORT_RC_FAST (C

macro), 1713
SOC_TIMER_GROUP_SUPPORT_XTAL (*C macro*), 1713
SOC_TIMER_GROUP_TIMERS_PER_GROUP (*C macro*), 1713
SOC_TIMER_GROUP_TOTAL_TIMERS (*C macro*), 1713
SOC_TIMER_GROUPS (*C macro*), 1713
SOC_TIMER_SUPPORT_ETM (*C macro*), 1713
SOC_TIMER_SUPPORT_SLEEP_RETENTION (*C macro*), 1713
SOC_TOUCH_PROXIMITY_CHANNEL_NUM (*C macro*), 1714
SOC_TOUCH_PROXIMITY_MEAS_DONE_SUPPORTED (*C macro*), 1714
SOC_TOUCH_SAMPLE_CFG_NUM (*C macro*), 1714
SOC_TOUCH_SENSOR_NUM (*C macro*), 1714
SOC_TOUCH_SENSOR_SUPPORTED (*C macro*), 1700
SOC_TOUCH_SENSOR_VERSION (*C macro*), 1714
SOC_TOUCH_SUPPORT_FREQ_HOP (*C macro*), 1714
SOC_TOUCH_SUPPORT_PROX_SENSING (*C macro*), 1714
SOC_TOUCH_SUPPORT_SLEEP_WAKEUP (*C macro*), 1714
SOC_TOUCH_SUPPORT_WATERPROOF (*C macro*), 1714
SOC_TSENS_IS_INDEPENDENT_FROM_ADC (*C macro*), 1717
SOC_TWAI_BRP_MAX (*C macro*), 1714
SOC_TWAI_BRP_MIN (*C macro*), 1714
SOC_TWAI_CLK_SUPPORT_XTAL (*C macro*), 1714
SOC_TWAI_CLKS (*C macro*), 289
SOC_TWAI_CONTROLLER_NUM (*C macro*), 1714
SOC_TWAI_SUPPORTED (*C macro*), 1697
SOC_TWAI_SUPPORTS_RX_STATUS (*C macro*), 1714
SOC_UART_BITRATE_MAX (*C macro*), 1715
SOC_UART_CLKS (*C macro*), 288
SOC_UART_FIFO_LEN (*C macro*), 1715
SOC_UART_HAS_LP_UART (*C macro*), 1716
SOC_UART_HP_NUM (*C macro*), 1715
SOC_UART_LP_NUM (*C macro*), 1715
SOC_UART_NUM (*C macro*), 1715
SOC_UART_SUPPORT_FSM_TX_WAIT_SEND (*C macro*), 1716
SOC_UART_SUPPORT_PLL_F80M_CLK (*C macro*), 1715
SOC_UART_SUPPORT_RTC_CLK (*C macro*), 1716
SOC_UART_SUPPORT_SLEEP_RETENTION (*C macro*), 1716
SOC_UART_SUPPORT_WAKEUP_INT (*C macro*), 1716
SOC_UART_SUPPORT_XTAL_CLK (*C macro*), 1716
SOC_UART_SUPPORTED (*C macro*), 1697
SOC_ULP_LP_UART_SUPPORTED (*C macro*), 1699
SOC_ULP_SUPPORTED (*C macro*), 1698
SOC_USB_OTG_SUPPORTED (*C macro*), 1697
SOC_USB_SERIAL_JTAG_SUPPORTED (*C macro*), 1698
SOC_WDT_SUPPORTED (*C macro*), 1699
SOC_WIFI_LIGHT_SLEEP_CLK_WIDTH (*C macro*), 1716
SOC_WIRELESS_HOST_SUPPORTED (*C macro*), 1697
soc_xtal_freq_t (*C++ enum*), 292
soc_xtal_freq_t::SOC_XTAL_FREQ_40M (*C++ enumerator*), 292
SOC_XTAL_SUPPORT_40M (*C macro*), 1700
spi_bus_add_device (*C++ function*), 707
spi_bus_add_flash_device (*C++ function*), 672
spi_bus_config_t (*C++ struct*), 704
spi_bus_config_t::data0_io_num (*C++ member*), 704
spi_bus_config_t::data1_io_num (*C++ member*), 704
spi_bus_config_t::data2_io_num (*C++ member*), 704
spi_bus_config_t::data3_io_num (*C++ member*), 704
spi_bus_config_t::data4_io_num (*C++ member*), 704
spi_bus_config_t::data5_io_num (*C++ member*), 704
spi_bus_config_t::data6_io_num (*C++ member*), 705
spi_bus_config_t::data7_io_num (*C++ member*), 705
spi_bus_config_t::flags (*C++ member*), 705
spi_bus_config_t::intr_flags (*C++ member*), 705
spi_bus_config_t::isr_cpu_id (*C++ member*), 705
spi_bus_config_t::max_transfer_sz (*C++ member*), 705
spi_bus_config_t::miso_io_num (*C++ member*), 704
spi_bus_config_t::mosi_io_num (*C++ member*), 704
spi_bus_config_t::quadhd_io_num (*C++ member*), 704
spi_bus_config_t::quadwp_io_num (*C++ member*), 704
spi_bus_config_t::sclk_io_num (*C++ member*), 704
spi_bus_free (*C++ function*), 703
spi_bus_get_max_transaction_len (*C++ function*), 711
spi_bus_initialize (*C++ function*), 703
spi_bus_remove_device (*C++ function*), 707
spi_bus_remove_flash_device (*C++ function*), 672
spi_clock_source_t (*C++ type*), 701
spi_command_t (*C++ enum*), 702
spi_command_t::SPI_CMD_HD_EN_QPI (*C++ enumerator*), 702

spi_command_t::SPI_CMD_HD_INT0 (C++ enumerator), 702
spi_command_t::SPI_CMD_HD_INT1 (C++ enumerator), 702
spi_command_t::SPI_CMD_HD_INT2 (C++ enumerator), 702
spi_command_t::SPI_CMD_HD_RDBUF (C++ enumerator), 702
spi_command_t::SPI_CMD_HD_RDDMA (C++ enumerator), 702
spi_command_t::SPI_CMD_HD_SEG_END (C++ enumerator), 702
spi_command_t::SPI_CMD_HD_WR_END (C++ enumerator), 702
spi_command_t::SPI_CMD_HD_WRBUF (C++ enumerator), 702
spi_command_t::SPI_CMD_HD_WRDMA (C++ enumerator), 702
spi_common_dma_t (C++ enum), 706
spi_common_dma_t::SPI_DMA_CH_AUTO (C++ enumerator), 707
spi_common_dma_t::SPI_DMA_DISABLED (C++ enumerator), 706
SPI_DEVICE_3WIRE (C macro), 715
spi_device_acquire_bus (C++ function), 710
SPI_DEVICE_BIT_LSBFIRST (C macro), 715
SPI_DEVICE_CLK_AS_CS (C macro), 715
SPI_DEVICE_DDRCLK (C macro), 715
spi_device_get_actual_freq (C++ function), 710
spi_device_get_trans_result (C++ function), 708
SPI_DEVICE_HALFDUPLEX (C macro), 715
spi_device_handle_t (C++ type), 716
spi_device_interface_config_t (C++ struct), 711
spi_device_interface_config_t::address_bits (C++ member), 711
spi_device_interface_config_t::clock_source (C++ member), 712
spi_device_interface_config_t::clock_speed_hz (C++ member), 712
spi_device_interface_config_t::command_bits (C++ member), 711
spi_device_interface_config_t::cs_ena_posttrans (C++ member), 712
spi_device_interface_config_t::cs_ena_pretrans (C++ member), 712
spi_device_interface_config_t::dummy_bits (C++ member), 711
spi_device_interface_config_t::duty_cycle_pos (C++ member), 712
spi_device_interface_config_t::flags (C++ member), 712
spi_device_interface_config_t::input_delay_ns (C++ member), 712
spi_device_interface_config_t::mode (C++ member), 711
spi_device_interface_config_t::post_cb (C++ member), 712
spi_device_interface_config_t::pre_cb (C++ member), 712
spi_device_interface_config_t::queue_size (C++ member), 712
spi_device_interface_config_t::spics_io_num (C++ member), 712
SPI_DEVICE_NO_DUMMY (C macro), 715
SPI_DEVICE_NO_RETURN_RESULT (C macro), 715
spi_device_polling_end (C++ function), 709
spi_device_polling_start (C++ function), 709
spi_device_polling_transmit (C++ function), 709
SPI_DEVICE_POSITIVE_CS (C macro), 715
spi_device_queue_trans (C++ function), 708
spi_device_release_bus (C++ function), 710
SPI_DEVICE_RXBIT_LSBFIRST (C macro), 714
spi_device_transmit (C++ function), 708
SPI_DEVICE_TXBIT_LSBFIRST (C macro), 714
spi_dma_chan_t (C++ type), 706
spi_event_t (C++ enum), 701
spi_event_t::SPI_EV_BUF_RX (C++ enumerator), 701
spi_event_t::SPI_EV_BUF_TX (C++ enumerator), 701
spi_event_t::SPI_EV_CMD9 (C++ enumerator), 702
spi_event_t::SPI_EV_CMDA (C++ enumerator), 702
spi_event_t::SPI_EV_RECV (C++ enumerator), 702
spi_event_t::SPI_EV_RECV_DMA_READY (C++ enumerator), 702
spi_event_t::SPI_EV_SEND (C++ enumerator), 702
spi_event_t::SPI_EV_SEND_DMA_READY (C++ enumerator), 702
spi_event_t::SPI_EV_TRANS (C++ enumerator), 702
spi_flash_cache2phys (C++ function), 682
SPI_FLASH_CACHE2PHYS_FAIL (C macro), 683
spi_flash_chip_t (C++ type), 681
SPI_FLASH_CONFIG_CONF_BITS (C macro), 688
spi_flash_counter_t (C++ type), 692
spi_flash_counters_t (C++ type), 692
spi_flash_dump_counters (C++ function), 691
spi_flash_encryption_t (C++ struct), 685
spi_flash_encryption_t::flash_encryption_check (C++ member), 686
spi_flash_encryption_t::flash_encryption_data_pre (C++ member), 685
spi_flash_encryption_t::flash_encryption_destroy (C++ member), 685
spi_flash_encryption_t::flash_encryption_disable (C++ member), 685

spi_flash_encryption_t::flash_encryption_enable (C++ member), 685
spi_flash_encryption_t::flash_encryption_mode (C++ member), 685
spi_flash_get_counters (C++ function), 691
spi_flash_host_driver_s (C++ struct), 686
spi_flash_host_driver_s::check_suspend (C++ member), 687
spi_flash_host_driver_s::common_command (C++ member), 686
spi_flash_host_driver_s::configure_host_io_mode (C++ member), 687
spi_flash_host_driver_s::dev_config (C++ member), 686
spi_flash_host_driver_s::erase_block (C++ member), 686
spi_flash_host_driver_s::erase_chip (C++ member), 686
spi_flash_host_driver_s::erase_sector (C++ member), 686
spi_flash_host_driver_s::flush_cache (C++ member), 687
spi_flash_host_driver_s::host_status (C++ member), 687
spi_flash_host_driver_s::poll_cmd_done (C++ member), 687
spi_flash_host_driver_s::program_page (C++ member), 686
spi_flash_host_driver_s::read (C++ member), 687
spi_flash_host_driver_s::read_data_slicer (C++ member), 687
spi_flash_host_driver_s::read_id (C++ member), 686
spi_flash_host_driver_s::read_status (C++ member), 686
spi_flash_host_driver_s::resume (C++ member), 687
spi_flash_host_driver_s::set_write_protect (C++ member), 686
spi_flash_host_driver_s::supports_direct_read (C++ member), 687
spi_flash_host_driver_s::supports_direct_write (C++ member), 687
spi_flash_host_driver_s::sus_setup (C++ member), 688
spi_flash_host_driver_s::suspend (C++ member), 688
spi_flash_host_driver_s::write_data_slicer (C++ member), 687
spi_flash_host_driver_t (C++ type), 688
spi_flash_host_inst_t (C++ struct), 686
spi_flash_host_inst_t::driver (C++ member), 686
spi_flash_mmap (C++ function), 681
spi_flash_mmap_dump (C++ function), 682
spi_flash_mmap_get_free_pages (C++ function), 682
spi_flash_mmap_handle_t (C++ type), 683
spi_flash_mmap_memory_t (C++ enum), 683
spi_flash_mmap_memory_t::SPI_FLASH_MMAP_DATA (C++ enumerator), 683
spi_flash_mmap_memory_t::SPI_FLASH_MMAP_INST (C++ enumerator), 684
spi_flash_mmap_pages (C++ function), 682
SPI_FLASH_MMU_PAGE_SIZE (C macro), 683
spi_flash_munmap (C++ function), 682
SPI_FLASH_OPI_FLAG (C macro), 688
SPI_FLASH_OS_IS_ERASING_STATUS_FLAG (C macro), 681
spi_flash_phys2cache (C++ function), 682
SPI_FLASH_READ_MODE_MIN (C macro), 688
spi_flash_reset_counters (C++ function), 691
SPI_FLASH_SEC_SIZE (C macro), 683
spi_flash_sus_cmd_conf (C++ struct), 685
spi_flash_sus_cmd_conf::cmd_rdsr (C++ member), 685
spi_flash_sus_cmd_conf::res_cmd (C++ member), 685
spi_flash_sus_cmd_conf::reserved (C++ member), 685
spi_flash_sus_cmd_conf::sus_cmd (C++ member), 685
spi_flash_sus_cmd_conf::sus_mask (C++ member), 685
SPI_FLASH_TRANS_FLAG_BYTE_SWAP (C macro), 688
SPI_FLASH_TRANS_FLAG_CMD16 (C macro), 688
SPI_FLASH_TRANS_FLAG_IGNORE_BASEIO (C macro), 688
SPI_FLASH_TRANS_FLAG_PE_CMD (C macro), 688
spi_flash_trans_t (C++ struct), 684
spi_flash_trans_t::address (C++ member), 684
spi_flash_trans_t::address_bitlen (C++ member), 684
spi_flash_trans_t::command (C++ member), 684
spi_flash_trans_t::dummy_bitlen (C++ member), 684
spi_flash_trans_t::flags (C++ member), 684
spi_flash_trans_t::io_mode (C++ member), 684
spi_flash_trans_t::miso_data (C++ member), 684
spi_flash_trans_t::miso_len (C++ member), 684
spi_flash_trans_t::mosi_data (C++ member), 684
spi_flash_trans_t::mosi_len (C++ member), 684
spi_flash_trans_t::reserved (C++ member), 684

SPI_FLASH_YIELD_REQ_SUSPEND (*C macro*), 681
SPI_FLASH_YIELD_REQ_YIELD (*C macro*), 681
SPI_FLASH_YIELD_STA_RESUME (*C macro*), 681
spi_get_actual_clock (*C++ function*), 710
spi_get_freq_limit (*C++ function*), 711
spi_get_timing (*C++ function*), 710
spi_host_device_t (*C++ enum*), 701
spi_host_device_t::SPI1_HOST (*C++ enumerator*), 701
spi_host_device_t::SPI2_HOST (*C++ enumerator*), 701
spi_host_device_t::SPI3_HOST (*C++ enumerator*), 701
spi_host_device_t::SPI_HOST_MAX (*C++ enumerator*), 701
spi_line_mode_t (*C++ struct*), 701
spi_line_mode_t::addr_lines (*C++ member*), 701
spi_line_mode_t::cmd_lines (*C++ member*), 701
spi_line_mode_t::data_lines (*C++ member*), 701
SPI_MASTER_FREQ_10M (*C macro*), 714
SPI_MASTER_FREQ_11M (*C macro*), 714
SPI_MASTER_FREQ_13M (*C macro*), 714
SPI_MASTER_FREQ_16M (*C macro*), 714
SPI_MASTER_FREQ_20M (*C macro*), 714
SPI_MASTER_FREQ_26M (*C macro*), 714
SPI_MASTER_FREQ_40M (*C macro*), 714
SPI_MASTER_FREQ_80M (*C macro*), 714
SPI_MASTER_FREQ_8M (*C macro*), 714
SPI_MASTER_FREQ_9M (*C macro*), 714
SPI_MAX_DMA_LEN (*C macro*), 705
SPI_SLAVE_BIT_LSBFIRST (*C macro*), 722
spi_slave_chan_t (*C++ enum*), 730
spi_slave_chan_t::SPI_SLAVE_CHAN_RX (*C++ enumerator*), 731
spi_slave_chan_t::SPI_SLAVE_CHAN_TX (*C++ enumerator*), 730
spi_slave_free (*C++ function*), 720
spi_slave_get_trans_result (*C++ function*), 720
SPI_SLAVE_HD_APPEND_MODE (*C macro*), 730
spi_slave_hd_append_trans (*C++ function*), 727
SPI_SLAVE_HD_BIT_LSBFIRST (*C macro*), 730
spi_slave_hd_callback_config_t (*C++ struct*), 728
spi_slave_hd_callback_config_t::arg (*C++ member*), 729
spi_slave_hd_callback_config_t::cb_buffer_rx (*C++ member*), 729
spi_slave_hd_callback_config_t::cb_buffer_tx (*C++ member*), 729
spi_slave_hd_callback_config_t::cb_cmd9 (*C++ member*), 729
spi_slave_hd_callback_config_t::cb_cmdA (*C++ member*), 729
spi_slave_hd_callback_config_t::cb_recv (*C++ member*), 729
spi_slave_hd_callback_config_t::cb_recv_dma_ready (*C++ member*), 729
spi_slave_hd_callback_config_t::cb_send_dma_ready (*C++ member*), 729
spi_slave_hd_callback_config_t::cb_sent (*C++ member*), 729
spi_slave_hd_data_t (*C++ struct*), 728
spi_slave_hd_data_t::arg (*C++ member*), 728
spi_slave_hd_data_t::data (*C++ member*), 728
spi_slave_hd_data_t::flags (*C++ member*), 728
spi_slave_hd_data_t::len (*C++ member*), 728
spi_slave_hd_data_t::trans_len (*C++ member*), 728
spi_slave_hd_deinit (*C++ function*), 726
spi_slave_hd_event_t (*C++ struct*), 728
spi_slave_hd_event_t::event (*C++ member*), 728
spi_slave_hd_event_t::trans (*C++ member*), 728
spi_slave_hd_get_append_trans_res (*C++ function*), 727
spi_slave_hd_get_trans_res (*C++ function*), 726
spi_slave_hd_init (*C++ function*), 726
spi_slave_hd_queue_trans (*C++ function*), 726
spi_slave_hd_read_buffer (*C++ function*), 727
SPI_SLAVE_HD_RXBIT_LSBFIRST (*C macro*), 730
spi_slave_hd_slot_config_t (*C++ struct*), 729
spi_slave_hd_slot_config_t::address_bits (*C++ member*), 730
spi_slave_hd_slot_config_t::cb_config (*C++ member*), 730
spi_slave_hd_slot_config_t::command_bits (*C++ member*), 729
spi_slave_hd_slot_config_t::dma_chan (*C++ member*), 730
spi_slave_hd_slot_config_t::dummy_bits (*C++ member*), 730
spi_slave_hd_slot_config_t::flags (*C++ member*), 729
spi_slave_hd_slot_config_t::mode (*C++ member*), 729
spi_slave_hd_slot_config_t::queue_size (*C++ member*), 730
spi_slave_hd_slot_config_t::spics_io_num (*C++ member*), 729
SPI_SLAVE_HD_TRANS_DMA_BUFFER_ALIGN_AUTO

(C macro), 730
SPI_SLAVE_HD_TXBIT_LSBFIRST (C macro), 730
spi_slave_hd_write_buffer (C++ function), 727
spi_slave_initialize (C++ function), 719
spi_slave_interface_config_t (C++ struct), 721
spi_slave_interface_config_t::flags (C++ member), 721
spi_slave_interface_config_t::mode (C++ member), 721
spi_slave_interface_config_t::post_setup_cb (C++ member), 722
spi_slave_interface_config_t::post_trans_cb (C++ member), 722
spi_slave_interface_config_t::queue_size (C++ member), 721
spi_slave_interface_config_t::spics_io_num (C++ member), 721
SPI_SLAVE_NO_RETURN_RESULT (C macro), 722
spi_slave_queue_trans (C++ function), 720
SPI_SLAVE_RXBIT_LSBFIRST (C macro), 722
SPI_SLAVE_TRANS_DMA_BUFFER_ALIGN_AUTO (C macro), 723
spi_slave_transaction_t (C++ struct), 722
spi_slave_transaction_t::flags (C++ member), 722
spi_slave_transaction_t::length (C++ member), 722
spi_slave_transaction_t::rx_buffer (C++ member), 722
spi_slave_transaction_t::trans_len (C++ member), 722
spi_slave_transaction_t::tx_buffer (C++ member), 722
spi_slave_transaction_t::user (C++ member), 722
spi_slave_transmit (C++ function), 721
SPI_SLAVE_TXBIT_LSBFIRST (C macro), 722
SPI_SWAP_DATA_RX (C macro), 705
SPI_SWAP_DATA_TX (C macro), 705
SPI_TRANS_CS_KEEP_ACTIVE (C macro), 716
SPI_TRANS_DMA_BUFFER_ALIGN_MANUAL (C macro), 716
SPI_TRANS_MODE_DIO (C macro), 715
SPI_TRANS_MODE_DIOQIO_ADDR (C macro), 715
SPI_TRANS_MODE_OCT (C macro), 716
SPI_TRANS_MODE_QIO (C macro), 715
SPI_TRANS_MULTILINE_ADDR (C macro), 716
SPI_TRANS_MULTILINE_CMD (C macro), 716
SPI_TRANS_USE_RXDATA (C macro), 715
SPI_TRANS_USE_TXDATA (C macro), 715
SPI_TRANS_VARIABLE_ADDR (C macro), 716
SPI_TRANS_VARIABLE_CMD (C macro), 715
SPI_TRANS_VARIABLE_DUMMY (C macro), 716
spi_transaction_ext_t (C++ struct), 713
spi_transaction_ext_t::address_bits (C++ member), 714
spi_transaction_ext_t::base (C++ member), 713
spi_transaction_ext_t::command_bits (C++ member), 714
spi_transaction_ext_t::dummy_bits (C++ member), 714
spi_transaction_t (C++ struct), 712
spi_transaction_t::addr (C++ member), 713
spi_transaction_t::cmd (C++ member), 713
spi_transaction_t::flags (C++ member), 713
spi_transaction_t::length (C++ member), 713
spi_transaction_t::rx_buffer (C++ member), 713
spi_transaction_t::rx_data (C++ member), 713
spi_transaction_t::rxlength (C++ member), 713
spi_transaction_t::tx_buffer (C++ member), 713
spi_transaction_t::tx_data (C++ member), 713
spi_transaction_t::user (C++ member), 713
SPICOMMON_BUSFLAG_DUAL (C macro), 706
SPICOMMON_BUSFLAG_GPIO_PINS (C macro), 706
SPICOMMON_BUSFLAG_IO4_IO7 (C macro), 706
SPICOMMON_BUSFLAG_IOMUX_PINS (C macro), 706
SPICOMMON_BUSFLAG_MASTER (C macro), 706
SPICOMMON_BUSFLAG_MISO (C macro), 706
SPICOMMON_BUSFLAG_MOSI (C macro), 706
SPICOMMON_BUSFLAG_NATIVE_PINS (C macro), 706
SPICOMMON_BUSFLAG_OCTAL (C macro), 706
SPICOMMON_BUSFLAG_QUAD (C macro), 706
SPICOMMON_BUSFLAG_SCLK (C macro), 706
SPICOMMON_BUSFLAG_SLAVE (C macro), 705
SPICOMMON_BUSFLAG_WPHD (C macro), 706
StaticRingbuffer_t (C++ type), 1563
StreamBufferCallbackFunction_t (C++ type), 1536
StreamBufferHandle_t (C++ type), 1536

T

task_wdt_msg_handler (C++ type), 1758
taskDISABLE_INTERRUPTS (C macro), 1460
taskENABLE_INTERRUPTS (C macro), 1460
taskENTER_CRITICAL (C macro), 1460
taskENTER_CRITICAL_FROM_ISR (C macro), 1460
taskENTER_CRITICAL_ISR (C macro), 1460
taskEXIT_CRITICAL (C macro), 1460
taskEXIT_CRITICAL_FROM_ISR (C macro), 1460
taskEXIT_CRITICAL_ISR (C macro), 1460
TaskHandle_t (C++ type), 1465

TaskHookFunction_t (*C++ type*), 1465
taskSCHEDULER_NOT_STARTED (*C macro*), 1460
taskSCHEDULER_RUNNING (*C macro*), 1460
taskSCHEDULER_SUSPENDED (*C macro*), 1460
TaskStatus_t (*C++ type*), 1465
taskVALID_CORE_ID (*C macro*), 1460
taskYIELD (*C macro*), 1460
temperature_sensor_abs_threshold_config_t (*C++ struct*), 754
temperature_sensor_abs_threshold_config_t::high_threshold (*C++ member*), 754
temperature_sensor_abs_threshold_config_t::low_threshold (*C++ member*), 754
temperature_sensor_clk_src_t (*C++ type*), 755
TEMPERATURE_SENSOR_CONFIG_DEFAULT (*C macro*), 754
temperature_sensor_config_t (*C++ struct*), 753
temperature_sensor_config_t::clk_src (*C++ member*), 753
temperature_sensor_config_t::range_max (*C++ member*), 753
temperature_sensor_config_t::range_min (*C++ member*), 753
temperature_sensor_delta_threshold_config_t (*C++ struct*), 754
temperature_sensor_delta_threshold_config_t::delta_thresh (*C++ member*), 754
temperature_sensor_delta_threshold_config_t::intr_delta (*C++ member*), 754
temperature_sensor_disable (*C++ function*), 751
temperature_sensor_enable (*C++ function*), 751
temperature_sensor_etm_event_config_t (*C++ struct*), 756
temperature_sensor_etm_event_config_t::event_type (*C++ member*), 757
temperature_sensor_etm_event_type_t (*C++ enum*), 755
temperature_sensor_etm_event_type_t::TEMPERATURE_SENSOR_ETM_EVENT_MAX (*C++ enumerator*), 755
temperature_sensor_etm_event_type_t::TEMPERATURE_SENSOR_ETM_EVENT_OVER_LIMIT (*C++ enumerator*), 755
temperature_sensor_etm_task_config_t (*C++ struct*), 757
temperature_sensor_etm_task_config_t::task_type (*C++ member*), 757
temperature_sensor_etm_task_type_t (*C++ enum*), 755
temperature_sensor_etm_task_type_t::TEMPERATURE_SENSOR_ETM_TASK_MAX (*C++ enumerator*), 755
temperature_sensor_etm_task_type_t::TEMPERATURE_SENSOR_TASK_START (*C++ enumerator*), 755
temperature_sensor_etm_task_type_t::TEMPERATURE_SENSOR_TASK_STOP (*C++ enumerator*), 755
temperature_sensor_event_callbacks_t (*C++ struct*), 753
temperature_sensor_event_callbacks_t::on_threshold (*C++ member*), 754
temperature_sensor_get_celsius (*C++ function*), 752
temperature_sensor_handle_t (*C++ type*), 754
temperature_sensor_install (*C++ function*), 751
temperature_sensor_new_etm_event (*C++ function*), 756
temperature_sensor_new_etm_task (*C++ function*), 756
temperature_sensor_register_callbacks (*C++ function*), 752
temperature_sensor_set_absolute_threshold (*C++ function*), 752
temperature_sensor_set_delta_threshold (*C++ function*), 752
temperature_sensor_threshold_event_data_t (*C++ struct*), 753
temperature_sensor_threshold_event_data_t::celsius_value (*C++ member*), 753
temperature_sensor_threshold_event_data_t::intr_condition (*C++ member*), 753
temperature_sensor_uninstall (*C++ function*), 751
temperature_val_intr_condition_t (*C++ enum*), 755
temperature_val_intr_condition_t::TEMPERATURE_VAL_INTR_DELTA (*C++ enumerator*), 755
temperature_val_intr_condition_t::TEMPERATURE_VAL_INTR_THRESH (*C++ enumerator*), 755
TimerCallbackFunction_t (*C++ type*), 1518
TimerHandle_t (*C++ type*), 1518
tls_keep_alive_cfg (*C++ struct*), 62
tls_keep_alive_cfg::keep_alive_count (*C++ member*), 63
tls_keep_alive_cfg::keep_alive_enable (*C++ member*), 63
tls_keep_alive_cfg::keep_alive_idle (*C++ member*), 63
tls_keep_alive_cfg::keep_alive_interval (*C++ member*), 63
tls_keep_alive_cfg_t (*C++ type*), 67
TlsDeleteCallbackFunction_t (*C++ type*), 1572
topic_t (*C++ struct*), 49
topic_t::filter (*C++ member*), 49
topic_t::qos (*C++ member*), 49
TOUCH_BASE_EVENT_MAX (*C++ type*), 777
touch_base_event_data_t (*C++ struct*), 774
touch_base_event_data_t::chan (*C++ member*), 775
touch_base_event_data_t::chan_id (*C++ member*), 775
touch_base_event_data_t::status_mask

(C++ member), 775
touch_benchmark_filter_mode_t (C++ enum), 779
touch_benchmark_filter_mode_t::TOUCH_BM_IIR_FILTER (C++ enumerator), 779
touch_benchmark_filter_mode_t::TOUCH_BM_JITTER_FILTER (C++ enumerator), 779
touch_benchmark_filter_mode_t::TOUCH_BM_MAX_FILTER_EVENT (C++ enumerator), 779
touch_benchmark_filter_mode_t::TOUCH_BM_MEDIAN_FILTER (C++ enumerator), 779
touch_benchmark_filter_mode_t::TOUCH_BM_NO_FILTER (C++ enumerator), 779
touch_chan_benchmark_config_t (C++ struct), 776
touch_chan_benchmark_config_t::do_reset (C++ member), 776
touch_chan_data_type_t (C++ enum), 778
touch_chan_data_type_t::TOUCH_CHAN_DATA_TYPE_BENCHMARK (C++ enumerator), 778
touch_chan_data_type_t::TOUCH_CHAN_DATA_TYPE_PROXIMITY (C++ enumerator), 778
touch_chan_data_type_t::TOUCH_CHAN_DATA_TYPE_SMOOTH (C++ enumerator), 778
touch_chan_shield_cap_t (C++ enum), 778
touch_chan_shield_cap_t::TOUCH_SHIELD_CAP_120PF (C++ enumerator), 778
touch_chan_shield_cap_t::TOUCH_SHIELD_CAP_160PF (C++ enumerator), 778
touch_chan_shield_cap_t::TOUCH_SHIELD_CAP_200PF (C++ enumerator), 778
touch_chan_shield_cap_t::TOUCH_SHIELD_CAP_240PF (C++ enumerator), 779
touch_chan_shield_cap_t::TOUCH_SHIELD_CAP_280PF (C++ enumerator), 779
touch_chan_shield_cap_t::TOUCH_SHIELD_CAP_320PF (C++ enumerator), 779
touch_chan_shield_cap_t::TOUCH_SHIELD_CAP_40PF (C++ enumerator), 778
touch_chan_shield_cap_t::TOUCH_SHIELD_CAP_80PF (C++ enumerator), 778
touch_channel_config_benchmark (C++ function), 768
touch_channel_config_t (C++ struct), 772
touch_channel_config_t::active_thresh (C++ member), 772
touch_channel_handle_t (C++ type), 770
touch_channel_read_data (C++ function), 768
touch_event_callbacks_t (C++ struct), 775
touch_event_callbacks_t::on_active (C++ member), 775
touch_event_callbacks_t::on_inactive (C++ member), 775
touch_event_callbacks_t::on_measure_done (C++ member), 775
touch_event_callbacks_t::on_proximity_meas_done (C++ member), 776
touch_event_callbacks_t::on_scan_done (C++ member), 775
touch_event_callbacks_t::on_timeout (C++ member), 776
touch_event_data_t (C++ type), 777
touch_intr_event_t (C++ enum), 780
touch_intr_event_t::TOUCH_INTR_EVENT_ACTIVE (C++ enumerator), 780
touch_intr_event_t::TOUCH_INTR_EVENT_INACTIVE (C++ enumerator), 780
touch_intr_event_t::TOUCH_INTR_EVENT_MEASURE_DONE (C++ enumerator), 780
touch_intr_event_t::TOUCH_INTR_EVENT_PROXIMITY_DONE (C++ enumerator), 780
touch_intr_event_t::TOUCH_INTR_EVENT_SCAN_DONE (C++ enumerator), 780
touch_intr_event_t::TOUCH_INTR_EVENT_TIMEOUT (C++ enumerator), 780
TOUCH_MAX_CHAN_ID (C macro), 776
touch_meas_done_event_data_t (C++ type), 777
TOUCH_MIN_CHAN_ID (C macro), 776
touch_smooth_done_event_data_t (C++ type), 777
TOUCH_PROXIMITY_CHAN_NUM (C macro), 770
touch_proximity_config_t (C++ struct), 774
touch_proximity_config_t::charge_times (C++ member), 774
touch_proximity_config_t::proximity_chan (C++ member), 774
touch_proximity_config_t::scan_times (C++ member), 774
TOUCH_SAMPLE_CFG_NUM (C macro), 770
touch_scan_done_event_data_t (C++ type), 777
touch_sensor_config_dslp_t (C++ type), 777
touch_sensor_config_filter (C++ function), 766
touch_sensor_config_proximity_sensing (C++ function), 769
touch_sensor_config_sleep_wakeup (C++ function), 769
touch_sensor_config_t (C++ struct), 771
touch_sensor_config_t::max_meas_time_us (C++ member), 772
touch_sensor_config_t::meas_interval_us (C++ member), 772
touch_sensor_config_t::output_mode (C++ member), 772
touch_sensor_config_t::power_on_wait_us (C++ member), 771
touch_sensor_config_t::sample_cfg (C++ member), 772
touch_sensor_config_t::sample_cfg_num

(C++ member), 772
touch_sensor_config_waterproof (C++ function), 769
TOUCH_SENSOR_DEFAULT_BASIC_CONFIG (C macro), 776
TOUCH_SENSOR_DEFAULT_FILTER_CONFIG (C macro), 777
touch_sensor_del_channel (C++ function), 765
touch_sensor_del_controller (C++ function), 764
touch_sensor_disable (C++ function), 766
touch_sensor_enable (C++ function), 766
touch_sensor_filter_config_t (C++ struct), 772
touch_sensor_filter_config_t::active_hysteresis (C++ member), 773
touch_sensor_filter_config_t::benchmark (C++ member), 772
touch_sensor_filter_config_t::data (C++ member), 773
touch_sensor_filter_config_t::debounce_cnt (C++ member), 773
touch_sensor_filter_config_t::denoise_lvl (C++ member), 772
touch_sensor_filter_config_t::filter_mode (C++ member), 772
touch_sensor_filter_config_t::jitter_step (C++ member), 772
touch_sensor_filter_config_t::smooth_filter (C++ member), 773
touch_sensor_handle_t (C++ type), 770
touch_sensor_new_channel (C++ function), 764
touch_sensor_new_controller (C++ function), 764
touch_sensor_reconfig_channel (C++ function), 765
touch_sensor_reconfig_controller (C++ function), 765
touch_sensor_register_callbacks (C++ function), 768
touch_sensor_sample_config_t (C++ struct), 771
touch_sensor_sample_config_t::bias_volt (C++ member), 771
touch_sensor_sample_config_t::bypass_shield_output (C++ member), 771
touch_sensor_sample_config_t::charge_times (C++ member), 771
touch_sensor_sample_config_t::div_num (C++ member), 771
touch_sensor_sample_config_t::high_drv (C++ member), 771
touch_sensor_sample_config_t::low_drv (C++ member), 771
touch_sensor_sample_config_t::rc_filter_cap (C++ member), 771
touch_sensor_sample_config_t::rc_filter_res (C++ member), 771
touch_sensor_start_continuous_scanning (C++ function), 767
touch_sensor_stop_continuous_scanning (C++ function), 767
touch_sensor_trigger_oneshot_scanning (C++ function), 767
TOUCH_SENSOR_V3_DEFAULT_SAMPLE_CONFIG (C macro), 777
touch_sleep_config_t (C++ struct), 773
touch_sleep_config_t::deep_slp_chan (C++ member), 773
touch_sleep_config_t::deep_slp_sens_cfg (C++ member), 773
touch_sleep_config_t::deep_slp_thresh (C++ member), 773
touch_sleep_config_t::slp_wakeup_lvl (C++ member), 773
touch_sleep_wakeup_level_t (C++ enum), 778
touch_sleep_wakeup_level_t::TOUCH_DEEP_SLEEP_WAKE (C++ enumerator), 778
touch_sleep_wakeup_level_t::TOUCH_LIGHT_SLEEP_WAK (C++ enumerator), 778
touch_smooth_filter_mode_t (C++ enum), 779
touch_smooth_filter_mode_t::TOUCH_SMOOTH_IIR_FILT (C++ enumerator), 779
touch_smooth_filter_mode_t::TOUCH_SMOOTH_IIR_FILT (C++ enumerator), 780
touch_smooth_filter_mode_t::TOUCH_SMOOTH_IIR_FILT (C++ enumerator), 780
touch_smooth_filter_mode_t::TOUCH_SMOOTH_NO_FILTE (C++ enumerator), 779
touch_timeout_event_data_t (C++ type), 777
TOUCH_TOTAL_CHAN_NUM (C macro), 770
touch_waterproof_config_t (C++ struct), 773
touch_waterproof_config_t::flags (C++ member), 774
touch_waterproof_config_t::guard_chan (C++ member), 774
touch_waterproof_config_t::immersion_proof (C++ member), 774
touch_waterproof_config_t::shield_chan (C++ member), 774
touch_waterproof_config_t::shield_drv (C++ member), 774
transaction_cb_t (C++ type), 716
tskIDLE_PRIORITY (C macro), 1460
tskNO_AFFINITY (C macro), 1460
twai_clear_receive_queue (C++ function), 800
twai_clear_receive_queue_v2 (C++ function), 800
twai_clear_transmit_queue (C++ function), 800
twai_clear_transmit_queue_v2 (C++ func-

tion), 800
twai_clock_source_t (C++ type), 793
twai_driver_install (C++ function), 794
twai_driver_install_v2 (C++ function), 794
twai_driver_uninstall (C++ function), 795
twai_driver_uninstall_v2 (C++ function), 795
TWAI_ERR_PASS_THRESH (C macro), 793
TWAI_EXTD_ID_MASK (C macro), 793
twai_filter_config_t (C++ struct), 792
twai_filter_config_t::acceptance_code (C++ member), 792
twai_filter_config_t::acceptance_mask (C++ member), 792
twai_filter_config_t::single_filter (C++ member), 793
TWAI_FRAME_EXTD_ID_LEN_BYTES (C macro), 793
TWAI_FRAME_MAX_DLC (C macro), 793
TWAI_FRAME_STD_ID_LEN_BYTES (C macro), 793
twai_general_config_t (C++ struct), 801
twai_general_config_t::alerts_enabled (C++ member), 801
twai_general_config_t::bus_off_io (C++ member), 801
twai_general_config_t::clkout_divider (C++ member), 801
twai_general_config_t::clkout_io (C++ member), 801
twai_general_config_t::controller_id (C++ member), 801
twai_general_config_t::intr_flags (C++ member), 801
twai_general_config_t::mode (C++ member), 801
twai_general_config_t::rx_io (C++ member), 801
twai_general_config_t::rx_queue_len (C++ member), 801
twai_general_config_t::tx_io (C++ member), 801
twai_general_config_t::tx_queue_len (C++ member), 801
twai_get_status_info (C++ function), 799
twai_get_status_info_v2 (C++ function), 799
twai_handle_t (C++ type), 802
twai_initiate_recovery (C++ function), 799
twai_initiate_recovery_v2 (C++ function), 799
TWAI_IO_UNUSED (C macro), 802
twai_message_t (C++ struct), 791
twai_message_t::data (C++ member), 792
twai_message_t::data_length_code (C++ member), 791
twai_message_t::dlc_non_comp (C++ member), 791
twai_message_t::extd (C++ member), 791
twai_message_t::flags (C++ member), 791
twai_message_t::identifier (C++ member), 791
twai_message_t::reserved (C++ member), 791
twai_message_t::rtr (C++ member), 791
twai_message_t::self (C++ member), 791
twai_message_t::ss (C++ member), 791
twai_mode_t (C++ enum), 793
twai_mode_t::TWAI_MODE_LISTEN_ONLY (C++ enumerator), 793
twai_mode_t::TWAI_MODE_NO_ACK (C++ enumerator), 793
twai_mode_t::TWAI_MODE_NORMAL (C++ enumerator), 793
twai_read_alerts (C++ function), 797
twai_read_alerts_v2 (C++ function), 798
twai_receive (C++ function), 797
twai_receive_v2 (C++ function), 797
twai_reconfigure_alerts (C++ function), 798
twai_reconfigure_alerts_v2 (C++ function), 798
twai_start (C++ function), 795
twai_start_v2 (C++ function), 795
twai_state_t (C++ enum), 802
twai_state_t::TWAI_STATE_BUS_OFF (C++ enumerator), 803
twai_state_t::TWAI_STATE_RECOVERING (C++ enumerator), 803
twai_state_t::TWAI_STATE_RUNNING (C++ enumerator), 803
twai_state_t::TWAI_STATE_STOPPED (C++ enumerator), 802
twai_status_info_t (C++ struct), 801
twai_status_info_t::arb_lost_count (C++ member), 802
twai_status_info_t::bus_error_count (C++ member), 802
twai_status_info_t::msgs_to_rx (C++ member), 802
twai_status_info_t::msgs_to_tx (C++ member), 802
twai_status_info_t::rx_error_counter (C++ member), 802
twai_status_info_t::rx_missed_count (C++ member), 802
twai_status_info_t::rx_overrun_count (C++ member), 802
twai_status_info_t::state (C++ member), 802
twai_status_info_t::tx_error_counter (C++ member), 802
twai_status_info_t::tx_failed_count (C++ member), 802
TWAI_STD_ID_MASK (C macro), 793
twai_stop (C++ function), 795
twai_stop_v2 (C++ function), 796
twai_timing_config_t (C++ struct), 792

twai_timing_config_t::brp (C++ member), 792
twai_timing_config_t::clk_src (C++ member), 792
twai_timing_config_t::quanta_resolution_hz (C++ member), 792
twai_timing_config_t::sjw (C++ member), 792
twai_timing_config_t::triple_sampling (C++ member), 792
twai_timing_config_t::tseg_1 (C++ member), 792
twai_timing_config_t::tseg_2 (C++ member), 792
twai_transmit (C++ function), 796
twai_transmit_v2 (C++ function), 796

U

uart_at_cmd_t (C++ struct), 823
uart_at_cmd_t::char_num (C++ member), 823
uart_at_cmd_t::cmd_char (C++ member), 823
uart_at_cmd_t::gap_tout (C++ member), 824
uart_at_cmd_t::post_idle (C++ member), 824
uart_at_cmd_t::pre_idle (C++ member), 824
UART_BITRATE_MAX (C macro), 822
uart_clear_intr_status (C++ function), 812
uart_config_t (C++ struct), 821
uart_config_t::backup_before_sleep (C++ member), 821
uart_config_t::baud_rate (C++ member), 821
uart_config_t::data_bits (C++ member), 821
uart_config_t::flags (C++ member), 821
uart_config_t::flow_ctrl (C++ member), 821
uart_config_t::lp_source_clk (C++ member), 821
uart_config_t::parity (C++ member), 821
uart_config_t::rx_flow_ctrl_thresh (C++ member), 821
uart_config_t::source_clk (C++ member), 821
uart_config_t::stop_bits (C++ member), 821
UART_CTS_GPIO13_DIRECT_CHANNEL (C macro), 829
UART_CTS_GPIO9_DIRECT_CHANNEL (C macro), 828
uart_disable_intr_mask (C++ function), 812
uart_disable_pattern_det_intr (C++ function), 817
uart_disable_rx_intr (C++ function), 813
uart_disable_tx_intr (C++ function), 813
uart_driver_delete (C++ function), 810
uart_driver_install (C++ function), 809
uart_enable_intr_mask (C++ function), 812
uart_enable_pattern_det_baud_intr (C++ function), 817
uart_enable_rx_intr (C++ function), 813
uart_enable_tx_intr (C++ function), 813
uart_event_t (C++ struct), 822
uart_event_t::size (C++ member), 822
uart_event_t::timeout_flag (C++ member), 822
uart_event_t::type (C++ member), 822
uart_event_type_t (C++ enum), 822
uart_event_type_t::UART_BREAK (C++ enumerator), 823
uart_event_type_t::UART_BUFFER_FULL (C++ enumerator), 823
uart_event_type_t::UART_DATA (C++ enumerator), 822
uart_event_type_t::UART_DATA_BREAK (C++ enumerator), 823
uart_event_type_t::UART_EVENT_MAX (C++ enumerator), 823
uart_event_type_t::UART_FIFO_OVF (C++ enumerator), 823
uart_event_type_t::UART_FRAME_ERR (C++ enumerator), 823
uart_event_type_t::UART_PARITY_ERR (C++ enumerator), 823
uart_event_type_t::UART_PATTERN_DET (C++ enumerator), 823
uart_event_type_t::UART_WAKEUP (C++ enumerator), 823
UART_FIFO_LEN (C macro), 822
uart_flush (C++ function), 816
uart_flush_input (C++ function), 816
uart_get_baudrate (C++ function), 811
uart_get_buffered_data_len (C++ function), 816
uart_get_collision_flag (C++ function), 819
uart_get_hw_flow_ctrl (C++ function), 812
uart_get_parity (C++ function), 811
uart_get_sclk_freq (C++ function), 811
uart_get_stop_bits (C++ function), 810
uart_get_tx_buffer_free_size (C++ function), 816
uart_get_wakeup_threshold (C++ function), 820
uart_get_word_length (C++ function), 810
UART_GPIO10_DIRECT_CHANNEL (C macro), 828
UART_GPIO11_DIRECT_CHANNEL (C macro), 828
UART_GPIO12_DIRECT_CHANNEL (C macro), 828
UART_GPIO13_DIRECT_CHANNEL (C macro), 828
UART_GPIO37_DIRECT_CHANNEL (C macro), 828
UART_GPIO38_DIRECT_CHANNEL (C macro), 828
UART_GPIO8_DIRECT_CHANNEL (C macro), 828
UART_GPIO9_DIRECT_CHANNEL (C macro), 828
UART_HW_FIFO_LEN (C macro), 822
uart_hw_flowcontrol_t (C++ enum), 826
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_CTS (C++ enumerator), 826

uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_CTS_RTS (C++ enumerator), 826
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_DISABLE (C++ enumerator), 826
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_MAX (C++ enumerator), 826
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_RTS (C++ enumerator), 826
uart_intr_config (C++ function), 814
uart_intr_config_t (C++ struct), 821
uart_intr_config_t::intr_enable_mask (C++ member), 821
uart_intr_config_t::rx_timeout_thresh (C++ member), 821
uart_intr_config_t::rxfifo_full_thresh (C++ member), 822
uart_intr_config_t::txfifo_empty_intr_thresh (C++ member), 822
uart_is_driver_installed (C++ function), 810
uart_isr_handle_t (C++ type), 822
uart_mode_t (C++ enum), 825
uart_mode_t::UART_MODE_IRDA (C++ enumerator), 825
uart_mode_t::UART_MODE_RS485_APP_CTRL (C++ enumerator), 825
uart_mode_t::UART_MODE_RS485_COLLISION_DETECT (C++ enumerator), 825
uart_mode_t::UART_MODE_RS485_HALF_DUPLEX (C++ enumerator), 825
uart_mode_t::UART_MODE_UART (C++ enumerator), 825
UART_NUM_0_CTS_DIRECT_GPIO_NUM (C macro), 828
UART_NUM_0_RTS_DIRECT_GPIO_NUM (C macro), 828
UART_NUM_0_RXD_DIRECT_GPIO_NUM (C macro), 828
UART_NUM_0_TXD_DIRECT_GPIO_NUM (C macro), 828
UART_NUM_1_CTS_DIRECT_GPIO_NUM (C macro), 828
UART_NUM_1_RTS_DIRECT_GPIO_NUM (C macro), 828
UART_NUM_1_RXD_DIRECT_GPIO_NUM (C macro), 828
UART_NUM_1_TXD_DIRECT_GPIO_NUM (C macro), 828
uart_param_config (C++ function), 814
uart_parity_t (C++ enum), 826
uart_parity_t::UART_PARITY_DISABLE (C++ enumerator), 826
uart_parity_t::UART_PARITY_EVEN (C++ enumerator), 826
uart_parity_t::UART_PARITY_ODD (C++ enumerator), 826
uart_pattern_get_pos (C++ function), 817
uart_pattern_pop_pos (C++ function), 817
uart_pattern_queue_reset (C++ function), 818
UART_PIN_NO_CHANGE (C macro), 822
uart_port_t (C++ enum), 824
uart_port_t::LP_UART_NUM_0 (C++ enumerator), 825
uart_port_t::UART_NUM_0 (C++ enumerator), 824
uart_port_t::UART_NUM_1 (C++ enumerator), 824
uart_port_t::UART_NUM_2 (C++ enumerator), 825
uart_port_t::UART_NUM_3 (C++ enumerator), 825
uart_port_t::UART_NUM_4 (C++ enumerator), 825
uart_port_t::UART_NUM_MAX (C++ enumerator), 825
uart_read_bytes (C++ function), 816
UART_RTS_GPIO12_DIRECT_CHANNEL (C macro), 829
UART_RTS_GPIO8_DIRECT_CHANNEL (C macro), 828
UART_RXD_GPIO11_DIRECT_CHANNEL (C macro), 829
UART_RXD_GPIO38_DIRECT_CHANNEL (C macro), 828
uart_sclk_t (C++ type), 824
uart_set_always_rx_timeout (C++ function), 820
uart_set_baudrate (C++ function), 811
uart_set_dtr (C++ function), 814
uart_set_hw_flow_ctrl (C++ function), 812
uart_set_line_inverse (C++ function), 811
uart_set_loop_back (C++ function), 820
uart_set_mode (C++ function), 818
uart_set_parity (C++ function), 811
uart_set_pin (C++ function), 813
uart_set_rts (C++ function), 814
uart_set_rx_full_threshold (C++ function), 818
uart_set_rx_timeout (C++ function), 819
uart_set_stop_bits (C++ function), 810
uart_set_sw_flow_ctrl (C++ function), 812
uart_set_tx_empty_threshold (C++ function), 818
uart_set_tx_idle_num (C++ function), 814
uart_set_wakeup_threshold (C++ function), 819
uart_set_word_length (C++ function), 810
uart_signal_inv_t (C++ enum), 827
uart_signal_inv_t::UART_SIGNAL_CTS_INV (C++ enumerator), 827
uart_signal_inv_t::UART_SIGNAL_DSR_INV (C++ enumerator), 827
uart_signal_inv_t::UART_SIGNAL_DTR_INV (C++ enumerator), 827
uart_signal_inv_t::UART_SIGNAL_INV_DISABLE

(C++ enumerator), 827
uart_signal_inv_t::UART_SIGNAL_IRDA_RX_INV (C++ enumerator), 827
uart_signal_inv_t::UART_SIGNAL_IRDA_TX_INV (C++ enumerator), 827
uart_signal_inv_t::UART_SIGNAL_RTS_INV (C++ enumerator), 827
uart_signal_inv_t::UART_SIGNAL_RXD_INV (C++ enumerator), 827
uart_signal_inv_t::UART_SIGNAL_TXD_INV (C++ enumerator), 827
uart_stop_bits_t (C++ enum), 826
uart_stop_bits_t::UART_STOP_BITS_1 (C++ enumerator), 826
uart_stop_bits_t::UART_STOP_BITS_1_5 (C++ enumerator), 826
uart_stop_bits_t::UART_STOP_BITS_2 (C++ enumerator), 826
uart_stop_bits_t::UART_STOP_BITS_MAX (C++ enumerator), 826
uart_sw_flowctrl_t (C++ struct), 824
uart_sw_flowctrl_t::xoff_char (C++ member), 824
uart_sw_flowctrl_t::xoff_thrd (C++ member), 824
uart_sw_flowctrl_t::xon_char (C++ member), 824
uart_sw_flowctrl_t::xon_thrd (C++ member), 824
uart_tx_chars (C++ function), 815
UART_TXD_GPIO10_DIRECT_CHANNEL (C macro), 828
UART_TXD_GPIO37_DIRECT_CHANNEL (C macro), 828
uart_vfs_dev_port_set_rx_line_endings (C++ function), 1339
uart_vfs_dev_port_set_tx_line_endings (C++ function), 1340
uart_vfs_dev_register (C++ function), 1339
uart_vfs_dev_use_driver (C++ function), 1340
uart_vfs_dev_use_nonblocking (C++ function), 1340
uart_wait_tx_done (C++ function), 815
uart_wait_tx_idle_polling (C++ function), 820
uart_word_length_t (C++ enum), 825
uart_word_length_t::UART_DATA_5_BITS (C++ enumerator), 825
uart_word_length_t::UART_DATA_6_BITS (C++ enumerator), 825
uart_word_length_t::UART_DATA_7_BITS (C++ enumerator), 825
uart_word_length_t::UART_DATA_8_BITS (C++ enumerator), 825
uart_word_length_t::UART_DATA_BITS_MAX (C++ enumerator), 826
uart_write_bytes (C++ function), 815
uart_write_bytes_with_break (C++ function), 815
ulp_lp_core_cfg_t (C++ struct), 1734
ulp_lp_core_cfg_t::lp_timer_sleep_duration_us (C++ member), 1734
ulp_lp_core_cfg_t::skip_lp_rom_boot (C++ member), 1734
ulp_lp_core_cfg_t::wakeup_source (C++ member), 1734
ulp_lp_core_delay_cycles (C++ function), 1743
ulp_lp_core_delay_us (C++ function), 1743
ulp_lp_core_get_wakeup_cause (C++ function), 1743
ulp_lp_core_gpio_clear_intr_status (C++ function), 1745
ulp_lp_core_gpio_get_level (C++ function), 1744
ulp_lp_core_gpio_init (C++ function), 1744
ulp_lp_core_gpio_input_disable (C++ function), 1744
ulp_lp_core_gpio_input_enable (C++ function), 1744
ulp_lp_core_gpio_intr_enable (C++ function), 1745
ulp_lp_core_gpio_output_disable (C++ function), 1744
ulp_lp_core_gpio_output_enable (C++ function), 1744
ulp_lp_core_gpio_pulldown_disable (C++ function), 1745
ulp_lp_core_gpio_pulldown_enable (C++ function), 1745
ulp_lp_core_gpio_pullup_disable (C++ function), 1745
ulp_lp_core_gpio_pullup_enable (C++ function), 1745
ulp_lp_core_gpio_set_level (C++ function), 1744
ulp_lp_core_gpio_set_output_mode (C++ function), 1745
ulp_lp_core_halt (C++ function), 1743
ulp_lp_core_intr_disable (C++ function), 1750
ulp_lp_core_intr_enable (C++ function), 1750
ulp_lp_core_load_binary (C++ function), 1734
ulp_lp_core_run (C++ function), 1733
ulp_lp_core_stop (C++ function), 1734
ulp_lp_core_stop_lp_core (C++ function), 1743
ulp_lp_core_sw_intr_clear (C++ function), 1744
ulp_lp_core_sw_intr_enable (C++ function), 1743
ulp_lp_core_sw_intr_trigger (C++ function), 1734

ulp_lp_core_update_wakeup_cause (C++ function), 1743
ulp_lp_core_wait_for_intr (C++ function), 1744
ulp_lp_core_wakeup_main_processor (C++ function), 1743
ULP_LP_CORE_WAKEUP_SOURCE_ETM (C macro), 1734
ULP_LP_CORE_WAKEUP_SOURCE_HP_CPU (C macro), 1734
ULP_LP_CORE_WAKEUP_SOURCE_LP_IO (C macro), 1734
ULP_LP_CORE_WAKEUP_SOURCE_LP_TIMER (C macro), 1734
ULP_LP_CORE_WAKEUP_SOURCE_LP_UART (C macro), 1734
ulTaskGenericNotifyValueClear (C++ function), 1457
ulTaskGetIdleRunTimeCounter (C++ function), 1453
ulTaskGetIdleRunTimePercent (C++ function), 1454
ulTaskNotifyTakeIndexed (C macro), 1464
ulTaskNotifyValueClear (C macro), 1465
ulTaskNotifyValueClearIndexed (C macro), 1465
USB_B_DESCRIPTOR_TYPE_BOS (C macro), 869
USB_B_DESCRIPTOR_TYPE_CONFIGURATION (C macro), 868
USB_B_DESCRIPTOR_TYPE_CS_RADIO_CONTROL (C macro), 869
USB_B_DESCRIPTOR_TYPE_DEBUG (C macro), 869
USB_B_DESCRIPTOR_TYPE_DEVICE (C macro), 868
USB_B_DESCRIPTOR_TYPE_DEVICE_CAPABILITY (C macro), 869
USB_B_DESCRIPTOR_TYPE_DEVICE_QUALIFIER (C macro), 868
USB_B_DESCRIPTOR_TYPE_ENCRYPTION_TYPE (C macro), 869
USB_B_DESCRIPTOR_TYPE_ENDPOINT (C macro), 868
USB_B_DESCRIPTOR_TYPE_INTERFACE (C macro), 868
USB_B_DESCRIPTOR_TYPE_INTERFACE_ASSOCIATION (C macro), 869
USB_B_DESCRIPTOR_TYPE_INTERFACE_POWER (C macro), 869
USB_B_DESCRIPTOR_TYPE_KEY (C macro), 869
USB_B_DESCRIPTOR_TYPE_OTG (C macro), 869
USB_B_DESCRIPTOR_TYPE_OTHER_SPEED_CONFIGURATION (C macro), 869
USB_B_DESCRIPTOR_TYPE_PIPE_USAGE (C macro), 869
USB_B_DESCRIPTOR_TYPE_RPIPE (C macro), 869
USB_B_DESCRIPTOR_TYPE_SECURITY (C macro), 869
USB_B_DESCRIPTOR_TYPE_STRING (C macro), 868
USB_B_DESCRIPTOR_TYPE_WIRE_ADAPTER (C macro), 869
USB_B_DESCRIPTOR_TYPE_WIRELESS_ENDPOINT_COMP (C macro), 869
USB_B_ENDPOINT_ADDRESS_EP_DIR_MASK (C macro), 873
USB_B_ENDPOINT_ADDRESS_EP_NUM_MASK (C macro), 873
USB_B_REQUEST_CLEAR_FEATURE (C macro), 870
USB_B_REQUEST_GET_CONFIGURATION (C macro), 870
USB_B_REQUEST_GET_DESCRIPTOR (C macro), 870
USB_B_REQUEST_GET_INTERFACE (C macro), 870
USB_B_REQUEST_GET_STATUS (C macro), 870
USB_B_REQUEST_SET_ADDRESS (C macro), 870
USB_B_REQUEST_SET_CONFIGURATION (C macro), 870
USB_B_REQUEST_SET_DESCRIPTOR (C macro), 870
USB_B_REQUEST_SET_FEATURE (C macro), 870
USB_B_REQUEST_SET_INTERFACE (C macro), 870
USB_B_REQUEST_SYNCH_FRAME (C macro), 870
USB_BM_ATTRIBUTES_BATTERY (C macro), 873
USB_BM_ATTRIBUTES_ONE (C macro), 872
USB_BM_ATTRIBUTES_SELFPOWER (C macro), 873
USB_BM_ATTRIBUTES_SYNC_ADAPTIVE (C macro), 874
USB_BM_ATTRIBUTES_SYNC_ASYNC (C macro), 873
USB_BM_ATTRIBUTES_SYNC_NONE (C macro), 873
USB_BM_ATTRIBUTES_SYNC_SYNC (C macro), 874
USB_BM_ATTRIBUTES_SYNCTYPE_MASK (C macro), 873
USB_BM_ATTRIBUTES_USAGE_DATA (C macro), 874
USB_BM_ATTRIBUTES_USAGE_FEEDBACK (C macro), 874
USB_BM_ATTRIBUTES_USAGE_IMPLICIT_FB (C macro), 874
USB_BM_ATTRIBUTES_USAGETYPE_MASK (C macro), 874
USB_BM_ATTRIBUTES_WAKEUP (C macro), 873
USB_BM_ATTRIBUTES_XFER_BULK (C macro), 873
USB_BM_ATTRIBUTES_XFER_CONTROL (C macro), 873
USB_BM_ATTRIBUTES_XFER_INT (C macro), 873
USB_BM_ATTRIBUTES_XFER_ISOC (C macro),

873
USB_BM_ATTRIBUTES_XFERTYPE_MASK (C macro), 873
USB_BM_REQUEST_TYPE_DIR_IN (C macro), 869
USB_BM_REQUEST_TYPE_DIR_OUT (C macro), 869
USB_BM_REQUEST_TYPE_RECIP_DEVICE (C macro), 870
USB_BM_REQUEST_TYPE_RECIP_ENDPOINT (C macro), 870
USB_BM_REQUEST_TYPE_RECIP_INTERFACE (C macro), 870
USB_BM_REQUEST_TYPE_RECIP_MASK (C macro), 870
USB_BM_REQUEST_TYPE_RECIP_OTHER (C macro), 870
USB_BM_REQUEST_TYPE_TYPE_CLASS (C macro), 869
USB_BM_REQUEST_TYPE_TYPE_MASK (C macro), 870
USB_BM_REQUEST_TYPE_TYPE_RESERVED (C macro), 870
USB_BM_REQUEST_TYPE_TYPE_STANDARD (C macro), 869
USB_BM_REQUEST_TYPE_TYPE_VENDOR (C macro), 869
USB_CLASS_APP_SPEC (C macro), 872
USB_CLASS_AUDIO (C macro), 871
USB_CLASS_AUDIO_VIDEO (C macro), 872
USB_CLASS_BILLBOARD (C macro), 872
USB_CLASS_CDC_DATA (C macro), 872
USB_CLASS_COMM (C macro), 871
USB_CLASS_CONTENT_SEC (C macro), 872
USB_CLASS_CSCID (C macro), 872
USB_CLASS_HID (C macro), 871
USB_CLASS_HUB (C macro), 872
USB_CLASS_MASS_STORAGE (C macro), 872
USB_CLASS_MISC (C macro), 872
USB_CLASS_PER_INTERFACE (C macro), 871
USB_CLASS_PERSONAL_HEALTHCARE (C macro), 872
USB_CLASS_PHYSICAL (C macro), 872
USB_CLASS_PRINTER (C macro), 872
USB_CLASS_STILL_IMAGE (C macro), 872
USB_CLASS_USB_TYPE_C_BRIDGE (C macro), 872
USB_CLASS_VENDOR_SPEC (C macro), 872
USB_CLASS_VIDEO (C macro), 872
USB_CLASS_WIRELESS_CONTROLLER (C macro), 872
USB_CONFIG_DESC_SIZE (C macro), 872
usb_config_desc_t (C++ union), 865
usb_config_desc_t::bConfigurationValue (C++ member), 865
usb_config_desc_t::bDescriptorType (C++ member), 865
usb_config_desc_t::bLength (C++ member), 865
usb_config_desc_t::bmAttributes (C++ member), 865
usb_config_desc_t::bMaxPower (C++ member), 865
usb_config_desc_t::bNumInterfaces (C++ member), 865
usb_config_desc_t::iConfiguration (C++ member), 865
usb_config_desc_t::USB_DESC_ATTR (C++ member), 865
usb_config_desc_t::val (C++ member), 865
usb_config_desc_t::wTotalLength (C++ member), 865
USB_DESC_ATTR (C macro), 868
USB_DEVICE_DESC_SIZE (C macro), 871
usb_device_desc_t (C++ union), 864
usb_device_desc_t::bcdDevice (C++ member), 864
usb_device_desc_t::bcdUSB (C++ member), 864
usb_device_desc_t::bDescriptorType (C++ member), 864
usb_device_desc_t::bDeviceClass (C++ member), 864
usb_device_desc_t::bDeviceProtocol (C++ member), 864
usb_device_desc_t::bDeviceSubClass (C++ member), 864
usb_device_desc_t::bLength (C++ member), 864
usb_device_desc_t::bMaxPacketSize0 (C++ member), 864
usb_device_desc_t::bNumConfigurations (C++ member), 865
usb_device_desc_t::idProduct (C++ member), 864
usb_device_desc_t::idVendor (C++ member), 864
usb_device_desc_t::iManufacturer (C++ member), 864
usb_device_desc_t::iProduct (C++ member), 864
usb_device_desc_t::iSerialNumber (C++ member), 864
usb_device_desc_t::USB_DESC_ATTR (C++ member), 865
usb_device_desc_t::val (C++ member), 865
usb_device_handle_t (C++ type), 859
usb_device_info_t (C++ struct), 857
usb_device_info_t::bConfigurationValue (C++ member), 857
usb_device_info_t::bMaxPacketSize0 (C++ member), 857
usb_device_info_t::dev_addr (C++ member), 857
usb_device_info_t::parent (C++ member), 857
usb_device_info_t::speed (C++ member),

857
usb_device_info_t::str_desc_manufacturer (C++ member), 857
usb_device_info_t::str_desc_product (C++ member), 857
usb_device_info_t::str_desc_serial_num (C++ member), 857
usb_device_state_t (C++ enum), 874
usb_device_state_t::USB_DEVICE_STATE_ADDRESS (C++ enumerator), 875
usb_device_state_t::USB_DEVICE_STATE_ATTACHED (C++ enumerator), 874
usb_device_state_t::USB_DEVICE_STATE_CONFIGURED (C++ enumerator), 875
usb_device_state_t::USB_DEVICE_STATE_DEFAULT (C++ enumerator), 875
usb_device_state_t::USB_DEVICE_STATE_NOT_ATTACHED (C++ enumerator), 874
usb_device_state_t::USB_DEVICE_STATE_POWERED (C++ enumerator), 874
usb_device_state_t::USB_DEVICE_STATE_SUSPENDED (C++ enumerator), 875
usb_device_status_t (C++ union), 863
usb_device_status_t::remote_wakeup (C++ member), 863
usb_device_status_t::reserved (C++ member), 863
usb_device_status_t::self_powered (C++ member), 863
usb_device_status_t::USB_DESC_ATTR (C++ member), 863
usb_device_status_t::val (C++ member), 863
USB_EP_DESC_GET_EP_DIR (C macro), 874
USB_EP_DESC_GET_EP_NUM (C macro), 874
USB_EP_DESC_GET_MPS (C macro), 874
USB_EP_DESC_GET_MULT (C macro), 874
USB_EP_DESC_GET_SYNCTYPE (C macro), 874
USB_EP_DESC_GET_USAGETYPE (C macro), 874
USB_EP_DESC_GET_XFERTYPE (C macro), 874
USB_EP_DESC_SIZE (C macro), 873
usb_ep_desc_t (C++ union), 867
usb_ep_desc_t::bDescriptorType (C++ member), 867
usb_ep_desc_t::bEndpointAddress (C++ member), 867
usb_ep_desc_t::bInterval (C++ member), 868
usb_ep_desc_t::bLength (C++ member), 867
usb_ep_desc_t::bmAttributes (C++ member), 867
usb_ep_desc_t::USB_DESC_ATTR (C++ member), 868
usb_ep_desc_t::val (C++ member), 868
usb_ep_desc_t::wMaxPacketSize (C++ member), 867
usb_host_client_config_t (C++ struct), 852
usb_host_client_config_t::async (C++ member), 853
usb_host_client_config_t::callback_arg (C++ member), 853
usb_host_client_config_t::client_event_callback (C++ member), 853
usb_host_client_config_t::is_synchronous (C++ member), 853
usb_host_client_config_t::max_num_event_msg (C++ member), 853
usb_host_client_deregister (C++ function), 844
usb_host_client_event_cb_t (C++ type), 853
usb_host_client_event_msg_t (C++ struct), 851
usb_host_client_event_msg_t::address (C++ member), 851
usb_host_client_event_msg_t::dev_gone (C++ member), 852
usb_host_client_event_msg_t::dev_hdl (C++ member), 852
usb_host_client_event_msg_t::event (C++ member), 851
usb_host_client_event_msg_t::new_dev (C++ member), 851
usb_host_client_event_t (C++ enum), 853
usb_host_client_event_t::USB_HOST_CLIENT_EVENT_DE (C++ enumerator), 853
usb_host_client_event_t::USB_HOST_CLIENT_EVENT_NE (C++ enumerator), 853
usb_host_client_handle_events (C++ function), 844
usb_host_client_handle_t (C++ type), 853
usb_host_client_register (C++ function), 844
usb_host_client_unblock (C++ function), 845
usb_host_config_t (C++ struct), 852
usb_host_config_t::enum_filter_cb (C++ member), 852
usb_host_config_t::intr_flags (C++ member), 852
usb_host_config_t::root_port_unpowered (C++ member), 852
usb_host_config_t::skip_phy_setup (C++ member), 852
usb_host_device_addr_list_fill (C++ function), 846
usb_host_device_close (C++ function), 845
usb_host_device_free_all (C++ function), 845
usb_host_device_info (C++ function), 846
usb_host_device_open (C++ function), 845
usb_host_endpoint_clear (C++ function), 849
usb_host_endpoint_flush (C++ function), 849
usb_host_endpoint_halt (C++ function), 849
usb_host_enum_filter_cb_t (C++ type), 859
usb_host_free_config_desc (C++ function), 848
usb_host_get_active_config_descriptor

(C++ function), 847
usb_host_get_config_desc (C++ function), 847
usb_host_get_device_descriptor (C++ function), 846
usb_host_install (C++ function), 842
usb_host_interface_claim (C++ function), 848
usb_host_interface_release (C++ function), 848
USB_HOST_LIB_EVENT_FLAGS_ALL_FREE (C macro), 853
USB_HOST_LIB_EVENT_FLAGS_NO_CLIENTS (C macro), 853
usb_host_lib_handle_events (C++ function), 843
usb_host_lib_info (C++ function), 843
usb_host_lib_info_t (C++ struct), 852
usb_host_lib_info_t::num_clients (C++ member), 852
usb_host_lib_info_t::num_devices (C++ member), 852
usb_host_lib_set_root_port_power (C++ function), 843
usb_host_lib_unblock (C++ function), 843
usb_host_transfer_alloc (C++ function), 850
usb_host_transfer_free (C++ function), 850
usb_host_transfer_submit (C++ function), 850
usb_host_transfer_submit_control (C++ function), 851
usb_host_uninstall (C++ function), 842
USB_IAD_DESC_SIZE (C macro), 873
usb_iad_desc_t (C++ union), 866
usb_iad_desc_t::bDescriptorType (C++ member), 866
usb_iad_desc_t::bFirstInterface (C++ member), 866
usb_iad_desc_t::bFunctionClass (C++ member), 866
usb_iad_desc_t::bFunctionProtocol (C++ member), 866
usb_iad_desc_t::bFunctionSubClass (C++ member), 866
usb_iad_desc_t::bInterfaceCount (C++ member), 866
usb_iad_desc_t::bLength (C++ member), 866
usb_iad_desc_t::iFunction (C++ member), 866
usb_iad_desc_t::USB_DESC_ATTR (C++ member), 866
usb_iad_desc_t::val (C++ member), 866
USB_INTF_DESC_SIZE (C macro), 873
usb_intf_desc_t (C++ union), 866
usb_intf_desc_t::bAlternateSetting (C++ member), 867
usb_intf_desc_t::bDescriptorType (C++ member), 866
usb_intf_desc_t::bInterfaceClass (C++ member), 867
usb_intf_desc_t::bInterfaceNumber (C++ member), 867
usb_intf_desc_t::bInterfaceProtocol (C++ member), 867
usb_intf_desc_t::bInterfaceSubClass (C++ member), 867
usb_intf_desc_t::bLength (C++ member), 866
usb_intf_desc_t::bNumEndpoints (C++ member), 867
usb_intf_desc_t::iInterface (C++ member), 867
usb_intf_desc_t::USB_DESC_ATTR (C++ member), 867
usb_intf_desc_t::val (C++ member), 867
usb_isoc_packet_desc_t (C++ struct), 857
usb_isoc_packet_desc_t::actual_num_bytes (C++ member), 858
usb_isoc_packet_desc_t::num_bytes (C++ member), 858
usb_isoc_packet_desc_t::status (C++ member), 858
usb_parent_dev_info_t (C++ struct), 857
usb_parent_dev_info_t::dev_hdl (C++ member), 857
usb_parent_dev_info_t::port_num (C++ member), 857
usb_parse_endpoint_descriptor_by_address (C++ function), 855
usb_parse_endpoint_descriptor_by_index (C++ function), 855
usb_parse_interface_descriptor (C++ function), 855
usb_parse_interface_number_of_alternate (C++ function), 854
usb_parse_next_descriptor (C++ function), 854
usb_parse_next_descriptor_of_type (C++ function), 854
usb_print_config_descriptor (C++ function), 856
usb_print_device_descriptor (C++ function), 856
usb_print_string_descriptor (C++ function), 856
usb_round_up_to_mps (C++ function), 856
USB_SETUP_PACKET_INIT_GET_CONFIG (C macro), 871
USB_SETUP_PACKET_INIT_GET_CONFIG_DESC (C macro), 871
USB_SETUP_PACKET_INIT_GET_DEVICE_DESC (C macro), 871
USB_SETUP_PACKET_INIT_GET_STATUS (C macro), 871
USB_SETUP_PACKET_INIT_GET_STR_DESC (C macro), 871

USB_SETUP_PACKET_INIT_SET_ADDR (*C macro*), 871
USB_SETUP_PACKET_INIT_SET_CONFIG (*C macro*), 871
USB_SETUP_PACKET_INIT_SET_INTERFACE (*C macro*), 871
USB_SETUP_PACKET_SIZE (*C macro*), 869
usb_setup_packet_t (*C++ union*), 862
usb_setup_packet_t::bmRequestType (*C++ member*), 862
usb_setup_packet_t::bRequest (*C++ member*), 862
usb_setup_packet_t::USB_DESC_ATTR (*C++ member*), 863
usb_setup_packet_t::val (*C++ member*), 863
usb_setup_packet_t::wIndex (*C++ member*), 862
usb_setup_packet_t::wLength (*C++ member*), 863
usb_setup_packet_t::wValue (*C++ member*), 862
usb_speed_t (*C++ enum*), 861
usb_speed_t::USB_SPEED_FULL (*C++ enumerator*), 861
usb_speed_t::USB_SPEED_HIGH (*C++ enumerator*), 861
usb_speed_t::USB_SPEED_LOW (*C++ enumerator*), 861
USB_STANDARD_DESC_SIZE (*C macro*), 871
usb_standard_desc_t (*C++ union*), 863
usb_standard_desc_t::bDescriptorType (*C++ member*), 863
usb_standard_desc_t::bLength (*C++ member*), 863
usb_standard_desc_t::USB_DESC_ATTR (*C++ member*), 863
usb_standard_desc_t::val (*C++ member*), 864
USB_STR_DESC_SIZE (*C macro*), 874
usb_str_desc_t (*C++ union*), 868
usb_str_desc_t::bDescriptorType (*C++ member*), 868
usb_str_desc_t::bLength (*C++ member*), 868
usb_str_desc_t::USB_DESC_ATTR (*C++ member*), 868
usb_str_desc_t::val (*C++ member*), 868
usb_str_desc_t::wData (*C++ member*), 868
USB_SUBCLASS_VENDOR_SPEC (*C macro*), 872
usb_transfer_cb_t (*C++ type*), 861
USB_TRANSFER_FLAG_ZERO_PACK (*C macro*), 859
usb_transfer_s (*C++ struct*), 858
usb_transfer_s::actual_num_bytes (*C++ member*), 858
usb_transfer_s::bEndpointAddress (*C++ member*), 858
usb_transfer_s::callback (*C++ member*), 859
usb_transfer_s::context (*C++ member*), 859
usb_transfer_s::data_buffer (*C++ member*), 858
usb_transfer_s::data_buffer_size (*C++ member*), 858
usb_transfer_s::device_handle (*C++ member*), 858
usb_transfer_s::flags (*C++ member*), 858
usb_transfer_s::isoc_packet_desc (*C++ member*), 859
usb_transfer_s::num_bytes (*C++ member*), 858
usb_transfer_s::num_isoc_packets (*C++ member*), 859
usb_transfer_s::status (*C++ member*), 858
usb_transfer_s::timeout_ms (*C++ member*), 858
usb_transfer_status_t (*C++ enum*), 861
usb_transfer_status_t::USB_TRANSFER_STATUS_CANCEL (*C++ enumerator*), 862
usb_transfer_status_t::USB_TRANSFER_STATUS_COMPLE (*C++ enumerator*), 861
usb_transfer_status_t::USB_TRANSFER_STATUS_ERROR (*C++ enumerator*), 861
usb_transfer_status_t::USB_TRANSFER_STATUS_NO_DEV (*C++ enumerator*), 862
usb_transfer_status_t::USB_TRANSFER_STATUS_OVERFL (*C++ enumerator*), 862
usb_transfer_status_t::USB_TRANSFER_STATUS_SKIPPE (*C++ enumerator*), 862
usb_transfer_status_t::USB_TRANSFER_STATUS_STALL (*C++ enumerator*), 862
usb_transfer_status_t::USB_TRANSFER_STATUS_TIMED_ (*C++ enumerator*), 861
usb_transfer_t (*C++ type*), 860
usb_transfer_type_t (*C++ enum*), 861
usb_transfer_type_t::USB_TRANSFER_TYPE_BULK (*C++ enumerator*), 861
usb_transfer_type_t::USB_TRANSFER_TYPE_CTRL (*C++ enumerator*), 861
usb_transfer_type_t::USB_TRANSFER_TYPE_INTR (*C++ enumerator*), 861
usb_transfer_type_t::USB_TRANSFER_TYPE_ISOCHRONOU (*C++ enumerator*), 861
USB_W_MAX_PACKET_SIZE_MPS_MASK (*C macro*), 873
USB_W_MAX_PACKET_SIZE_MULT_MASK (*C macro*), 873
USB_W_VALUE_DT_CONFIG (*C macro*), 870
USB_W_VALUE_DT_DEVICE (*C macro*), 870
USB_W_VALUE_DT_DEVICE_QUALIFIER (*C macro*), 871
USB_W_VALUE_DT_ENDPOINT (*C macro*), 871
USB_W_VALUE_DT_INTERFACE (*C macro*), 870
USB_W_VALUE_DT_INTERFACE_POWER (*C macro*), 871
USB_W_VALUE_DT_OTHER_SPEED_CONFIG (*C macro*), 871

USB_W_VALUE_DT_STRING (*C macro*), 870
uxQueueMessagesWaiting (*C++ function*), 1471
uxQueueMessagesWaitingFromISR (*C++ function*), 1473
uxQueueSpacesAvailable (*C++ function*), 1471
uxSemaphoreGetCount (*C macro*), 1500
uxSemaphoreGetCountFromISR (*C macro*), 1500
uxTaskGetNumberOfTasks (*C++ function*), 1449
uxTaskGetStackHighWaterMark (*C++ function*), 1449
uxTaskGetStackHighWaterMark2 (*C++ function*), 1450
uxTaskGetSystemState (*C++ function*), 1451
uxTaskPriorityGet (*C++ function*), 1443
uxTaskPriorityGetFromISR (*C++ function*), 1444
uxTimerGetReloadMode (*C++ function*), 1508

V

vApplicationGetIdleTaskMemory (*C++ function*), 1450
vApplicationGetTimerTaskMemory (*C++ function*), 1509
vEventGroupDelete (*C++ function*), 1525
vEventGroupDeleteWithCaps (*C++ function*), 1572
VFS_FAT_MOUNT_DEFAULT_CONFIG (*C macro*), 1264
vMessageBufferDelete (*C macro*), 1543
vMessageBufferDeleteWithCaps (*C++ function*), 1571
vprintf_like_t (*C++ type*), 1636
vQueueAddToRegistry (*C++ function*), 1474
vQueueDelete (*C++ function*), 1471
vQueueDeleteWithCaps (*C++ function*), 1569
vQueueUnregisterQueue (*C++ function*), 1474
vRingbufferDelete (*C++ function*), 1561
vRingbufferDeleteWithCaps (*C++ function*), 1563
vRingbufferGetInfo (*C++ function*), 1562
vRingbufferReturnItem (*C++ function*), 1560
vRingbufferReturnItemFromISR (*C++ function*), 1561
vSemaphoreCreateBinary (*C macro*), 1487
vSemaphoreDelete (*C macro*), 1500
vSemaphoreDeleteWithCaps (*C++ function*), 1570
vStreamBufferDelete (*C++ function*), 1532
vStreamBufferDeleteWithCaps (*C++ function*), 1571
vTaskAllocateMPURegions (*C++ function*), 1441
vTaskDelay (*C++ function*), 1442
vTaskDelete (*C++ function*), 1441
vTaskDeleteWithCaps (*C++ function*), 1569
vTaskGenericNotifyGiveFromISR (*C++ function*), 1455
vTaskGetInfo (*C++ function*), 1444
vTaskGetRunTimeStats (*C++ function*), 1453
vTaskList (*C++ function*), 1452
vTaskNotifyGiveFromISR (*C macro*), 1464
vTaskNotifyGiveIndexedFromISR (*C macro*), 1464
vTaskPrioritySet (*C++ function*), 1445
vTaskResume (*C++ function*), 1446
vTaskSetApplicationTaskTag (*C++ function*), 1450
vTaskSetThreadLocalStoragePointer (*C++ function*), 1450
vTaskSetThreadLocalStoragePointerAndDelCallback (*C++ function*), 1568
vTaskSetTimeOutState (*C++ function*), 1457
vTaskSuspend (*C++ function*), 1446
vTaskSuspendAll (*C++ function*), 1447
vTimerSetReloadMode (*C++ function*), 1508
vTimerSetTimerID (*C++ function*), 1505

W

walker_block_info (*C++ struct*), 1580
walker_block_info::ptr (*C++ member*), 1580
walker_block_info::size (*C++ member*), 1580
walker_block_info::used (*C++ member*), 1580
walker_block_info_t (*C++ type*), 1582
walker_heap_info (*C++ struct*), 1580
walker_heap_info::end (*C++ member*), 1580
walker_heap_info::start (*C++ member*), 1580
walker_heap_into_t (*C++ type*), 1582
wl_erase_range (*C++ function*), 1343
wl_handle_t (*C++ type*), 1344
WL_INVALID_HANDLE (*C macro*), 1344
wl_mount (*C++ function*), 1343
wl_read (*C++ function*), 1344
wl_sector_size (*C++ function*), 1344
wl_size (*C++ function*), 1344
wl_unmount (*C++ function*), 1343
wl_write (*C++ function*), 1343

X

xEventGroupClearBits (*C++ function*), 1521
xEventGroupClearBitsFromISR (*C macro*), 1525
xEventGroupCreate (*C++ function*), 1519
xEventGroupCreateStatic (*C++ function*), 1519
xEventGroupCreateWithCaps (*C++ function*), 1571
xEventGroupGetBits (*C macro*), 1527
xEventGroupGetBitsFromISR (*C++ function*), 1525
xEventGroupGetStaticBuffer (*C++ function*), 1525
xEventGroupSetBits (*C++ function*), 1522
xEventGroupSetBitsFromISR (*C macro*), 1526

xEventGroupSync (*C++ function*), 1523
xEventGroupWaitBits (*C++ function*), 1520
xMessageBufferCreateStaticWithCallback (*C macro*), 1537
xMessageBufferCreateWithCallback (*C macro*), 1537
xMessageBufferCreateWithCaps (*C++ function*), 1571
xMessageBufferGetStaticBuffers (*C macro*), 1538
xMessageBufferIsEmpty (*C macro*), 1544
xMessageBufferIsFull (*C macro*), 1544
xMessageBufferNextLengthBytes (*C macro*), 1544
xMessageBufferReceive (*C macro*), 1541
xMessageBufferReceiveCompletedFromISR (*C macro*), 1545
xMessageBufferReceiveFromISR (*C macro*), 1542
xMessageBufferReset (*C macro*), 1544
xMessageBufferSend (*C macro*), 1539
xMessageBufferSendCompletedFromISR (*C macro*), 1544
xMessageBufferSendFromISR (*C macro*), 1540
xMessageBufferSpaceAvailable (*C macro*), 1544
xMessageBufferSpacesAvailable (*C macro*), 1544
xQueueAddToSet (*C++ function*), 1475
xQueueCreate (*C macro*), 1476
xQueueCreateSet (*C++ function*), 1474
xQueueCreateStatic (*C macro*), 1477
xQueueCreateWithCaps (*C++ function*), 1569
xQueueGenericSend (*C++ function*), 1467
xQueueGenericSendFromISR (*C++ function*), 1471
xQueueGetStaticBuffers (*C macro*), 1478
xQueueGiveFromISR (*C++ function*), 1472
xQueueIsQueueEmptyFromISR (*C++ function*), 1473
xQueueIsQueueFullFromISR (*C++ function*), 1473
xQueueOverwrite (*C macro*), 1481
xQueueOverwriteFromISR (*C macro*), 1484
xQueuePeek (*C++ function*), 1468
xQueuePeekFromISR (*C++ function*), 1469
xQueueReceive (*C++ function*), 1470
xQueueReceiveFromISR (*C++ function*), 1472
xQueueRemoveFromSet (*C++ function*), 1475
xQueueReset (*C macro*), 1486
xQueueSelectFromSet (*C++ function*), 1475
xQueueSelectFromSetFromISR (*C++ function*), 1476
xQueueSend (*C macro*), 1480
xQueueSendFromISR (*C macro*), 1485
xQueueSendToBack (*C macro*), 1479
xQueueSendToBackFromISR (*C macro*), 1483
xQueueSendToFront (*C macro*), 1478
xQueueSendToFrontFromISR (*C macro*), 1482
xRingbufferAddToQueueSetRead (*C++ function*), 1562
xRingbufferCanRead (*C++ function*), 1562
xRingbufferCreate (*C++ function*), 1555
xRingbufferCreateNoSplit (*C++ function*), 1555
xRingbufferCreateStatic (*C++ function*), 1555
xRingbufferCreateWithCaps (*C++ function*), 1563
xRingbufferGetCurFreeSize (*C++ function*), 1561
xRingbufferGetMaxItemSize (*C++ function*), 1561
xRingbufferGetStaticBuffer (*C++ function*), 1563
xRingbufferPrintInfo (*C++ function*), 1563
xRingbufferReceive (*C++ function*), 1557
xRingbufferReceiveFromISR (*C++ function*), 1558
xRingbufferReceiveSplit (*C++ function*), 1558
xRingbufferReceiveSplitFromISR (*C++ function*), 1559
xRingbufferReceiveUpTo (*C++ function*), 1559
xRingbufferReceiveUpToFromISR (*C++ function*), 1560
xRingbufferRemoveFromQueueSetRead (*C++ function*), 1562
xRingbufferSend (*C++ function*), 1556
xRingbufferSendAcquire (*C++ function*), 1557
xRingbufferSendComplete (*C++ function*), 1557
xRingbufferSendFromISR (*C++ function*), 1556
xSemaphoreCreateBinary (*C macro*), 1487
xSemaphoreCreateBinaryStatic (*C macro*), 1488
xSemaphoreCreateBinaryWithCaps (*C++ function*), 1569
xSemaphoreCreateCounting (*C macro*), 1498
xSemaphoreCreateCountingStatic (*C macro*), 1499
xSemaphoreCreateCountingWithCaps (*C++ function*), 1570
xSemaphoreCreateMutex (*C macro*), 1495
xSemaphoreCreateMutexStatic (*C macro*), 1495
xSemaphoreCreateMutexWithCaps (*C++ function*), 1570
xSemaphoreCreateRecursiveMutex (*C macro*), 1496
xSemaphoreCreateRecursiveMutexStatic (*C macro*), 1497
xSemaphoreCreateRecursiveMutexWithCaps (*C++ function*), 1570
xSemaphoreGetMutexHolder (*C macro*), 1500
xSemaphoreGetMutexHolderFromISR (*C*

macro), 1500
xSemaphoreGetStaticBuffer (*C macro*), 1500
xSemaphoreGive (*C macro*), 1491
xSemaphoreGiveFromISR (*C macro*), 1493
xSemaphoreGiveRecursive (*C macro*), 1492
xSemaphoreTake (*C macro*), 1489
xSemaphoreTakeFromISR (*C macro*), 1494
xSemaphoreTakeRecursive (*C macro*), 1490
xSTATIC_RINGBUFFER (*C++ struct*), 1563
xStreamBufferBytesAvailable (*C++ function*), 1533
xStreamBufferCreateStaticWithCallback (*C macro*), 1535
xStreamBufferCreateWithCallback (*C macro*), 1534
xStreamBufferCreateWithCaps (*C++ function*), 1571
xStreamBufferGetStaticBuffers (*C++ function*), 1527
xStreamBufferIsEmpty (*C++ function*), 1532
xStreamBufferIsFull (*C++ function*), 1532
xStreamBufferReceive (*C++ function*), 1530
xStreamBufferReceiveCompletedFromISR (*C++ function*), 1534
xStreamBufferReceiveFromISR (*C++ function*), 1531
xStreamBufferReset (*C++ function*), 1532
xStreamBufferSend (*C++ function*), 1528
xStreamBufferSendCompletedFromISR (*C++ function*), 1533
xStreamBufferSendFromISR (*C++ function*), 1529
xStreamBufferSetTriggerLevel (*C++ function*), 1533
xStreamBufferSpacesAvailable (*C++ function*), 1533
xTASK_STATUS (*C++ struct*), 1459
xTASK_STATUS::eCurrentState (*C++ member*), 1459
xTASK_STATUS::pcTaskName (*C++ member*), 1459
xTASK_STATUS::pxStackBase (*C++ member*), 1459
xTASK_STATUS::ulRunTimeCounter (*C++ member*), 1459
xTASK_STATUS::usStackHighWaterMark (*C++ member*), 1459
xTASK_STATUS::uxBasePriority (*C++ member*), 1459
xTASK_STATUS::uxCurrentPriority (*C++ member*), 1459
xTASK_STATUS::xCoreID (*C++ member*), 1460
xTASK_STATUS::xHandle (*C++ member*), 1459
xTASK_STATUS::xTaskNumber (*C++ member*), 1459
xTaskAbortDelay (*C++ function*), 1443
xTaskCallApplicationTaskHook (*C++ function*), 1451
xTaskCatchUpTicks (*C++ function*), 1458
xTaskCheckForTimeOut (*C++ function*), 1457
xTaskCreate (*C++ function*), 1438
xTaskCreatePinnedToCore (*C++ function*), 1566
xTaskCreatePinnedToCoreWithCaps (*C++ function*), 1568
xTaskCreateStatic (*C++ function*), 1439
xTaskCreateStaticPinnedToCore (*C++ function*), 1566
xTaskCreateWithCaps (*C++ function*), 1568
xTaskDelayUntil (*C++ function*), 1442
xTaskGenericNotifyStateClear (*C++ function*), 1456
xTaskGenericNotifyWait (*C++ function*), 1454
xTaskGetApplicationTaskTag (*C++ function*), 1450
xTaskGetApplicationTaskTagFromISR (*C++ function*), 1450
xTaskGetCoreID (*C++ function*), 1567
xTaskGetCurrentTaskHandleForCore (*C++ function*), 1567
xTaskGetHandle (*C++ function*), 1449
xTaskGetIdleTaskHandle (*C++ function*), 1451
xTaskGetStaticBuffers (*C++ function*), 1449
xTaskGetTickCount (*C++ function*), 1449
xTaskGetTickCountFromISR (*C++ function*), 1449
xTaskNotifyAndQueryIndexed (*C macro*), 1462
xTaskNotifyAndQueryIndexedFromISR (*C macro*), 1463
xTaskNotifyGiveIndexed (*C macro*), 1463
xTaskNotifyIndexed (*C macro*), 1460
xTaskNotifyIndexedFromISR (*C macro*), 1462
xTaskNotifyStateClear (*C macro*), 1465
xTaskNotifyStateClearIndexed (*C macro*), 1465
xTaskNotifyWait (*C macro*), 1463
xTaskNotifyWaitIndexed (*C macro*), 1463
xTaskResumeAll (*C++ function*), 1448
xTaskResumeFromISR (*C++ function*), 1447
xTimerChangePeriod (*C macro*), 1510
xTimerChangePeriodFromISR (*C macro*), 1516
xTimerCreate (*C++ function*), 1501
xTimerCreateStatic (*C++ function*), 1503
xTimerDelete (*C macro*), 1511
xTimerGetExpiryTime (*C++ function*), 1508
xTimerGetPeriod (*C++ function*), 1508
xTimerGetReloadMode (*C++ function*), 1508
xTimerGetStaticBuffer (*C++ function*), 1509
xTimerGetTimerDaemonTaskHandle (*C++ function*), 1506
xTimerIsTimerActive (*C++ function*), 1506
xTimerPendFunctionCall (*C++ function*), 1507
xTimerPendFunctionCallFromISR (*C++ function*), 1506
xTimerReset (*C macro*), 1512
xTimerResetFromISR (*C macro*), 1517

xTimerStart (*C macro*), 1509
xTimerStartFromISR (*C macro*), 1514
xTimerStop (*C macro*), 1510
xTimerStopFromISR (*C macro*), 1515