ESP32-C2

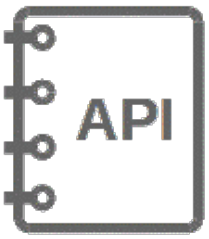
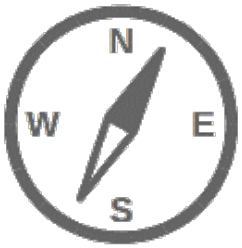
ESP-IDF Programming Guide

Release v5.1
乐鑫信息科技
2023 年 06 月 28 日

Table of contents

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

本文档仅包含针对 ESP32-C2 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-C2 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-C2 开发板等步骤。

备注： 这是 ESP-IDF 稳定版本 v5.1 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-C2 SoC 芯片支持以下功能：

- 2.4 GHz Wi-Fi
- 低功耗蓝牙
- 高性能 32 位 RISC-V 单核处理器
- 多种外设
- 适用于较简单、大批量生产的物联网应用

ESP32-C2 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-C2 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-C2** 开发板
- **USB 数据线** (A 转 Micro-B)
- 电脑（Windows、Linux 或 macOS）

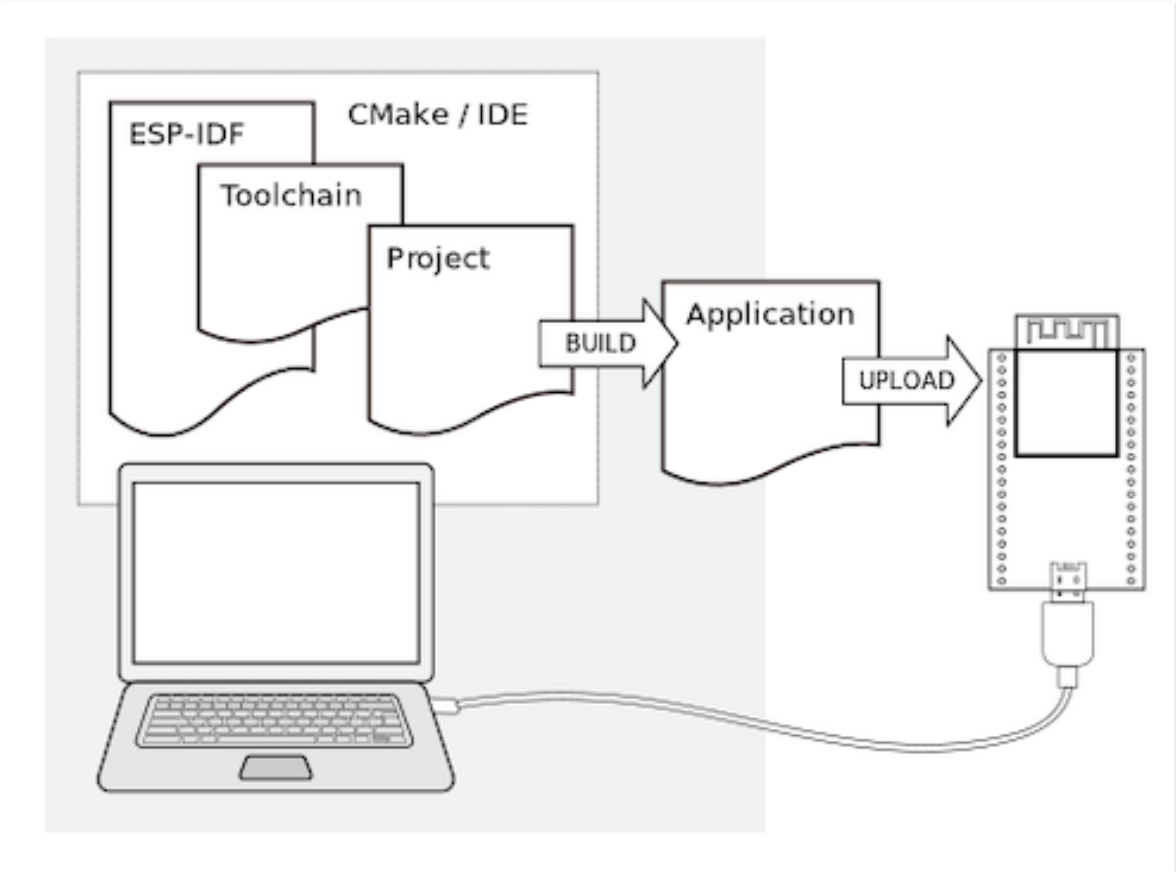
备注： 目前一些开发板使用的是 USB Type C 接口。请确保使用合适的数据线来连接开发板!

以下是 ESP32-C2 官方开发板，点击链接可了解更多硬件信息。

1.2.2 软件：

如需在 **ESP32-C2** 上使用 ESP-IDF，请安装以下软件：

- 设置 **工具链**，用于编译 ESP32-C2 代码；
- **编译构建工具**——CMake 和 Ninja 编译构建工具，用于编译 ESP32-C2 **应用程序**；
- 获取 **ESP-IDF** 软件开发框架。该框架已经基本包含 ESP32-C2 使用的 API（软件库和源代码）和运行 **工具链**的脚本；



1.3 安装

我们提供以下方法帮助安装所有需要的软件，可根据需要选择其中之一。

1.3.1 IDE

备注： 建议您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

请根据您的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-C2 构建固件，包括 Python、Git、交叉编译器、CMake 和 Ninja 编译工具等。

在本入门指南中，我们通过 **命令提示符**进行有关操作。不过，您在安装 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：控制面板-更改日期、时间或数字格式-管理选项卡-更改系统地域-勾选选项 “Beta：使用 Unicode UTF-8 支持全球语言” -点击确定-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`:

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

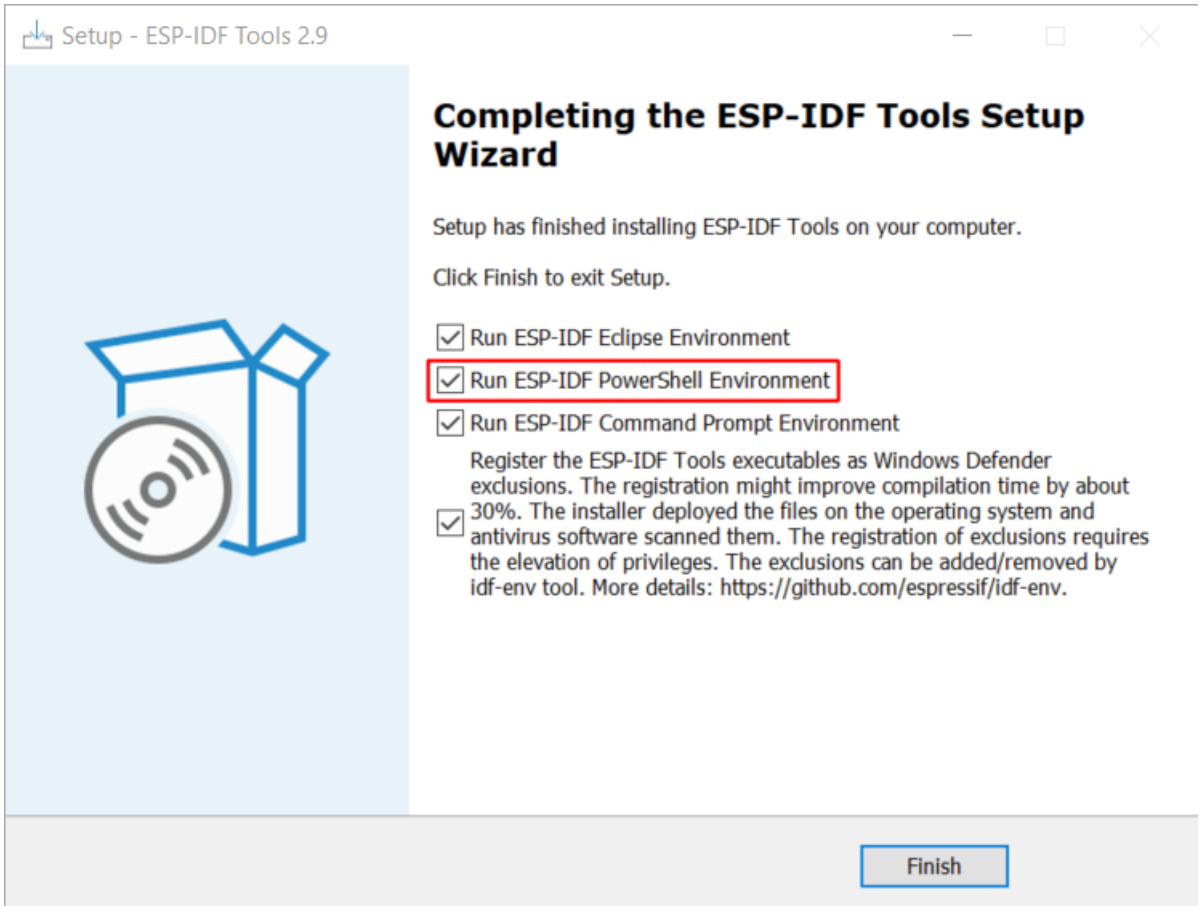


图 1: 完成 ESP-IDF 工具安装向导时运行 Run ESP-IDF PowerShell Environment

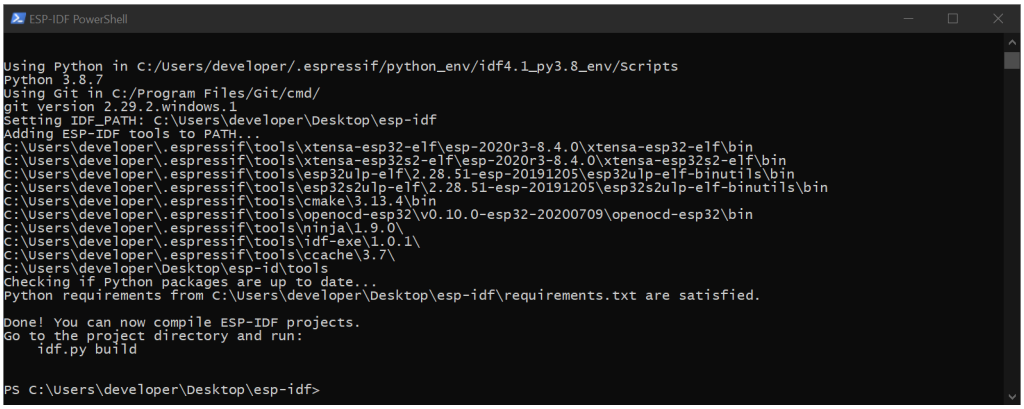


图 2: ESP-IDF PowerShell

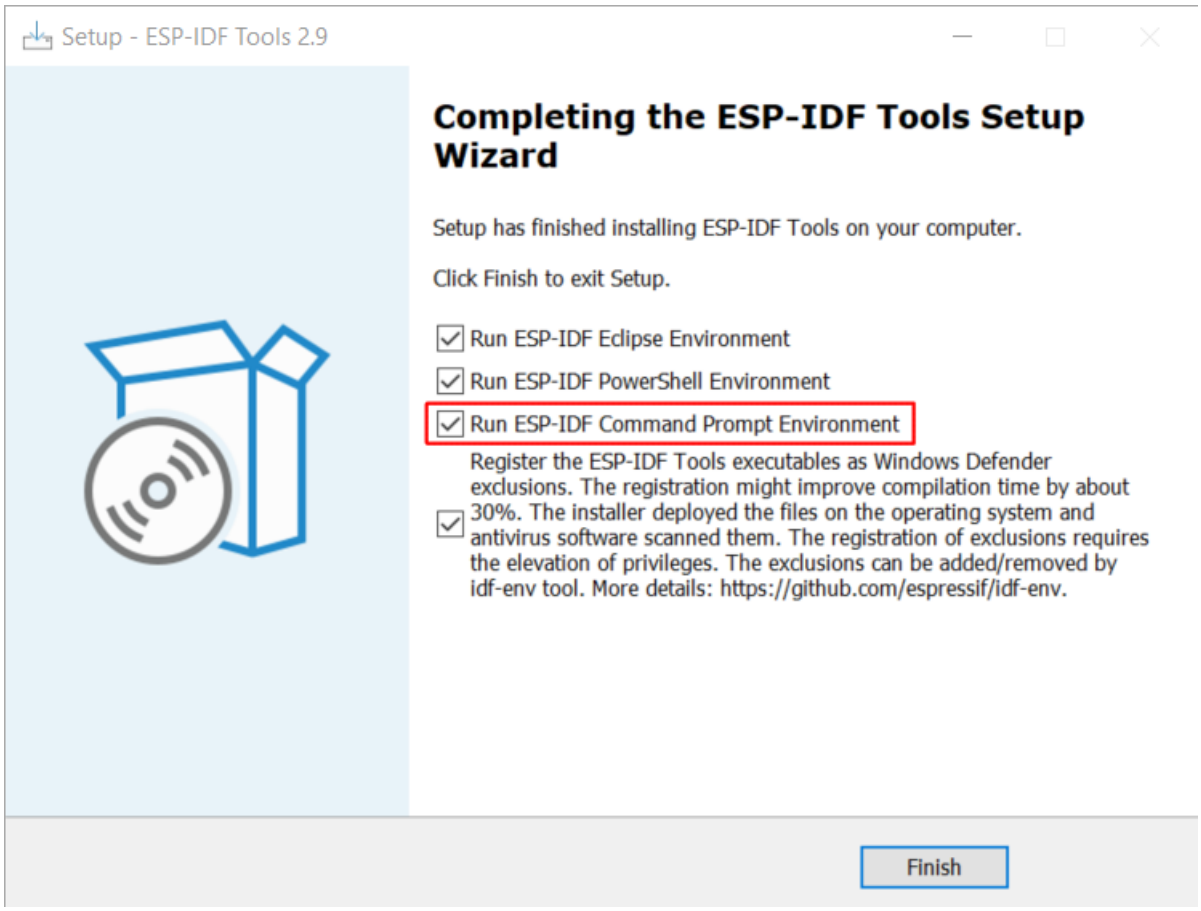


图 3: 完成 ESP-IDF 工具安装向导时运行 Run ESP-IDF Command Prompt (cmd.exe)

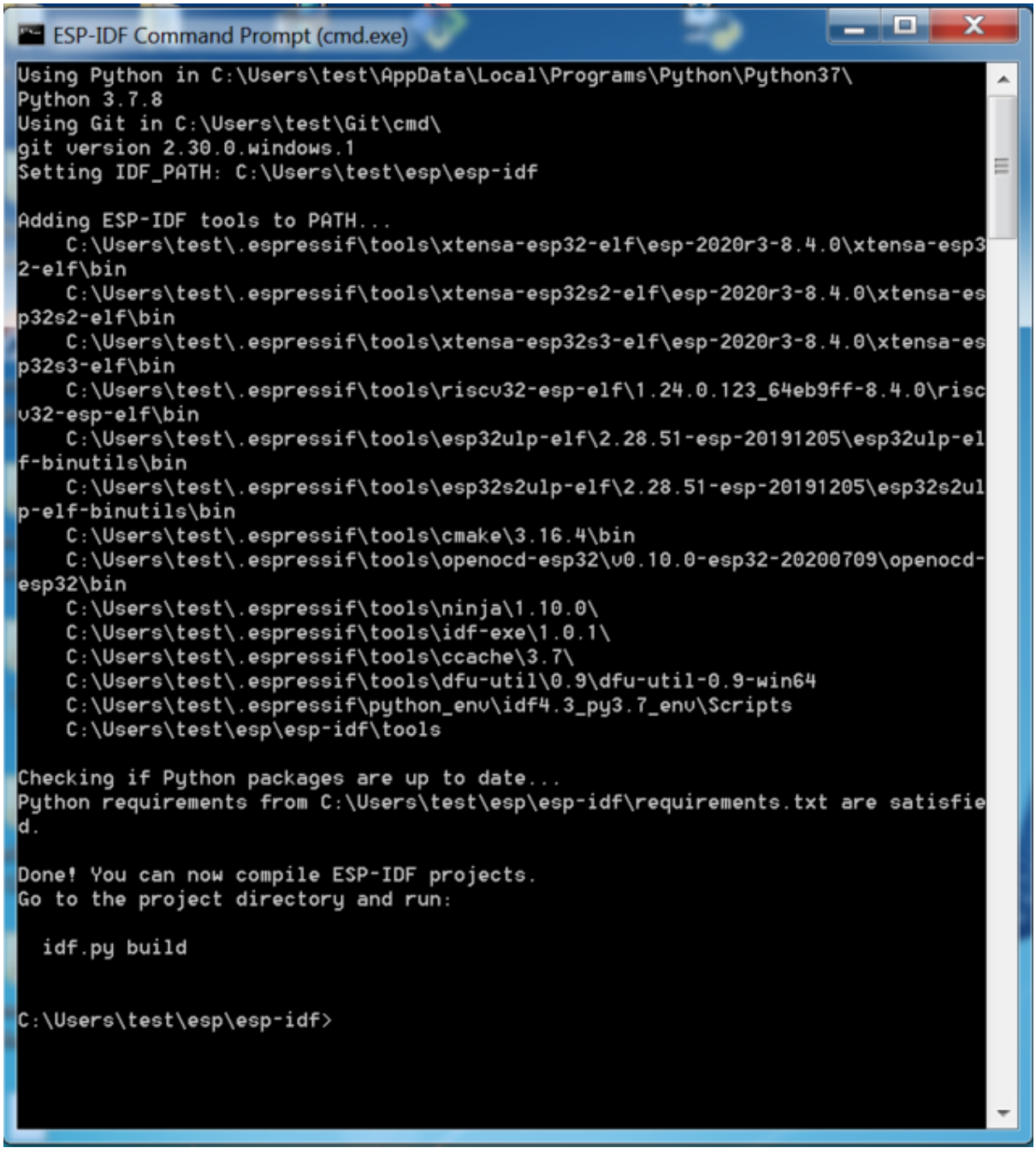


图 4: ESP-IDF 命令提示符窗口

使用命令提示符 在后续步骤中，我们将使用 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，您可还以通过 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。如果您的电脑上存在多个 ESP-IDF 路径（比如您需要不同版本的 ESP-IDF），您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 工作路径指定为您希望使用的 ESP-IDF 路径。
2. 或者，您可以运行 `cmd.exe`，并切换至您希望使用的 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。如果您在使用时遇到有关“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 现在您已经具备了使用 ESP-IDF 的所有条件，接下来将介绍如何开始您的第一个工程。

本指南将帮助您完成使用 ESP-IDF 的第一步。按照本指南，您将使用 ESP32-C2 创建第一个工程，并构建、烧录和监控设备输出。

备注： 如果您还未安装 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，您可以准备开发 ESP32-C2 应用程序了。您可以从 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 工程复制至您本地的 `~/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，您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，请将您的 ESP32-C2 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-C2 创建串口连接*。

备注： 请记住串口名，您会在后续步骤中使用。

配置工程 请进入 `hello_world` 目录，设置 ESP32-C2 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 esp32c2
idf.py menuconfig
```

打开一个新工程后，应首先使用 `idf.py set-target esp32c2` 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）。您也可以直接将“目标”配置为环境变量（此时可跳过该步骤）。更多信息，请见*Select the Target Chip: set-target*。

正确操作上述步骤后，系统将显示以下菜单：

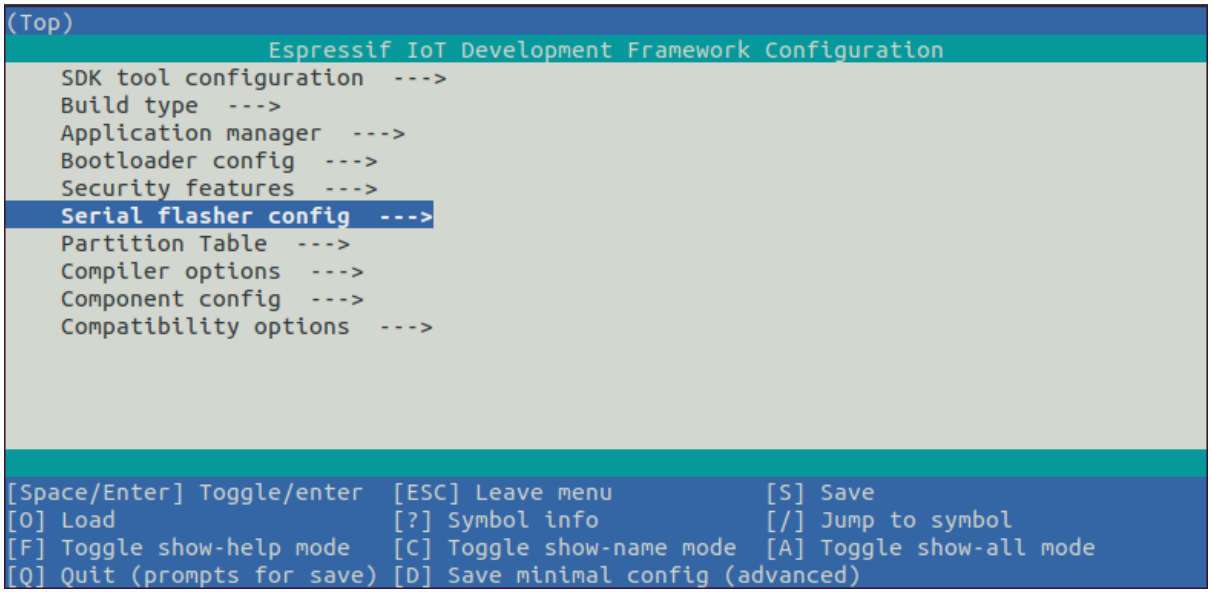


图 5: 工程配置—主窗口

您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： 您终端窗口中显示出的菜单颜色可能会与上图不同。您可以通过选项 `--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
```

(下页继续)

(续上页)

```
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您的 ESP32-C2 开发板：

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

请将 PORT 替换为 ESP32-C2 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，请前往烧录故障排除 或与 *ESP32-C2* 创建串口连接 获取更多详细信息。

常规操作 在烧录过程中，您会看到类似如下的输出日志：

```
...
esptool.py esp32c2 -p /dev/ttyUSB0 -b 460800 --before=default_reset --after=hard_
↪reset write_flash --flash_mode dio --flash_freq 60m --flash_size 2MB 0x0␣
↪bootloader/bootloader.bin 0x10000 hello_world.bin 0x8000 partition_table/
↪partition-table.bin
esptool.py v3.3.1
Serial port /dev/ttyUSB0
Connecting....
Chip is ESP32-C2 (revision 1)
Features: Wi-Fi
Crystal is 40MHz
MAC: 10:97:bd:f0:e5:0c
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00000000 to 0x00004fff...
Flash will be erased from 0x00010000 to 0x0002ffff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 18192 bytes to 10989...
Writing at 0x00000000... (100 %)
Wrote 18192 bytes (10989 compressed) at 0x00000000 in 0.6 seconds (effective 248.5␣
↪kbit/s)...
Hash of data verified.
Compressed 128640 bytes to 65895...
Writing at 0x00010000... (20 %)
Writing at 0x00019539... (40 %)
Writing at 0x00020bf2... (60 %)
Writing at 0x00027de1... (80 %)
Writing at 0x0002f480... (100 %)
Wrote 128640 bytes (65895 compressed) at 0x00010000 in 1.7 seconds (effective 603.
↪0 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 360.1␣
↪kbit/s)...
```

(下页继续)

(续上页)

```
Hash of data verified.

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

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。

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

监视输出 您可以使用 `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 替换为您的串口名称。

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，您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 esp32c2 chip with 1 CPU core(s), WiFi/BLE, silicon revision 0, 2 MB␣
↪embedded flash
Minimum free heap size: 203888 bytes
    Restarting in 9 seconds...
    Restarting in 8 seconds...
    Restarting in 7 seconds...
```

您可使用快捷键 `Ctrl+]`，退出 IDF 监视器。

如果 IDF 监视器在烧录后很快发生错误，或打印信息全是乱码（如下），很有可能是因为您的开发板采用了 26 MHz 晶振，而 ESP-IDF 默认支持大多数开发板使用的 40 MHz 晶振。

```
e@@@)(Xn@@y.!@@(@PW+)@@Hn9a￣/9@!@t5@@P@~@k@@e@ea@5@jA
~zY@@Y(1@,1 @@ e@@@)(Xn@@y.!Dr@zY(@ jpi@|@+z5Ymvp
```

此时，您可以：

1. 退出监视器。
2. 返回 `menuconfig`。
3. 进入 `Component config` -> `Hardware Settings` -> `Main XTAL Config` -> `Main XTAL frequency` 进行配置，将*CONFIG_XTAL_FREQ_SEL* 设置为 26 MHz。
4. 重新 编译和烧录应用程序。

在当前的 ESP-IDF 版本中，ESP32-C2 支持的主晶振频率如下：

- 26 MHz
- 40 MHz

备注： 您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 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，您已完成 ESP32-C2 的入门学习！

现在，您可以尝试一些其他 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-C2，因为 ESP32-C2 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-C2，或者不存在这个表格，那么即表示 ESP32-C2 支持这个示例程序。

其他提示

权限问题 /dev/ttyUSB0 使用某些 Linux 版本向 ESP32-C2 烧录固件时，可能会出现 `Failed to open port /dev/ttyUSB0` 错误消息。此时可以将用户添加至*Linux Dialout 组*。

兼容的 Python 版本 ESP-IDF 支持 Python 3.7 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-C2 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。如您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，您可在该窗口使用所有已安装的工具。

有些情况下，您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，您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，您就可以通过命令提示符使用 ESP-IDF 工具了。

与 ESP32-C2 创建串口连接

可以使用 USB 至 UART 桥，与 ESP32-C2 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。

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-C2 之间通过 UART 连接。

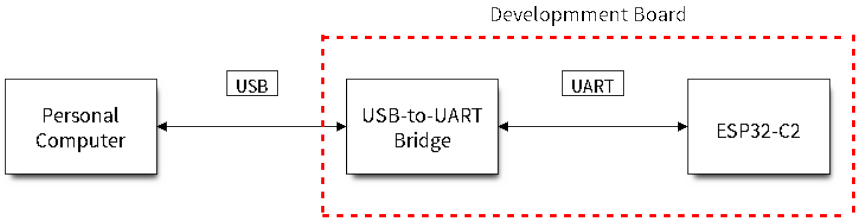


图 6: 安装有 USB 至 UART 桥的开发板

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 UART 进行烧录 本节描述如何使用 USB 至 UART 桥在 ESP32-C2 和 PC 之间建立串行连接。板上桥与外部桥均适用。

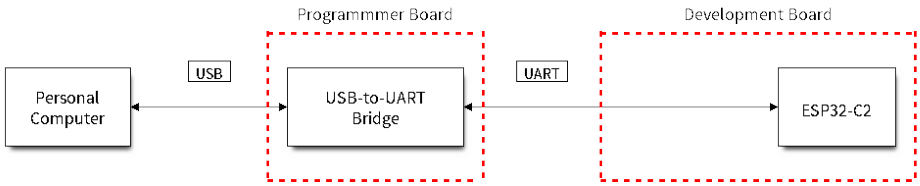


图 7: 外部 USB 至 UART 桥

连接 ESP32-C2 和 PC 用 USB 线将 ESP32-C2 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-C2 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-C2 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-C2 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，您可以运行以下命令，包括定义波特率的可选参数。

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

如需改变烧录器的波特率，请用需要的波特率代替 `BAUD`。默认的波特率为 `460800`。

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-C2 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-C2 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-C2 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-C2 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。

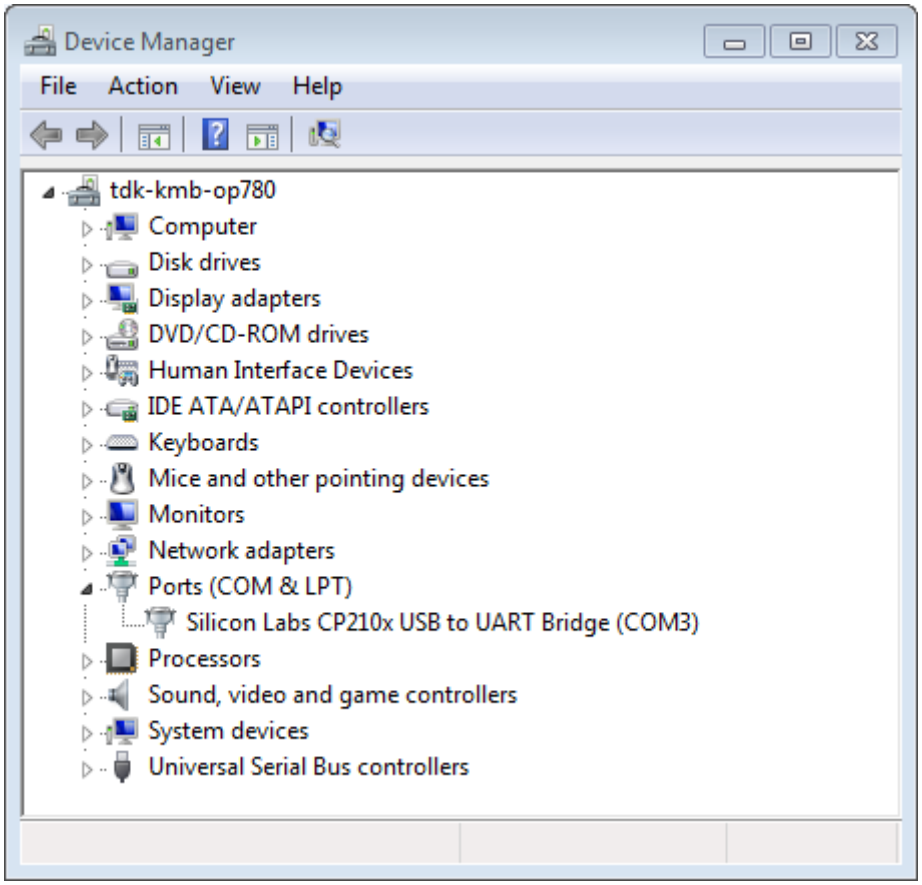


图 8: 设备管理器中 ESP32-DevKitC 的 USB 至 UART 桥

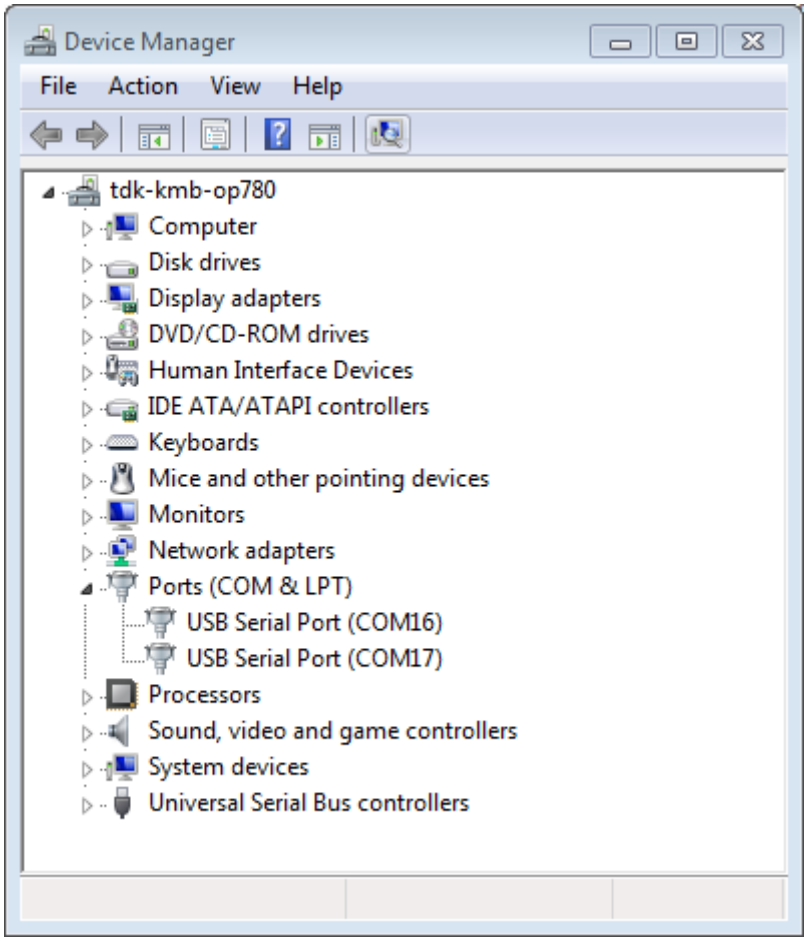


图 9: Windows 设备管理器中 ESP-WROVER-KIT 的两个 USB 串行端口

在 Linux 中添加用户到 dialout 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，您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-C2 后终端上是否有输出，从而验证串口连接是否可用。

使用 40 MHz 的 XTAL 时，ESP32-C2 的控制台波特率默认为 115200；使用 26 MHz 的 XTAL 时，其波特率默认为 74880。

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。您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。

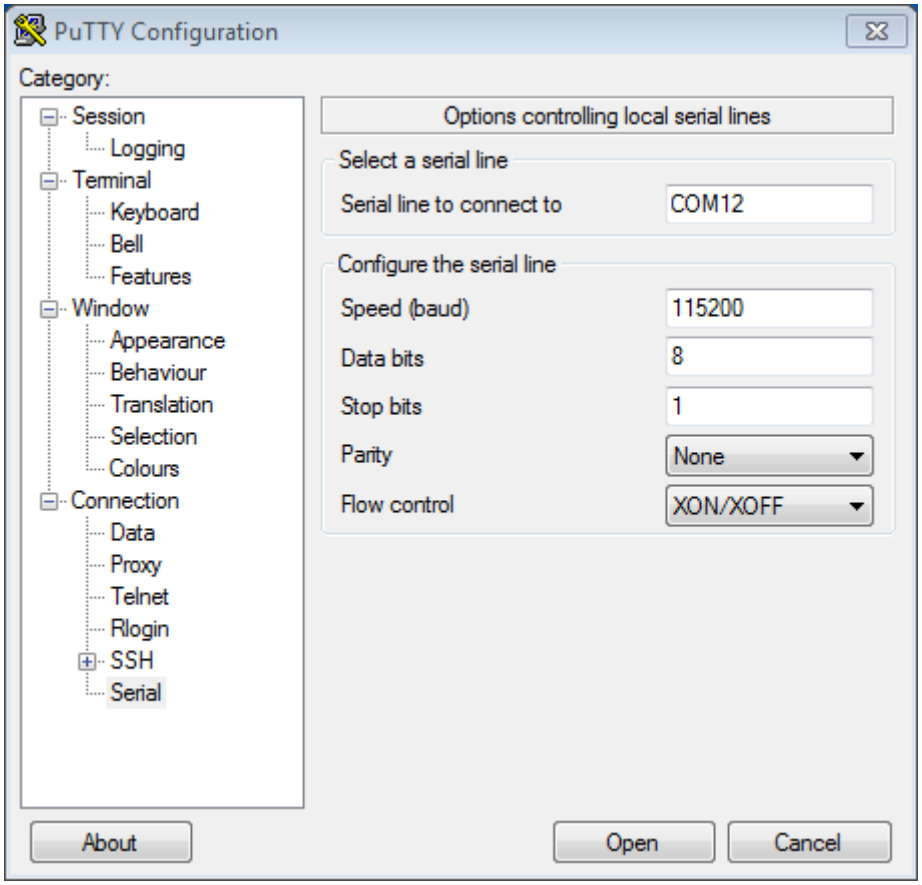


图 10: 在 Windows 操作系统中使用 PuTTY 设置串口通信参数

然后，请检查 ESP32-C2 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-C2 的应用程序，请参考输出示例。

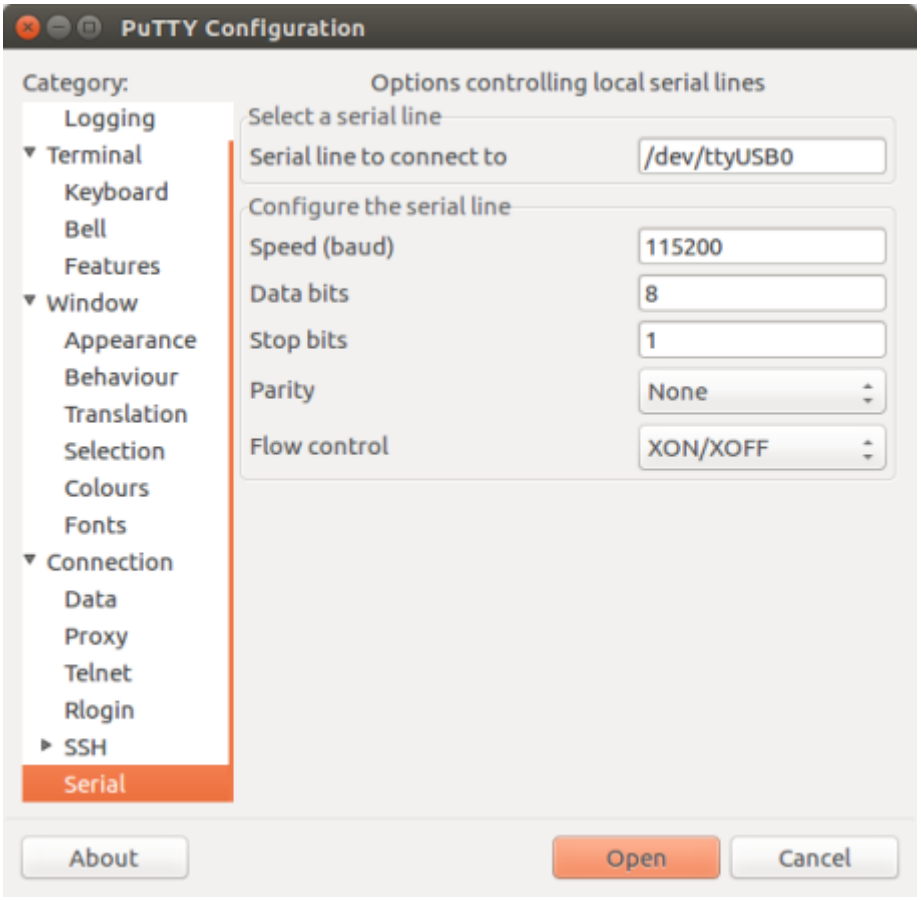


图 11: 在 Linux 操作系统中使用 PuTTY 设置串口通信参数

备注： 请在验证完串口通信正常后，关闭串口终端。如果您让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。

macOS 操作系统 macOS 提供了 **屏幕**命令，因此您不用安装串口终端程序。

- 参考*在 Linux 和 macOS 上查看端口*，运行以下命令:

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

- 您会看到类似如下输出:

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

- 根据您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。

- 您需要的正是 **屏幕**显示的日志。日志内容取决于加载到 ESP32-C2 的应用程序，请参考输出示例。请使用 Ctrl-A + \ 键退出 **屏幕**会话。

备注： 请在验证完串口通信正常后，关闭 **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
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，您可以继续进行安装，并最终将应用程序上载到 ESP32-C2。

备注： 在某些串口接线方式下，在 ESP32-C2 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 & GPIO0 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。

如您在安装 ESP32-C2 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-C2（请参考*与 ESP32-C2 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-C2 的 `GPIO9` 和 `CHIP_PU` (EN) 管脚上，因此 DTR 和 RTS 的电压电平变化会使 ESP32-C2 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：

- 您的硬件没有连接到 `GPIO9` 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，也可以将您的 ESP32-C2 开发板手动设置为固件下载模式（复位）。

- 对于乐鑫开发板，您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 (`GPIO9`) 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 `GPIO9` 拉低。

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 IDF 的其他特性。

在 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。

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 线暂停应用程序	重置目标设备，通过 RTS 线（如已连接）进入引导加载程序，此时开发板不运行任何程序。等待其他设备启动时可以使用此操作。
• 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 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
```

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`。

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]`。

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）。

通过将*CONFIG_ESP_SYSTEM_PANIC* 设置为 `GDBStub on 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，可打印任何内容。

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
```

IDF 监视器已知问题

Windows 环境下已知问题

- 由于 Windows 控制台限制，有些箭头键及其他一些特殊键无法在 GDB 中使用。
- 偶然情况下，`idf.py` 退出时，可能会在 IDF 监视器恢复之前暂停 30 秒。
- GDB 运行时，可能会暂停一段时间，然后才开始与 GDBStub 进行通信。

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-C2 设置 ESP-IDF 的具体步骤。

第一步：安装准备 为了在 ESP32-C2 中使用 ESP-IDF，需要根据操作系统安装一些软件包。以下安装指南可协助您安装 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-venv python3-
↪setuptools 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您使用的系统，请参考您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，您可以运行:

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

  - 若有 MacPorts，您可以运行:

    ```
    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: 如您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 如果您使用的是 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
```

您需要运行如下命令来安装 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您当前使用的 Python 版本:

```
python --version
```

如果输出结果是 `Python 2.7.17`，则代表您的默认解析器是 Python 2.7。这时需要您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-C2 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，切换到您要保存 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 v5.1 --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，您还需要为支持 ESP32-C2 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 esp32c2
```

或使用 Fish shell：

```
cd ~/esp/esp-idf
./install.fish esp32c2
```

上述命令仅仅为 ESP32-C2 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，如您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)
```

可运行您电脑 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
```

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。您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，重新设置环境变量 `IDF_TOOLS_PATH`。注意，请确保您的用户账号已经具备了读写该路径的权限。

如果修改了 `IDF_TOOLS_PATH` 变量，请确保该变量在每次执行安装脚本 (`install.bat`、`install.ps1` 或 `install.sh`) 和导出脚本 (`export.bat`、`export.ps1` 或 `export.sh`) 均保持一致。

第四步：设置环境变量 此时，您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！

如果您需要经常运行 ESP-IDF，您可以为执行 `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。

现在您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 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 现在您已经具备了使用 ESP-IDF 的所有条件，接下来将介绍如何开始您的第一个工程。

本指南将帮助您完成使用 ESP-IDF 的第一步。按照本指南，您将使用 ESP32-C2 创建第一个工程，并构建、烧录和监控设备输出。

备注： 如果您还未安装 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，您可以准备开发 ESP32-C2 应用程序了。您可以从 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 工程复制至您本地的 `~/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，您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，请将您的 ESP32-C2 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-C2 创建串口连接*。

备注： 请记住串口名，您会在后续步骤中使用。

配置工程 请进入 `hello_world` 目录，设置 ESP32-C2 为目标芯片，然后运行工程配置工具 `menuconfig`。

```
cd ~/esp/hello_world
idf.py set-target esp32c2
idf.py menuconfig
```

打开一个新工程后，应首先使用 `idf.py set-target esp32c2` 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）。您也可以直接将“目标”配置为环境变量（此时可跳过该步骤）。更多信息，请见*Select the Target Chip: set-target*。

正确操作上述步骤后，系统将显示以下菜单：

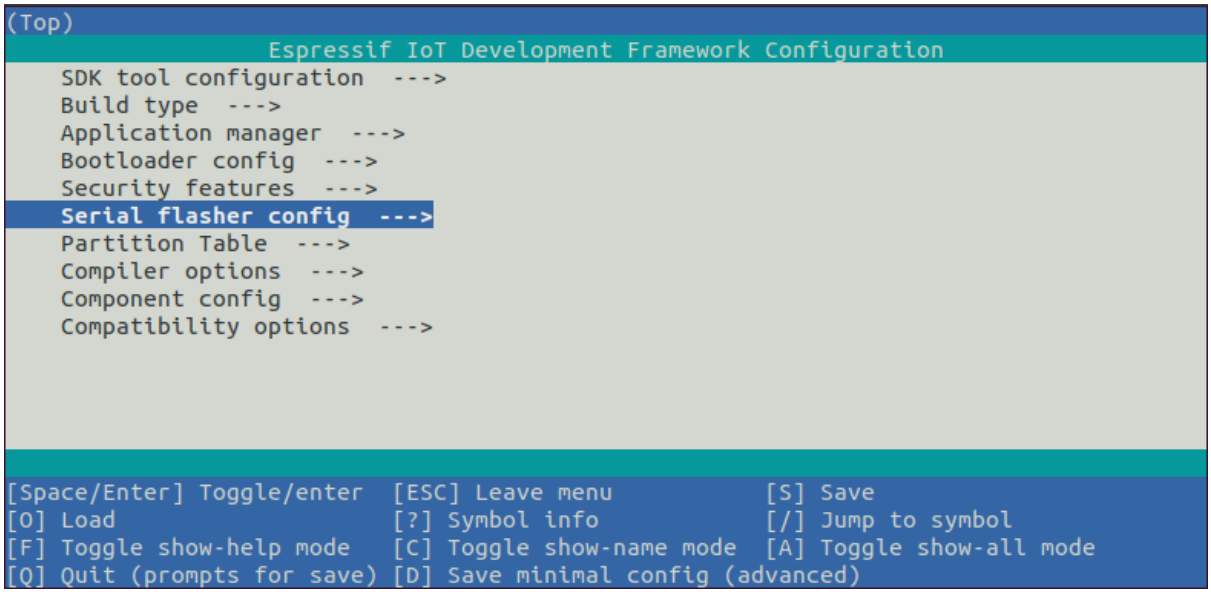


图 12: 工程配置—主窗口

您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： 您终端窗口中显示出的菜单颜色可能会与上图不同。您可以通过选项 `--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您的 ESP32-C2 开发板：

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

请将 PORT 替换为 ESP32-C2 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，请前往烧录故障排除 或与 *ESP32-C2* 创建串口连接 获取更多详细信息。

常规操作 在烧录过程中，您会看到类似如下的输出日志：

```
...
esptool.py esp32c2 -p /dev/ttyUSB0 -b 460800 --before=default_reset --after=hard_
↪reset write_flash --flash_mode dio --flash_freq 60m --flash_size 2MB 0x0␣
↪bootloader/bootloader.bin 0x10000 hello_world.bin 0x8000 partition_table/
↪partition-table.bin
esptool.py v3.3.1
Serial port /dev/ttyUSB0
Connecting....
Chip is ESP32-C2 (revision 1)
Features: Wi-Fi
Crystal is 40MHz
MAC: 10:97:bd:f0:e5:0c
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00000000 to 0x00004fff...
Flash will be erased from 0x00010000 to 0x0002ffff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 18192 bytes to 10989...
Writing at 0x00000000... (100 %)
Wrote 18192 bytes (10989 compressed) at 0x00000000 in 0.6 seconds (effective 248.5␣
↪kbit/s)...
Hash of data verified.
Compressed 128640 bytes to 65895...
Writing at 0x00010000... (20 %)
Writing at 0x00019539... (40 %)
Writing at 0x00020bf2... (60 %)
Writing at 0x00027de1... (80 %)
Writing at 0x0002f480... (100 %)
Wrote 128640 bytes (65895 compressed) at 0x00010000 in 1.7 seconds (effective 603.
↪0 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 360.1␣
↪kbit/s)...
Hash of data verified.

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

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。

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

监视输出 您可以使用 `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 替换为您的串口名称。

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，您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 esp32c2 chip with 1 CPU core(s), WiFi/BLE, silicon revision 0, 2 MB␣
↪embedded flash
Minimum free heap size: 203888 bytes
    Restarting in 9 seconds...
    Restarting in 8 seconds...
    Restarting in 7 seconds...
```

您可使用快捷键 `Ctrl+]`，退出 IDF 监视器。

如果 IDF 监视器在烧录后很快发生错误，或打印信息全是乱码（如下），很有可能是因为您的开发板采用了 26 MHz 晶振，而 ESP-IDF 默认支持大多数开发板使用的 40 MHz 晶振。

```
e@@@)(Xn@@y.!@@(@PW+)@@Hn9a￣/9@!@t5@@P@~@k@@e@ea@5@jA
~zY@@Y(1@,1 @@ e@@@)(Xn@@y.!Dr@zY(@ jpi@|@+z5Ymvp
```

此时，您可以：

1. 退出监视器。
2. 返回 `menuconfig`。
3. 进入 `Component config` –> `Hardware Settings` –> `Main XTAL Config` –> `Main XTAL frequency` 进行配置，将*CONFIG_XTAL_FREQ_SEL* 设置为 26 MHz。
4. 重新 编译和烧录应用程序。

在当前的 ESP-IDF 版本中，ESP32-C2 支持的主晶振频率如下：

- 26 MHz
- 40 MHz

备注： 您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 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，您已完成 ESP32-C2 的入门学习！

现在，您可以尝试一些其他 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-C2，因为 ESP32-C2 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-C2，或者不存在这个表格，那么即表示 ESP32-C2 支持这个示例程序。

其他提示

权限问题 /dev/ttyUSB0 使用某些 Linux 版本向 ESP32-C2 烧录固件时，可能会出现 `Failed to open port /dev/ttyUSB0` 错误消息。此时可以将用户添加至*Linux Dialout 组*。

兼容的 Python 版本 ESP-IDF 支持 Python 3.7 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，EESP-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，您在使用时，也应注意更新您本地的版本。最简单的方法是：直接删除您本地的 `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您使用的 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-C2 创建串口连接*
- Eclipse Plugin
- VSCode Extension
- *IDF 监视器*

1.4 编译第一个工程

如果您已经安装好 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。

Chapter 2

API 参考

2.1 API Conventions

This document describes conventions and assumptions common to ESP-IDF Application Programming Interfaces (APIs).

ESP-IDF provides several kinds of programming interfaces:

- C functions, structures, enums, type definitions and preprocessor macros declared in public header files of ESP-IDF components. Various pages in the API Reference section of the programming guide contain descriptions of these functions, structures and types.
- Build system functions, predefined variables and options. These are documented in the *build system guide*.
- *Kconfig* options can can be used in code and in the build system (CMakeLists.txt) files.
- *Host tools* and their command line parameters are also part of ESP-IDF interface.

ESP-IDF consists of components written specifically for ESP-IDF as well as third-party libraries. In some cases, an ESP-IDF-specific wrapper is added to the third-party library, providing an interface that is either simpler or better integrated with the rest of ESP-IDF facilities. In other cases, the original API of the third-party library is presented to the application developers.

Following sections explain some of the aspects of ESP-IDF APIs and their usage.

2.1.1 Error handling

Most ESP-IDF APIs return error codes defined with `esp_err_t` type. See *Error Handling* section for more information about error handling approaches. *Error Code Reference* contains the list of error codes returned by ESP-IDF components.

2.1.2 Configuration structures

重要： Correct initialization of configuration structures is an important part in making the application compatible with future versions of ESP-IDF.

Most initialization or configuration functions in ESP-IDF take as an argument a pointer to a configuration structure. For example:

```
const esp_timer_create_args_t my_timer_args = {
    .callback = &my_timer_callback,
    .arg = callback_arg,
    .name = "my_timer"
};
esp_timer_handle_t my_timer;
esp_err_t err = esp_timer_create(&my_timer_args, &my_timer);
```

Initialization functions never store the pointer to the configuration structure, so it is safe to allocate the structure on the stack.

The application must initialize all fields of the structure. The following is incorrect:

```
esp_timer_create_args_t my_timer_args;
my_timer_args.callback = &my_timer_callback;
/* Incorrect! Fields .arg and .name are not initialized */
esp_timer_create(&my_timer_args, &my_timer);
```

Most ESP-IDF examples use C99 designated initializers for structure initialization, since they provide a concise way of setting a subset of fields, and zero-initializing the remaining fields:

```
const esp_timer_create_args_t my_timer_args = {
    .callback = &my_timer_callback,
    /* Correct, fields .arg and .name are zero-initialized */
};
```

The C++ language supports designated initializers syntax, too, but the initializers must be in the order of declaration. When using ESP-IDF APIs in C++ code, you may consider using the following pattern:

```
/* Correct, fields .dispatch_method, .name and .skip_unhandled_events are zero-
↪initialized */
const esp_timer_create_args_t my_timer_args = {
    .callback = &my_timer_callback,
    .arg = &my_arg,
};

///* Incorrect, .arg is declared after .callback in esp_timer_create_args_t */
//const esp_timer_create_args_t my_timer_args = {
//    .arg = &my_arg,
//    .callback = &my_timer_callback,
//};
```

For more information on designated initializers, see *Designated initializers*. Note that C++ language versions older than C++20 (not the default in the current version of ESP-IDF) do not support designated initializers. If you have to compile code with an older C++ standard than C++20, you may use GCC extensions to produce the following pattern:

```
esp_timer_create_args_t my_timer_args = {};
/* All the fields are zero-initialized */
my_timer_args.callback = &my_timer_callback;
```

Default initializers

For some configuration structures, ESP-IDF provides macros for setting default values of fields:

```
httpd_config_t config = HTTPD_DEFAULT_CONFIG();
/* HTTPD_DEFAULT_CONFIG expands to a designated initializer.
   Now all fields are set to the default values.
   Any field can still be modified: */
config.server_port = 8081;
```

(下页继续)

(续上页)

```
httpd_handle_t server;
esp_err_t err = httpd_start(&server, &config);
```

It is recommended to use default initializer macros whenever they are provided for a particular configuration structure.

2.1.3 Private APIs

Certain header files in ESP-IDF contain APIs intended to be used only in ESP-IDF source code, and not by the applications. Such header files often contain `private` or `esp_private` in their name or path. Certain components, such as *hal* only contain private APIs.

Private APIs may be removed or changed in an incompatible way between minor or patch releases.

2.1.4 Components in example projects

ESP-IDF examples contain a variety of projects demonstrating usage of ESP-IDF APIs. In order to reduce code duplication in the examples, a few common helpers are defined inside components that are used by multiple examples. This includes components located in common_components directory, as well as some of the components located in the examples themselves. These components are not considered to be part of the ESP-IDF API.

It is not recommended to reference these components directly in custom projects (via `EXTRA_COMPONENT_DIRS` build system variable), as they may change significantly between ESP-IDF versions. When starting a new project based on an ESP-IDF example, copy both the project and the common components it depends on out of ESP-IDF, and treat the common components as part of the project. Note that the common components are written with examples in mind, and might not include all the error handling required for production applications. Take time to read the code and understand if it applicable to your use case.

2.1.5 API Stability

ESP-IDF uses Semantic Versioning as explained in the *versions page*.

Minor and bugfix releases of ESP-IDF guarantee compatibility with previous releases. The sections below explain different aspects and limitations to compatibility.

Source level compatibility

ESP-IDF guarantees source level compatibility of C functions, structures, enums, type definitions and preprocessor macros declared in public header files of ESP-IDF components. Source level compatibility implies that the application can be recompiled with the newer version of ESP-IDF without changes.

The following changes are allowed between minor versions and do not break source level compatibility:

- Deprecating functions (using the `deprecated` attribute) and header files (using a preprocessor `#warning`). Deprecations are listed in ESP-IDF relese notes. It is recommended to update the source code to use the newer functions or files that replace the deprecated ones, however this is not mandatory. Deprecated functions and files can be removed in major versions of ESP-IDF.
- Renaming components, moving source and header files between components —provided that the build system ensures that correct files are still found.
- Renaming Kconfig options. Kconfig system *renaming mechanism* ensures that the original Kconfig option names can still be used by the application in `sdkconfig` file, CMake files and source code.

Lack of binary compatibility

ESP-IDF does not guarantee binary compatibility between releases. This means that if a precompiled library is built with one ESP-IDF version, it is not guaranteed to work the same way with the next minor or bugfix release. The following are the possible changes that keep source level compatibility but not binary compatibility:

- Changing numerical values for C enum members.
- Adding new structure members or changing the order of members. See *Configuration structures* for tips that help ensure compatibility.
- Replacing an `extern` function with a `static inline` one with the same signature, or vice versa.
- Replacing a function-like macro with a compatible C function.

Other exceptions from compatibility

While we try to make upgrading to a new ESP-IDF version easy, there are parts of ESP-IDF that may change between minor versions in an incompatible way. We appreciate issue reports about any unintended breaking changes that don't fall into the categories below.

- *Private APIs*.
- *Components in example projects*.
- Features clearly marked as "beta", "preview", or "experimental".
- Changes made to mitigate security issues or to replace insecure default behaviors with a secure ones.
- Features which were never functional. For example, if it was never possible to use a certain function or an enumeration value, it may get renamed (as part of fixing it) or removed. This includes software features which depend on non-functional chip hardware features.
- Unexpected or undefined behavior (for example, due to missing validation of argument ranges) that is not documented explicitly may be fixed/changed.
- Location of *Kconfig* options in menuconfig.
- Location and names of example projects.

2.2 应用层协议

2.2.1 ASIO port

Asio is a cross-platform C++ library, see https://think-async.com/Asio/. It provides a consistent asynchronous model using a modern C++ approach.

The ESP-IDF component *ASIO* has been moved from ESP-IDF since version v5.0 to a separate repository:

- ASIO component on GitHub

To add ASIO component in your project, please run *idf.py add-dependency espressif/asio*

Hosted Documentation

The documentation can be found on the link below:

- ASIO documentation (English)

2.2.2 ESP-Modbus

The Espressif ESP-Modbus Library (esp-modbus) supports Modbus communication in the networks based on RS485, Wi-Fi, Ethernet interfaces. The ESP-IDF component *freemodbus* has been moved from ESP-IDF since version v5.0 to a separate repository:

- ESP-Modbus component on GitHub

Hosted Documentation

The documentation can be found on the link below:

- ESP-Modbus documentation (English)

Application Example

The examples below demonstrate the ESP-Modbus library of serial, TCP ports for slave and master implementations accordingly.

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

Please refer to the specific example README.md for details.

Protocol References

- `https://modbus.org/specs.php`: Modbus Organization with protocol specifications.

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。您也可以选择设置*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",
      .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 Server verification*。

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-WROOM-32SE 中的安全元素
- *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

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** (*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.

参数
- **client** –*MQTT* client handle
- **topic** –
- **qos** –// TODO describe parameters

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

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` 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.

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 otherwise

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.

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 **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.

const char ***certificate**

Certificate data, default is NULL, not required to verify the server.

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.

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

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`.

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.

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.

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

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

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

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

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 *esp_err_t* (***mqtt_event_callback_t**)(*esp_mqtt_event_handle_t* event)

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.

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

Overview

The ESP-TLS component provides a simplified API interface for accessing the commonly used TLS functionality. It supports common scenarios like CA certification validation, SNI, ALPN negotiation, non-blocking connection among others. All the configuration can be specified in the `esp_tls_cfg_t` data structure. Once done, TLS communication can be conducted using the following APIs:

- *`esp_tls_init()`*: for initializing the TLS connection handle.
- *`esp_tls_conn_new_sync()`*: for opening a new blocking TLS connection.
- *`esp_tls_conn_new_async()`*: for opening a new non-blocking TLS connection.
- *`esp_tls_conn_read()`*: for reading from the connection.
- *`esp_tls_conn_write()`*: for writing into the connection.

- *esp_tls_conn_destroy()*: for freeing up the connection.

Any application layer protocol like HTTP1, HTTP2 etc can be executed on top of this layer.

Application Example

Simple HTTPS example that uses ESP-TLS to establish a secure socket connection: protocols/https_request.

Tree structure for ESP-TLS component

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

The ESP-TLS component has a file esp-tls/esp_tls.h which contain the public API headers for the component. Internally ESP-TLS component uses one of the two SSL/TLS Libraries between mbedtls and wolfssl for its operation. API specific to mbedtls are present in esp-tls/private_include/esp_tls_mbedtls.h and API specific to wolfssl are present in esp-tls/private_include/esp_tls_wolfssl.h.

TLS Server verification

The ESP-TLS provides multiple options for TLS server verification on the client side. The ESP-TLS client can verify the server by validating the peer's server certificate or with the help of pre-shared keys. The user should select only one of the following options in the *esp_tls_cfg_t* structure for TLS server verification. If no option is selected then client will return a fatal error by default at the time of the TLS connection setup.

- **cacert_buf** and **cacert_bytes**: The CA certificate can be provided in a buffer to the *esp_tls_cfg_t* structure. The ESP-TLS will use the CA certificate present in the buffer to verify the server. The following variables in *esp_tls_cfg_t* structure must be set.
  - `cacert_buf` - pointer to the buffer which contains the CA cert.
  - `cacert_bytes` - size of the CA certificate in bytes.
- **use_global_ca_store**: The `global_ca_store` can be initialized and set at once. Then it can be used to verify the server for all the ESP-TLS connections which have set `use_global_ca_store = true` in their respective *esp_tls_cfg_t* structure. See API Reference section below on information regarding different API used for initializing and setting up the `global_ca_store`.
- **crt_bundle_attach**: The ESP x509 Certificate Bundle API provides an easy way to include a bundle of custom x509 root certificates for TLS server verification. More details can be found at *ESP x509 Certificate Bundle*
- **psk_hint_key**: To use pre-shared keys for server verification, *CONFIG_ESP_TLS_PSK_VERIFICATION* should be enabled in the ESP-TLS menuconfig. Then the pointer to PSK hint and key should be provided to the *esp_tls_cfg_t* structure. The ESP-TLS will use the PSK for server verification only when no other option regarding the server verification is selected.
- **skip server verification**: This is an insecure option provided in the ESP-TLS for testing purpose. The option can be set by enabling *CONFIG_ESP_TLS_INSECURE* and *CONFIG_ESP_TLS_SKIP_SERVER_CERT_VERIFY* in the ESP-TLS menuconfig. When this option is enabled the ESP-TLS will skip server verification by default when no other options for server verification are selected in the *esp_tls_cfg_t* structure. *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.*

ESP-TLS Server cert selection hook

The ESP-TLS component provides an option to set the server cert selection hook when using the mbedTLS stack. This provides an 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). To enable this feature, please enable *CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK* in the ESP-TLS menuconfig. The certificate selection callback can be configured in the `esp_tls_cfg_t` structure as follows:

```
int cert_selection_callback(mbedtls_ssl_context *ssl)
{
    /* Code that the callback should execute */
    return 0;
}

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

Underlying SSL/TLS Library Options

The ESP-TLS component has an option to use mbedtls or wolfssl as their underlying SSL/TLS library. By default only mbedtls is available and is used, wolfssl SSL/TLS library is available publicly at https://github.com/espressif/esp-wolfssl. The repository provides wolfssl component in binary format, it also provides few examples which are useful for understanding the API. Please refer the repository README.md for information on licensing and other options. Please see below option for using wolfssl in your project.

备注: *As the library options are internal to ESP-TLS, switching the libraries will not change ESP-TLS specific code for a project.*

How to use wolfssl with ESP-IDF

There are two ways to use wolfssl in your project

1) Directly add wolfssl as a component in your project with following three commands.:

```
(First change directory (cd) to your project directory)
mkdir components
cd components
git clone https://github.com/espressif/esp-wolfssl.git
```

2) Add wolfssl as an extra component in your project.

- Download wolfssl with:

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

- Include esp-wolfssl in ESP-IDF with setting EXTRA_COMPONENT_DIRS in CMakeLists.txt of your project as done in wolfssl/examples. For reference see Optional Project variables in *build-system*.

After above steps, you will have option to choose wolfssl as underlying SSL/TLS library in configuration menu of your project as follows:

```
idf.py menuconfig -> ESP-TLS -> choose SSL/TLS Library -> mbedtls/wolfssl
```

Comparison between mbedtls and wolfssl

The following table shows a typical comparison between wolfssl and mbedtls when protocols/https_request example *(which has server authentication)* was run with both SSL/TLS libraries and with all respective configurations set to default. *(mbedtls IN_CONTENT length and OUT_CONTENT length were set to 16384 bytes and 4096 bytes respectively)*

Property	Wolfssl	Mbedtls
Total Heap Consumed	~19 Kb	~37 Kb
Task Stack Used	~2.2 Kb	~3.6 Kb
Bin size	~858 Kb	~736 Kb

备注： *These values are subject to change with change in configuration options and version of respective libraries.*

API Reference

Header File

- components/esp-tls/esp_tls.h

Functions

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.

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 **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 (Integrated with ESP32-WROOM-32SE)

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.

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 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.

Header File

- components/esp-tls/esp_tls_errors.h

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）。

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。

API 参考

Header File

- components/esp_http_client/include/esp_http_client.h

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 re-use 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 contol over the redirection event.

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

返回

- ESP_OK
- ESP_FAIL

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 preferrable 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

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

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 compatability 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_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 Disgest 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 Local Control

Overview

ESP Local Control (**esp_local_ctrl**) component in ESP-IDF provides capability to control an ESP device over HTTPS or BLE. It provides access to application defined **properties** that are available for reading / writing via a set of configurable handlers.

Initialization of the **esp_local_ctrl** service over BLE transport is performed as follows:

```
esp_local_ctrl_config_t config = {
    .transport = ESP_LOCAL_CTRL_TRANSPORT_BLE,
    .transport_config = {
        .ble = & (protocomm_ble_config_t) {
```

(下页继续)

(续上页)

```
            .device_name  = SERVICE_NAME,
            .service_uuid = {
                /* LSB <---------------------------------------
                * ---------------------------------------> MSB */
                0x21, 0xd5, 0x3b, 0x8d, 0xbd, 0x75, 0x68, 0x8a,
                0xb4, 0x42, 0xeb, 0x31, 0x4a, 0x1e, 0x98, 0x3d
            }
        }
    },
    .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));
```

Similarly for HTTPS transport:

```
/* 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 */
```

(下页继续)

(续上页)

```
        .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));
```

You may set security for transport in ESP local control using following options:

1. *PROTOCOM_SEC2*: specifies that SRP6a based key exchange and end to end encryption based on AES-GCM is used. This is the most preffered option as it adds a robust security with Augmented PAKE protocol i.e. SRP6a.
2. *PROTOCOM_SEC1*: specifies that Curve25519 based key exchange and end to end encryption based on AES-CTR is used.
3. *PROTOCOM_SEC0*: specifies that data will be exchanged as a plain text (no security).
4. *PROTOCOM_SEC_CUSTOM*: you can define your own security requirement. Please note that you will also have to provide *custom_handle* of type *protocomm_security_t* * in this context.

备注： The respective security schemes need to be enabled through the project configuration menu. Please refer to the Enabling protocom security version section in *Protocol Communication* for more details.

Creating a property

Now that we know how to start the **esp_local_ctrl** service, let's add a property to it. Each property must have a unique *name* (string), a *type* (e.g. enum), *flags* (bit fields) and *size*.

The *size* is to be kept 0, if we want our property value to be of variable length (e.g. if its a string or bytestream). For fixed length property value data-types, like int, float, etc., setting the *size* field to the right value, helps **esp_local_ctrl** to perform internal checks on arguments received with write requests.

The interpretation of *type* and *flags* fields is totally upto the application, hence they may be used as enumerations, bit-fields, or even simple integers. One way is to use *type* values to classify properties, while *flags* to specify characteristics of a property.

Here is an example property which is to function as a timestamp. It is assumed that the application defines *TYPE_TIMESTAMP* and *READONLY*, which are used for setting the *type* and *flags* fields here.

```
/* 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);
```

Also notice that there is a ctx field, which is set to point to some custom *func_get_time()*. This can be used inside the property get / set handlers to retrieve timestamp.

Here is an example of *get_prop_values()* handler, which is used for retrieving the timestamp.

```
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;
}
```

Here is an example of *set_prop_values()* handler. Notice how we restrict from writing to read-only properties.

```
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);
            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;
}
```

For complete example see protocols/esp_local_ctrl

Client Side Implementation

The client side implementation will have establish a protocomm session with the device first, over the supported mode of transport, and then send and receive protobuf messages understood by the **esp_local_ctrl** service. The service will translate these messages into requests and then call the appropriate handlers (set / get). Then, the generated response for each handler is again packed into a protobuf message and transmitted back to the client.

See below the various protobuf messages understood by the **esp_local_ctrl** service:

1. *get_prop_count* : This should simply return the total number of properties supported by the service
2. *get_prop_values* : This accepts an array of indices and should return the information (name, type, flags) and values of the properties corresponding to those indices

3. *set_prop_values* : This accepts an array of indices and an array of new values, which are used for setting the values of the properties corresponding to the indices

Note that indices may or may not be the same for a property, across multiple sessions. Therefore, the client must only use the names of the properties to uniquely identify them. So, every time a new session is established, the client should first call *get_prop_count* and then *get_prop_values*, hence form an index to name mapping for all properties. Now when calling *set_prop_values* for a set of properties, it must first convert the names to indexes, using the created mapping. As emphasized earlier, the client must refresh the index to name mapping every time a new session is established with the same device.

The various protocomm endpoints provided by **esp_local_ctrl** are listed below:

表 1: Endpoints provided by ESP Local Control

Endpoint Name (BLE + GATT Server)	URI (HTTPS Server + mDNS)	Description
esp_local_ctrl/version	https://<mdns-hostname>.local/esp_local_ctrl/version	Endpoint used for retrieving version string
esp_local_ctrl/control	https://<mdns-hostname>.local/esp_local_ctrl/control	Endpoint used for sending / receiving control messages

API Reference

Header File

- components/esp_local_ctrl/include/esp_local_ctrl.h

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_BLUEDROID_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 Serial Slave Link

Overview

Espressif provides several chips that can work as slaves. These slave devices rely on some common buses, and have their own communication protocols over those buses. The *esp_serial_slave_link* component is designed for the master to communicate with ESP slave devices through those protocols over the bus drivers.

After an *esp_serial_slave_link* device is initialized properly, the application can use it to communicate with the ESP slave devices conveniently.

Espressif Device protocols

For more details about Espressif device protocols, see the following documents.

ESP SPI Slave HD (Half Duplex) Mode Protocol

SPI Slave Capabilities of Espressif chips

	ESP32	ESP32-S2	ESP32-C3
SPI Slave HD	N	Y (v2)	Y (v2)
Tohost intr		N	N
Frhost intr		2 *	2 *
TX DMA		Y	Y
RX DMA		Y	Y
Shared registers		72	64

Introduction In the half duplex mode, the master has to use the protocol defined by the slave to communicate with the slave. Each transaction may consist of the following phases (list by the order they should exist):

- Command: 8-bit, master to slave
  This phase determines the rest phases of the transactions. See *Supported Commands*.
- Address: 8-bit, master to slave, optional
  For some commands (WRBUF, RDBUF), this phase specifies the address of the shared buffer to write to/read from. For other commands with this phase, they are meaningless but still have to exist in the transaction.
- Dummy: 8-bit, floating, optional
  This phase is the turnaround time between the master and the slave on the bus, and also provides enough time for the slave to prepare the data to send to the master.

- Data: variable length, the direction is also determined by the command.
  This may be a data OUT phase, in which the direction is slave to master, or a data IN phase, in which the direction is master to slave.

The *direction* means which side (master or slave) controls the MOSI, MISO, WP, and HD pins.

Data IO Modes In some IO modes, more data wires can be used to send the data. As a result, the SPI clock cycles required for the same amount of data will be less than in the 1-bit mode. For example, in QIO mode, address and data (IN and OUT) should be sent on all 4 data wires (MOSI, MISO, WP, and HD). Here are the modes supported by the ESP32-S2 SPI slave and the wire number used in corresponding modes.

Mode	command WN	address WN	dummy cycles	data WN
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

Normally, which mode is used is determined by the command sent by the master (See *Supported Commands*), except the QPI mode.

QPI Mode The QPI mode is a special state of the SPI Slave. The master can send the ENQPI command to put the slave into the QPI mode state. In the QPI mode, the command is also sent in 4-bit, thus it's not compatible with the normal modes. The master should only send QPI commands when the slave is in QPI mode. To exit from the QPI mode, master can send the EXQPI command.

Supported Commands

备注： The command name is in a master-oriented direction. For example, WRBUF means master writes the buffer of slave.

Name	Description	Command	Address	Data
WRBUF	Write buffer	0x01	Buf addr	master to slave, no longer than buffer size
RDBUF	Read buffer	0x02	Buf addr	slave to master, no longer than buffer size
WRDMA	Write DMA	0x03	8 bits	master to slave, no longer than length provided by slave
RDDMA	Read DMA	0x04	8 bits	slave to master, no longer than length provided by slave
SEG_DONE	Segments done	0x05	•	•
ENQPI	Enter QPI mode	0x06	•	•
WR_DONE	Write segments done	0x07	•	•
CMD8	Interrupt	0x08	•	•
CMD9	Interrupt	0x09	•	•
CMDA	Interrupt	0x0A	•	•
EXQPI	Exit QPI mode	0xDD	•	•

Moreover, WRBUF, RDBUF, WRDMA, RDDMA commands have their 2-bit and 4-bit version. To do transactions in 2-bit or 4-bit mode, send the original command ORed by the corresponding command mask below. For example, command 0xA1 means WRBUF in QIO mode.

Mode	Mask
1-bit	0x00
DOUT	0x10
DIO	0x50
QOUT	0x20
QIO	0xA0
QPI	0xA0

Segment Transaction Mode Segment transaction mode is the only mode supported by the SPI Slave HD driver for now. In this mode, for a transaction the slave load onto the DMA, the master is allowed to read or write in segments. This way the master doesn't have to prepare a large buffer as the size of data provided by the slave. After the master finishes reading/writing a buffer, it has to send the corresponding termination command to the slave as a synchronization signal. The slave driver will update new data (if exist) onto the DMA upon seeing the termination command.

The termination command is WR_DONE (0x07) for the WRDMA and CMD8 (0x08) for the RDDMA.

Here's an example for the flow the master read data from the slave DMA:

1. The slave loads 4092 bytes of data onto the RDDMA
2. The master do seven RDDMA transactions, each of them is 512 bytes long, and reads the first 3584 bytes from the slave

3. The master do the last RDDMA transaction of 512 bytes (equal, longer, or shorter than the total length loaded by the slave are all allowed). The first 508 bytes are valid data from the slave, while the last 4 bytes are meaningless bytes.
4. The master sends CMD8 to the slave
5. The slave loads another 4092 bytes of data onto the RDDMA
6. The master can start new reading transactions after it sends the CMD8

Terminology

- ESSL: Abbreviation for ESP Serial Slave Link, the component described by this document.
- Master: The device running the *esp_serial_slave_link* component.
- ESSL device: a virtual device on the master associated with an ESP slave device. The device context has the knowledge of the slave protocol above the bus, relying on some bus drivers to communicate with the slave.
- ESSL device handle: a handle to ESSL device context containing the configuration, status and data required by the ESSL component. The context stores the driver configurations, communication state, data shared by master and slave, etc.
  The context should be initialized before it is used, and get deinitialized if not used any more. The master application operates on the ESSL device through this handle.
- ESP slave: the slave device connected to the bus, which ESSL component is designed to communicate with.
- Bus: The bus over which the master and the slave communicate with each other.
- Slave protocol: The special communication protocol specified by Espressif HW/SW over the bus.
- TX buffer num: a counter, which is on the slave and can be read by the master, indicates the accumulated buffer numbers that the slave has loaded to the hardware to receive data from the master.
- RX data size: a counter, which is on the slave and can be read by the master, indicates the accumulated data size that the slave has loaded to the hardware to send to the master.

Services provided by ESP slave

There are some common services provided by the Espressif slaves:

1. Tohost Interrupts: The slave can inform the master about certain events by the interrupt line. (optional)
2. Frhost Interrupts: The master can inform the slave about certain events.
3. Tx FIFO (master to slave): the slave can send data in stream to the master. The SDIO slave can also indicate it has new data to send to master by the interrupt line.
   The slave updates the TX buffer num to inform the master how much data it can receive, and the master then read the TX buffer num, and take off the used buffer number to know how many buffers are remaining.
4. Rx FIFO (slave to master): the slave can receive data from the master in units of receiving buffers.
   The slave updates the RX data size to inform the master how much data it has prepared to send, and then the master read the data size, and take off the data length it has already received to know how many data is remaining.
5. Shared registers: the master can read some part of the registers on the slave, and also write these registers to let the slave read.

The services provided by the slave depends on the slave’ s model. See *SPI Slave Capabilities of Espressif chips* for more details.

Initialization of ESP Serial Slave Link

ESP SDIO Slave The ESP SDIO slave link (ESSL SDIO) devices relies on the sdmmc component. It includes the usage of communicating with ESP SDIO Slave device via SDSPI feature. The ESSL device should be initialized as below:

1. Initialize a sdmmc card (see :doc:‘ Document of SDMMC driver </api-reference/storage/sdmmc>‘) structure.
2. Call `sdmmc_card_init()` to initialize the card.
3. Initialize the ESSL device with `essl_sdio_config_t`. The *card* member should be the `sdmmc_card_t` got in step 2, and the *recv_buffer_size* member should be filled correctly according to pre-negotiated value.

4. Call `essl_init()` to do initialization of the SDIO part.
5. Call `essl_wait_for_ready()` to wait for the slave to be ready.

ESP SPI Slave

备注： If you are communicating with the ESP SDIO Slave device through SPI interface, you should use the *SDIO interface* instead.

Hasn't been supported yet.

APIs

After the initialization process above is performed, you can call the APIs below to make use of the services provided by the slave:

Tohost Interrupts (optional)

1. Call `essl_get_intr_ena()` to know which events will trigger the interrupts to the master.
2. Call `essl_set_intr_ena()` to set the events that will trigger interrupts to the master.
3. Call `essl_wait_int()` to wait until interrupt from the slave, or timeout.
4. When interrupt is triggered, call `essl_get_intr()` to know which events are active, and call `essl_clear_intr()` to clear them.

Frhost Interrupts

1. Call `essl_send_slave_intr()` to trigger general purpose interrupt of the slave.

TX FIFO

1. Call `essl_get_tx_buffer_num()` to know how many buffers the slave has prepared to receive data from the master. This is optional. The master will poll *tx_buffer_num* when it try to send packets to the slave, until the slave has enough buffer or timeout.
2. Call `essl_send_packet()` to send data to the slave.

RX FIFO

1. Call `essl_get_rx_data_size()` to know how many data the slave has prepared to send to the master. This is optional. When the master tries to receive data from the slave, it will update the *rx_data_size* for once, if the current *rx_data_size* is shorter than the buffer size the master prepared to receive. And it may poll the *rx_data_size* if the *rx_dat_size* keeps 0, until timeout.
2. Call `essl_get_packet()` to receive data from the slave.

Reset counters (Optional) Call `essl_reset_cnt()` to reset the internal counter if you find the slave has reset its counter.

Application Example

The example below shows how ESP32-C2 SDIO host and slave communicate with each other. The host use the ESSL SDIO.

peripherals/sdio.

Please refer to the specific example README.md for details.

API Reference

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 Certificate Bundle

Overview

The ESP x509 Certificate Bundle API provides an easy way to include a bundle of custom x509 root certificates for TLS server verification.

备注： The bundle is currently not available when using WolfSSL.

The bundle comes with the complete list of root certificates from Mozilla's NSS root certificate store. Using the gen_crt_bundle.py python utility the certificates' subject name and public key are stored in a file and embedded in the ESP32-C2 binary.

When generating the bundle you may choose between:

- The full root certificate bundle from Mozilla, containing more than 130 certificates. The current bundle was updated Tue Jan 10 04:12:06 2023 GMT.
- A pre-selected filter list of the name of the most commonly used root certificates, reducing the amount of certificates to around 41 while still having around 90% absolute usage coverage and 99% market share coverage according to SSL certificate authorities statistics.

In addition it is possible to specify a path to a certificate file or a directory containing certificates which then will be added to the generated bundle.

备注： Trusting all root certificates means the list will have to be updated if any of the certificates are retracted. This includes removing them from *cacrt_all.pem*.

Configuration

Most configuration is done through menuconfig. CMake will generate the bundle according to the configuration and embed it.

- *CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*: automatically build and attach the bundle.
- *CONFIG_MBEDTLS_DEFAULT_CERTIFICATE_BUNDLE*: decide which certificates to include from the complete root list.
- *CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE_PATH*: specify the path of any additional certificates to embed in the bundle.

To enable the bundle when using ESP-TLS simply pass the function pointer to the bundle attach function:

```
esp_tls_cfg_t cfg = {
     .crt_bundle_attach = esp_crt_bundle_attach,
};
```

This is done to avoid embedding the certificate bundle unless activated by the user.

If using mbedTLS directly then the bundle may be activated by directly calling the attach function during the setup process:

```
mbedtls_ssl_config conf;
mbedtls_ssl_config_init(&conf);

esp_crt_bundle_attach(&conf);
```

Generating the List of Root Certificates

The list of root certificates comes from Mozilla's NSS root certificate store, which can be found here The list can be downloaded and created by running the script `mk-ca-bundle.pl` that is distributed as a part of curl. Another alternative would be to download the finished list directly from the curl website: CA certificates extracted from Mozilla

The common certificates bundle were made by selecting the authorities with a market share of more than 1 % from w3tech's SSL Survey. These authorities were then used to pick the names of the certificates for the filter list, *cmn_crt_authorities.csv*, from this list provided by Mozilla.

Updating the Certificate Bundle

The bundle is embedded into the app and can be updated along with the app by an OTA update. If you want to include a more up-to-date bundle than the bundle currently included in ESP-IDF, then the certificate list can be downloaded from Mozilla as described in *Generating the List of Root Certificates*.

Application Example

Simple HTTPS example that uses ESP-TLS to establish a secure socket connection using the certificate bundle with two custom certificates added for verification: protocols/https_x509_bundle.

HTTPS example that uses ESP-TLS and the default bundle: protocols/https_request.

HTTPS example that uses mbedTLS and the default bundle: protocols/https_mbedtls.

API Reference

Header File

- components/mbedtls/esp_crt_bundle/include/esp_crt_bundle.h

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 occured 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 occured or an action must be taken by the calling process.

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 */
        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);
```

(下页继续)

(续上页)

```
    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

Functions

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

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 incase 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
- /foo/? matches /foo and /foo/
- /foo/* (sans the backslash) matches /foo/ and /foo/bar, but not /foo or /fo
- /foo/?* or /foo/*? (sans the backslash) matches /foo/, /foo/bar, and also /foo, but not /foox or /fo

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

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 delimeters (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_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_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

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

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

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

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 occured 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 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_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 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，您只需要生成证书，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，您可以使用此功能。

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。

API 参考

Header File

- components/esp_https_server/include/esp_https_server.h

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

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

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

(http_server default config had to be copied for customization)

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 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 **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 Echo

Overview

ICMP (Internet Control Message Protocol) is used for diagnostic or control purposes or generated in response to errors in IP operations. The common network util `ping` is implemented based on the ICMP packets with the type field value of 0, also called `Echo Reply`.

During a ping session, the source host firstly sends out an ICMP echo request packet and wait for an ICMP echo reply with specific times. In this way, it also measures the round-trip time for the messages. After receiving a valid ICMP echo reply, the source host will generate statistics about the IP link layer (e.g. packet loss, elapsed time, etc).

It is common that IoT device needs to check whether a remote server is alive or not. The device should show the warnings to users when it got offline. It can be achieved by creating a ping session and sending/parsing ICMP echo packets periodically.

To make this internal procedure much easier for users, ESP-IDF provides some out-of-box APIs.

Create a new ping session To create a ping session, you need to fill in the `esp_ping_config_t` configuration structure firstly, specifying target IP address, interval times, and etc. Optionally, you can also register some callback functions with the *esp_ping_callbacks_t*' structure.

Example method to create a new ping session and register callbacks:

```
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));
    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);
```

(下页继续)

(续上页)

```
    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;
    esp_ping_new_session(&ping_config, &cbs, &ping);
}
```

Start and Stop ping session You can start and stop ping session with the handle returned by `esp_ping_new_session`. Note that, the ping session won't start automatically after creation. If the ping session is stopped, and restart again, the sequence number in ICMP packets will recount from zero again.

Delete a ping session If a ping session won't be used any more, you can delete it with `esp_ping_delete_session`. Please make sure the ping session is in stop state (i.e. you have called `esp_ping_stop` before or the ping session has finished all the procedures) when you call this function.

Get runtime statistics As the example code above, you can call `esp_ping_get_profile` to get different runtime statistics of ping session in the callback function.

Application Example

ICMP echo example: protocols/icmp_echo

API Reference

Header File

- components/lwip/include/apps/ping/ping_sock.h

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 is a C library that implements cryptographic primitives, X.509 certificate manipulation and the SSL/TLS and DTLS protocols. Its small code footprint makes it suitable for embedded systems.

备注： ESP-IDF uses a fork of Mbed TLS which includes a few patches (related to hardware routines of certain modules like `bignum (MPI)` and `ECC`) over vanilla Mbed TLS.

Mbed TLS supports SSL 3.0 up to TLS 1.3 and DTLS 1.0 to 1.2 communication by providing the following:

- TCP/IP communication functions: listen, connect, accept, read/write.
- SSL/TLS communication functions: init, handshake, read/write.
- X.509 functions: CRT, CRL and key handling
- Random number generation
- Hashing
- Encryption/decryption

备注： Mbed TLS is in the process of migrating all the documentation to a single place. In the meantime, users can find the documentation at the old Mbed TLS site .

Mbed TLS Support in ESP-IDF

Please find the information about the Mbed TLS versions present in different branches of ESP-IDF here.

备注： Please refer the *ESP-IDF Migration Guide* to migrate from Mbed TLS version 2.x to version 3.0 or greater.

Application Examples

Examples in ESP-IDF use *ESP-TLS* which provides a simplified API interface for accessing the commonly used TLS functionality.

Refer to the examples protocols/https_server/simple (Simple HTTPS server) and protocols/https_request (Make HTTPS requests) for more information.

If the Mbed TLS API is to be used directly, refer to the example protocols/https_mbedtls.

Alternatives

ESP-TLS acts as an abstraction layer over the underlying SSL/TLS library and thus has an option to use Mbed TLS or wolfSSL as the underlying library. By default, only Mbed TLS is available and used in ESP-IDF whereas wolfSSL is available publicly at https://github.com/espressif/esp-wolfSSL with the upstream submodule pointer.

Please refer to *ESP-TLS: Underlying SSL/TLS Library Options* docs for more information on this and comparison of Mbed TLS and wolfSSL.

Important Config Options

Following is a brief list of important config options accessible at `Component Config -> mbedTLS`. The full list of config options can be found *here*.

- *CONFIG_MBEDTLS_SSL_PROTO_TLS1_2*: Support for TLS 1.2
- *CONFIG_MBEDTLS_SSL_PROTO_TLS1_3*: Support for TLS 1.3

- *CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*: Support for trusted root certificate bundle (more about this: *ESP x509 Certificate Bundle*)
- *CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS*: Support for TLS Session Resumption: Client session tickets
- *CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS*: Support for TLS Session Resumption: Server session tickets
- *CONFIG_MBEDTLS_HARDWARE_SHA*: Support for hardware SHA acceleration
- *CONFIG_MBEDTLS_HARDWARE_ECC*: Support for hardware ECC acceleration

备注： Mbed TLS v3.0.0 and later support only TLS 1.2 and TLS 1.3 (SSL 3.0, TLS 1.0, TLS 1.1 and DTLS 1.0 are not supported). The support for TLS 1.3 is experimental and only supports the client-side. More information about this can be found out here.

Performance and Memory Tweaks

Reducing Heap Usage The following table shows typical memory usage with different configs when the protocols/https_request example (with Server Validation enabled) was run with Mbed TLS as the SSL/TLS library.

Mbed TLS Test	Related Configs	Heap Usage (approx.)
Default	NA	42196 B
Enable SSL Variable Length	*CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH*	42120 B
Disable Keep Peer Certificate	*CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE*	38533 B
Enable Dynamic TX/RX Buffer	*CONFIG_MBEDTLS_DYNAMIC_BUFFER* *CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA* *CONFIG_MBEDTLS_DYNAMIC_FREE_CA_CERT*	22013 B

备注： These values are subject to change with change in configuration options and versions of Mbed TLS.

Reducing Binary Size Under `Component Config -> mbedTLS`, there are multiple Mbed TLS features which are enabled by default but can be disabled if not needed to save code size. More information can be about this can be found in *Minimizing Binary Size* docs.

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 蓝牙 API

2.3.1 BT COMMON

BT GENERIC DEFINES

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_bt_defs.h

Structures

struct **esp_bt_uuid_t**

UUID type.

Public Members

uint16_t **len**

UUID length, 16bit, 32bit or 128bit

uint16_t **uuid16**

16bit UUID

uint32_t **uuid32**

32bit UUID

uint8_t **uuid128**[ESP_UUID_LEN_128]

128bit UUID

union *esp_bt_uuid_t*::[anonymous] **uuid**

UUID

Macros

ESP_BLUEDROID_STATUS_CHECK(status)

ESP_BT_STATUS_BASE_FOR_HCI_ERR

ESP_BT_OCTET16_LEN

ESP_BT_OCTET8_LEN

ESP_DEFAULT_GATT_IF

Default GATT interface id.

ESP_BLE_PRIM_ADV_INT_MIN

Minimum advertising interval for undirected and low duty cycle directed advertising

ESP_BLE_PRIM_ADV_INT_MAX

Maximum advertising interval for undirected and low duty cycle directed advertising

ESP_BLE_CONN_INT_MIN

relate to BTM_BLE_CONN_INT_MIN in stack/btm_ble_api.h

ESP_BLE_CONN_INT_MAX

relate to BTM_BLE_CONN_INT_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_LATENCY_MAX

relate to ESP_BLE_CONN_LATENCY_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_SUP_TOUT_MIN

relate to BTM_BLE_CONN_SUP_TOUT_MIN in stack/btm_ble_api.h

ESP_BLE_CONN_SUP_TOUT_MAX

relate to ESP_BLE_CONN_SUP_TOUT_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_PARAM_UNDEF

ESP_BLE_SCAN_PARAM_UNDEF

ESP_BLE_IS_VALID_PARAM(x, min, max)

Check the param is valid or not.

ESP_UUID_LEN_16

ESP_UUID_LEN_32

ESP_UUID_LEN_128

ESP_BD_ADDR_LEN

Bluetooth address length.

ESP_BLE_ENC_KEY_MASK

Used to exchange the encryption key in the init key & response key.

ESP_BLE_ID_KEY_MASK

Used to exchange the IRK key in the init key & response key.

ESP_BLE_CSR_KEY_MASK

Used to exchange the CSRK key in the init key & response key.

ESP_BLE_LINK_KEY_MASK

Used to exchange the link key(this key just used in the BLE & BR/EDR coexist mode) in the init key & response key.

ESP_APP_ID_MIN

Minimum of the application id.

ESP_APP_ID_MAX

Maximum of the application id.

ESP_BD_ADDR_STR

ESP_BD_ADDR_HEX(addr)

Type Definitions

typedef uint8_t **esp_bt_octet16_t**[ESP_BT_OCTET16_LEN]

typedef uint8_t **esp_bt_octet8_t**[ESP_BT_OCTET8_LEN]

typedef uint8_t **esp_link_key**[ESP_BT_OCTET16_LEN]

typedef uint8_t **esp_bd_addr_t**[ESP_BD_ADDR_LEN]

Bluetooth device address.

typedef uint8_t **esp_ble_key_mask_t**

Enumerations

enum **esp_bt_status_t**

Status Return Value.

Values:

enumerator **ESP_BT_STATUS_SUCCESS**

enumerator **ESP_BT_STATUS_FAIL**

enumerator **ESP_BT_STATUS_NOT_READY**

enumerator **ESP_BT_STATUS_NOMEM**

enumerator **ESP_BT_STATUS_BUSY**

enumerator **ESP_BT_STATUS_DONE**

enumerator **ESP_BT_STATUS_UNSUPPORTED**

enumerator **ESP_BT_STATUS_PARM_INVALID**

enumerator **ESP_BT_STATUS_UNHANDLED**

enumerator **ESP_BT_STATUS_AUTH_FAILURE**

enumerator **ESP_BT_STATUS_RMT_DEV_DOWN**

enumerator **ESP_BT_STATUS_AUTH_REJECTED**

enumerator **ESP_BT_STATUS_INVALID_STATIC_RAND_ADDR**

enumerator **ESP_BT_STATUS_PENDING**

enumerator **ESP_BT_STATUS_UNACCEPT_CONN_INTERVAL**

enumerator **ESP_BT_STATUS_PARAM_OUT_OF_RANGE**

enumerator **ESP_BT_STATUS_TIMEOUT**

enumerator **ESP_BT_STATUS_PEER_LE_DATA_LEN_UNSUPPORTED**

enumerator **ESP_BT_STATUS_CONTROL_LE_DATA_LEN_UNSUPPORTED**

enumerator **ESP_BT_STATUS_ERR_ILLEGAL_PARAMETER_FMT**

enumerator **ESP_BT_STATUS_MEMORY_FULL**

enumerator **ESP_BT_STATUS_EIR_TOO_LARGE**

enumerator **ESP_BT_STATUS_HCI_SUCCESS**

enumerator **ESP_BT_STATUS_HCI_PENDING**

enumerator **ESP_BT_STATUS_HCI_ILLEGAL_COMMAND**

enumerator **ESP_BT_STATUS_HCI_NO_CONNECTION**

enumerator **ESP_BT_STATUS_HCI_HW_FAILURE**

enumerator **ESP_BT_STATUS_HCI_PAGE_TIMEOUT**

enumerator **ESP_BT_STATUS_HCI_AUTH_FAILURE**

enumerator **ESP_BT_STATUS_HCI_KEY_MISSING**

enumerator **ESP_BT_STATUS_HCI_MEMORY_FULL**

enumerator **ESP_BT_STATUS_HCI_CONNECTION_TOUT**

enumerator **ESP_BT_STATUS_HCI_MAX_NUM_OF_CONNECTIONS**

enumerator **ESP_BT_STATUS_HCI_MAX_NUM_OF_SCOS**

enumerator **ESP_BT_STATUS_HCI_CONNECTION_EXISTS**

enumerator **ESP_BT_STATUS_HCI_COMMAND_DISALLOWED**

enumerator **ESP_BT_STATUS_HCI_HOST_REJECT_RESOURCES**

enumerator **ESP_BT_STATUS_HCI_HOST_REJECT_SECURITY**

enumerator **ESP_BT_STATUS_HCI_HOST_REJECT_DEVICE**

enumerator **ESP_BT_STATUS_HCI_HOST_TIMEOUT**

enumerator **ESP_BT_STATUS_HCI_UNSUPPORTED_VALUE**

enumerator **ESP_BT_STATUS_HCI_ILLEGAL_PARAMETER_FMT**

enumerator **ESP_BT_STATUS_HCI_PEER_USER**

enumerator **ESP_BT_STATUS_HCI_PEER_LOW_RESOURCES**

enumerator **ESP_BT_STATUS_HCI_PEER_POWER_OFF**

enumerator **ESP_BT_STATUS_HCI_CONN_CAUSE_LOCAL_HOST**

enumerator **ESP_BT_STATUS_HCI_REPEATED_ATTEMPTS**

enumerator **ESP_BT_STATUS_HCI_PAIRING_NOT_ALLOWED**

enumerator **ESP_BT_STATUS_HCI_UNKNOWN_LMP_PDU**

enumerator **ESP_BT_STATUS_HCI_UNSUPPORTED_REM_FEATURE**

enumerator **ESP_BT_STATUS_HCI_SCO_OFFSET_REJECTED**

enumerator **ESP_BT_STATUS_HCI_SCO_INTERVAL_REJECTED**

enumerator **ESP_BT_STATUS_HCI_SCO_AIR_MODE**

enumerator **ESP_BT_STATUS_HCI_INVALID_LMP_PARAM**

enumerator **ESP_BT_STATUS_HCI_UNSPECIFIED**

enumerator **ESP_BT_STATUS_HCI_UNSUPPORTED_LMP_PARAMETERS**

enumerator **ESP_BT_STATUS_HCI_ROLE_CHANGE_NOT_ALLOWED**

enumerator **ESP_BT_STATUS_HCI_LMP_RESPONSE_TIMEOUT**

enumerator **ESP_BT_STATUS_HCI_LMP_ERR_TRANS_COLLISION**

enumerator **ESP_BT_STATUS_HCI_LMP_PDU_NOT_ALLOWED**

enumerator **ESP_BT_STATUS_HCI_ENCRY_MODE_NOT_ACCEPTABLE**

enumerator **ESP_BT_STATUS_HCI_UNIT_KEY_USED**

enumerator **ESP_BT_STATUS_HCI_QOS_NOT_SUPPORTED**

enumerator **ESP_BT_STATUS_HCI_INSTANT_PASSED**

enumerator **ESP_BT_STATUS_HCI_PAIRING_WITH_UNIT_KEY_NOT_SUPPORTED**

enumerator **ESP_BT_STATUS_HCI_DIFF_TRANSACTION_COLLISION**

enumerator **ESP_BT_STATUS_HCI_UNDEFINED_0x2B**

enumerator **ESP_BT_STATUS_HCI_QOS_UNACCEPTABLE_PARAM**

enumerator **ESP_BT_STATUS_HCI_QOS_REJECTED**

enumerator **ESP_BT_STATUS_HCI_CHAN_CLASSIF_NOT_SUPPORTED**

enumerator **ESP_BT_STATUS_HCI_INSUFFCIENT_SECURITY**

enumerator **ESP_BT_STATUS_HCI_PARAM_OUT_OF_RANGE**

enumerator **ESP_BT_STATUS_HCI_UNDEFINED_0x31**

enumerator **ESP_BT_STATUS_HCI_ROLE_SWITCH_PENDING**

enumerator **ESP_BT_STATUS_HCI_UNDEFINED_0x33**

enumerator **ESP_BT_STATUS_HCI_RESERVED_SLOT_VIOLATION**

enumerator **ESP_BT_STATUS_HCI_ROLE_SWITCH_FAILED**

enumerator **ESP_BT_STATUS_HCI_INQ_RSP_DATA_TOO_LARGE**

enumerator **ESP_BT_STATUS_HCI_SIMPLE_PAIRING_NOT_SUPPORTED**

enumerator **ESP_BT_STATUS_HCI_HOST_BUSY_PAIRING**

enumerator **ESP_BT_STATUS_HCI_REJ_NO_SUITABLE_CHANNEL**

enumerator **ESP_BT_STATUS_HCI_CONTROLLER_BUSY**

enumerator **ESP_BT_STATUS_HCI_UNACCEPT_CONN_INTERVAL**

enumerator **ESP_BT_STATUS_HCI_DIRECTED_ADVERTISING_TIMEOUT**

enumerator **ESP_BT_STATUS_HCI_CONN_TOUT_DUE_TO_MIC_FAILURE**

enumerator **ESP_BT_STATUS_HCI_CONN_FAILED_ESTABLISHMENT**

enumerator **ESP_BT_STATUS_HCI_MAC_CONNECTION_FAILED**

enum **esp_bt_dev_type_t**

Bluetooth device type.

Values:

enumerator **ESP_BT_DEVICE_TYPE_BREDR**

enumerator **ESP_BT_DEVICE_TYPE_BLE**

enumerator **ESP_BT_DEVICE_TYPE_DUMO**

enum **esp_ble_addr_type_t**

BLE device address type.

Values:

enumerator **BLE_ADDR_TYPE_PUBLIC**

enumerator **BLE_ADDR_TYPE_RANDOM**

enumerator **BLE_ADDR_TYPE_RPA_PUBLIC**

enumerator **BLE_ADDR_TYPE_RPA_RANDOM**

enum **esp_ble_wl_addr_type_t**

white list address type

Values:

enumerator **BLE_WL_ADDR_TYPE_PUBLIC**

enumerator **BLE_WL_ADDR_TYPE_RANDOM**

BT MAIN API

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_bt_main.h

Functions

esp_bluedroid_status_t **esp_bluedroid_get_status**(void)

Get bluetooth stack status.

返回 Bluetooth stack status

esp_err_t **esp_bluedroid_enable**(void)

Enable bluetooth, must after esp_bluedroid_init().

返回

- ESP_OK : Succeed
- Other : Failed

esp_err_t **esp_bluedroid_disable**(void)

Disable bluetooth, must prior to esp_bluedroid_deinit().

返回

- ESP_OK : Succeed
- Other : Failed

esp_err_t **esp_bluedroid_init**(void)

Init and alloc the resource for bluetooth, must be prior to every bluetooth stuff.

返回

- ESP_OK : Succeed
- Other : Failed

esp_err_t **esp_bluedroid_deinit**(void)

Deinit and free the resource for bluetooth, must be after every bluetooth stuff.

返回

- ESP_OK : Succeed
- Other : Failed

Enumerations

enum **esp_bluedroid_status_t**

Bluetooth stack status type, to indicate whether the bluetooth stack is ready.

Values:

enumerator **ESP_BLUEDROID_STATUS_UNINITIALIZED**

Bluetooth not initialized

enumerator **ESP_BLUEDROID_STATUS_INITIALIZED**

Bluetooth initialized but not enabled

enumerator **ESP_BLUEDROID_STATUS_ENABLED**

Bluetooth initialized and enabled

BT DEVICE APIs

Overview Bluetooth device reference APIs.

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_bt_device.h

Functions

const uint8_t ***esp_bt_dev_get_address**(void)

Get bluetooth device address. Must use after "esp_bluedroid_enable" .

返回 bluetooth device address (six bytes), or NULL if bluetooth stack is not enabled

esp_err_t **esp_bt_dev_set_device_name**(const char *name)

Set bluetooth device name. This function should be called after esp_bluedroid_enable() completes successfully.

A BR/EDR/LE device type shall have a single Bluetooth device name which shall be identical irrespective of the physical channel used to perform the name discovery procedure.

参数 **name** –**[in]** : device name to be set

返回

- ESP_OK : Succeed
- ESP_ERR_INVALID_ARG : if name is NULL pointer or empty, or string length out of limit
- ESP_ERR_INVALID_STATE : if bluetooth stack is not yet enabled
- ESP_FAIL : others

2.3.2 BT LE

GAP API

Application Example Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a SMP security client demo and its tutorial. This demo initiates its security parameters and acts as a GATT client, which can send a security request to the peer device and then complete the encryption procedure.
  - bluetooth/bluedroid/ble/gatt_security_client
  - GATT Security Client Example Walkthrough
- This is a SMP security server demo and its tutorial. This demo initiates its security parameters and acts as a GATT server, which can send a pair request to the peer device and then complete the encryption procedure.
  - bluetooth/bluedroid/ble/gatt_security_server
  - GATT Security Server Example Walkthrough

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_gap_ble_api.h

Functions

esp_err_t **esp_ble_gap_register_callback**(*esp_gap_ble_cb_t* callback)

This function is called to occur gap event, such as scan result.

参数 **callback** –**[in]** callback function

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_config_adv_data**(*esp_ble_adv_data_t* *adv_data)

This function is called to override the BTA default ADV parameters.

参数 **adv_data** –**[in]** Pointer to User defined ADV data structure. This memory space can not be freed until callback of config_adv_data is received.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_scan_params**(*esp_ble_scan_params_t* *scan_params)

This function is called to set scan parameters.

参数 **scan_params** –**[in]** Pointer to User defined scan_params data structure. This memory space can not be freed until callback of set_scan_params

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_start_scanning**(uint32_t duration)

This procedure keep the device scanning the peer device which advertising on the air.

参数 **duration** –**[in]** Keeping the scanning time, the unit is second.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_stop_scanning**(void)

This function call to stop the device scanning the peer device which advertising on the air.

返回

- ESP_OK : success
  - other : failed

esp_err_t **esp_ble_gap_start_advertising**(*esp_ble_adv_params_t* *adv_params)

This function is called to start advertising.

参数 **adv_params** –**[in]** pointer to User defined adv_params data structure.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_stop_advertising**(void)

This function is called to stop advertising.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_update_conn_params**(*esp_ble_conn_update_params_t* *params)

Update connection parameters, can only be used when connection is up.

参数 **params** –**[in]** - connection update parameters

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_pkt_data_len**(*esp_bd_addr_t* remote_device, uint16_t tx_data_length)

This function is to set maximum LE data packet size.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_rand_addr**(*esp_bd_addr_t* rand_addr)

This function sets the static Random Address and Non-Resolvable Private Address for the application.

参数 **rand_addr** –**[in]** the random address which should be setting

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_clear_rand_addr**(void)

This function clears the random address for the application.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_config_local_privacy**(bool privacy_enable)

Enable/disable privacy on the local device.

参数 **privacy_enable** –**[in]** - enable/disable privacy on remote device.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_config_local_icon**(uint16_t icon)

set local gap appearance icon

参数 **icon** –**[in]** - External appearance value, these values are defined by the Bluetooth SIG, please refer to https://www.bluetooth.com/specifications/assigned-numbers/

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_update_whitelist**(bool add_remove, *esp_bd_addr_t* remote_bda, *esp_ble_wl_addr_type_t* wl_addr_type)

Add or remove device from white list.

参数

- **add_remove** –**[in]** the value is true if added the ble device to the white list, and false remove to the white list.
- **remote_bda** –**[in]** the remote device address add/remove from the white list.
- **wl_addr_type** –**[in]** whitelist address type

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_clear_whitelist**(void)

Clear all white list.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_get_whitelist_size**(uint16_t *length)

Get the whitelist size in the controller.

参数 **length** –**[out]** the white list length.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_prefer_conn_params**(*esp_bd_addr_t* bd_addr, uint16_t min_conn_int, uint16_t max_conn_int, uint16_t slave_latency, uint16_t supervision_tout)

This function is called to set the preferred connection parameters when default connection parameter is not desired before connecting. This API can only be used in the master role.

参数

- **bd_addr** –**[in]** BD address of the peripheral
- **min_conn_int** –**[in]** minimum preferred connection interval
- **max_conn_int** –**[in]** maximum preferred connection interval
- **slave_latency** –**[in]** preferred slave latency
- **supervision_tout** –**[in]** preferred supervision timeout

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_device_name**(const char *name)

Set device name to the local device.

参数 **name** –**[in]** - device name.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_get_device_name**(void)

Get device name of the local device.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_get_local_used_addr**(*esp_bd_addr_t* local_used_addr, uint8_t *addr_type)

This function is called to get local used address and address type. uint8_t *esp_bt_dev_get_address(void) get the public address.

参数

- **local_used_addr** –**[in]** - current local used ble address (six bytes)
- **addr_type** –**[in]** - ble address type

返回 - ESP_OK : success

- other : failed

uint8_t ***esp_ble_resolve_adv_data**(uint8_t *adv_data, uint8_t type, uint8_t *length)

This function is called to get ADV data for a specific type.

参数

- **adv_data** –**[in]** - pointer of ADV data which to be resolved
- **type** –**[in]** - finding ADV data type
- **length** –**[out]** - return the length of ADV data not including type

返回 pointer of ADV data

esp_err_t **esp_ble_gap_config_adv_data_raw**(uint8_t *raw_data, uint32_t raw_data_len)

This function is called to set raw advertising data. User need to fill ADV data by self.

参数

- **raw_data** –**[in]** : raw advertising data
- **raw_data_len** –**[in]** : raw advertising data length , less than 31 bytes

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_config_scan_rsp_data_raw**(uint8_t *raw_data, uint32_t raw_data_len)

This function is called to set raw scan response data. User need to fill scan response data by self.

参数

- **raw_data** –**[in]** : raw scan response data
- **raw_data_len** –**[in]** : raw scan response data length , less than 31 bytes

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_read_rssi**(*esp_bd_addr_t* remote_addr)

This function is called to read the RSSI of remote device. The address of link policy results are returned in the gap callback function with ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT event.

参数 **remote_addr** –**[in]** : The remote connection device address.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_add_duplicate_scan_exceptional_device**(*esp_ble_duplicate_exceptional_info_type_t* type, *esp_duplicate_info_t* device_info)

This function is called to add a device info into the duplicate scan exceptional list.

参数

- **type** –**[in]** device info type, it is defined in esp_ble_duplicate_exceptional_info_type_t when type is MESH_BEACON_TYPE, MESH_PROV_SRV_ADV or MESH_PROXY_SRV_ADV , device_info is invalid.
- **device_info** –**[in]** the device information.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_remove_duplicate_scan_exceptional_device**(*esp_ble_duplicate_exceptional_info_type_t* type, *esp_duplicate_info_t* device_info)

This function is called to remove a device info from the duplicate scan exceptional list.

参数

- **type** –**[in]** device info type, it is defined in esp_ble_duplicate_exceptional_info_type_t when type is MESH_BEACON_TYPE, MESH_PROV_SRV_ADV or MESH_PROXY_SRV_ADV , device_info is invalid.
- **device_info** –**[in]** the device information.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_clean_duplicate_scan_exceptional_list**(*esp_duplicate_scan_exceptional_list_type_t* list_type)

This function is called to clean the duplicate scan exceptional list. This API will delete all device information in the duplicate scan exceptional list.

参数 **list_type** –**[in]** duplicate scan exceptional list type, the value can be one or more of esp_duplicate_scan_exceptional_list_type_t.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_security_param**(*esp_ble_sm_param_t* param_type, void *value, uint8_t len)

Set a GAP security parameter value. Overrides the default value.

```
                Secure connection is highly recommended to avoid some major
                vulnerabilities like 'Impersonation in the Pin Pairing Protocol'
                (CVE-2020-26555) and 'Authentication of the LE Legacy Pairing
                Protocol'.

                To accept only `secure connection mode`, it is necessary do as␣
↪following:
```

(下页继续)

(续上页)

```
                1. Set bit `ESP_LE_AUTH_REQ_SC_ONLY` (`param_type` is
                `ESP_BLE_SM_AUTHEN_REQ_MODE`), bit `ESP_LE_AUTH_BOND` and bit
                `ESP_LE_AUTH_REQ_MITM` is optional as required.

                2. Set to `ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_ENABLE` (`param_
↪type` is
                `ESP_BLE_SM_ONLY_ACCEPT_SPECIFIED_SEC_AUTH`).
```

参数

- **param_type** –**[in]** : the type of the param which to be set
- **value** –**[in]** : the param value
- **len** –**[in]** : the length of the param value

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_security_rsp**(*esp_bd_addr_t* bd_addr, bool accept)

Grant security request access.

参数

- **bd_addr** –**[in]** : BD address of the peer
- **accept** –**[in]** : accept the security request or not

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_set_encryption**(*esp_bd_addr_t* bd_addr, *esp_ble_sec_act_t* sec_act)

Set a gap parameter value. Use this function to change the default GAP parameter values.

参数

- **bd_addr** –**[in]** : the address of the peer device need to encryption
- **sec_act** –**[in]** : This is the security action to indicate what kind of BLE security level is required for the BLE link if the BLE is supported

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_passkey_reply**(*esp_bd_addr_t* bd_addr, bool accept, uint32_t passkey)

Reply the key value to the peer device in the legacy connection stage.

参数

- **bd_addr** –**[in]** : BD address of the peer
- **accept** –**[in]** : passkey entry successful or declined.
- **passkey** –**[in]** : passkey value, must be a 6 digit number, can be lead by 0.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_confirm_reply**(*esp_bd_addr_t* bd_addr, bool accept)

Reply the confirm value to the peer device in the secure connection stage.

参数

- **bd_addr** –**[in]** : BD address of the peer device
- **accept** –**[in]** : numbers to compare are the same or different.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_remove_bond_device**(*esp_bd_addr_t* bd_addr)

Removes a device from the security database list of peer device. It manages unpairing event while connected.

参数 **bd_addr** –**[in]** : BD address of the peer device

返回 - ESP_OK : success
- other : failed

int **esp_ble_get_bond_device_num**(void)

Get the device number from the security database list of peer device. It will return the device bonded number immediately.

返回 - >= 0 : bonded devices number.
- ESP_FAIL : failed

esp_err_t **esp_ble_get_bond_device_list**(int *dev_num, *esp_ble_bond_dev_t* *dev_list)

Get the device from the security database list of peer device. It will return the device bonded information immediately.

参数
- **dev_num** –**[inout]** Indicate the dev_list array(buffer) size as input. If dev_num is large enough, it means the actual number as output. Suggest that dev_num value equal to esp_ble_get_bond_device_num().
- **dev_list** –**[out]** an array(buffer) of *esp_ble_bond_dev_t* type. Use for storing the bonded devices address. The dev_list should be allocated by who call this API.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_oob_req_reply**(*esp_bd_addr_t* bd_addr, uint8_t *TK, uint8_t len)

This function is called to provide the OOB data for SMP in response to ESP_GAP_BLE_OOB_REQ_EVT.

参数
- **bd_addr** –**[in]** BD address of the peer device.
- **TK** –**[in]** Temporary Key value, the TK value shall be a 128-bit random number
- **len** –**[in]** length of temporary key, should always be 128-bit

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_sc_oob_req_reply**(*esp_bd_addr_t* bd_addr, uint8_t p_c[16], uint8_t p_r[16])

This function is called to provide the OOB data for SMP in response to ESP_GAP_BLE_SC_OOB_REQ_EVT.

参数
- **bd_addr** –**[in]** BD address of the peer device.
- **p_c** –**[in]** Confirmation value, it shall be a 128-bit random number
- **p_r** –**[in]** Randomizer value, it should be a 128-bit random number

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_create_sc_oob_data**(void)

This function is called to create the OOB data for SMP when secure connection.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_disconnect**(*esp_bd_addr_t* remote_device)

This function is to disconnect the physical connection of the peer device gattc may have multiple virtual GATT server connections when multiple app_id registered. esp_ble_gattc_close (esp_gatt_if_t gattc_if, uint16_t conn_id) only close one virtual GATT server connection. if there exist other virtual GATT server connections, it does not disconnect the physical connection. esp_ble_gap_disconnect(esp_bd_addr_t remote_device) disconnect the physical connection directly.

参数 **remote_device** –**[in]** : BD address of the peer device

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_get_current_conn_params**(*esp_bd_addr_t* bd_addr, *esp_gap_conn_params_t* *conn_params)

This function is called to read the connection parameters information of the device.

参数

- **bd_addr** –**[in]** BD address of the peer device.
- **conn_params** –**[out]** the connection parameters information

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_gap_ble_set_channels**(*esp_gap_ble_channels* channels)

BLE set channels.

参数 **channels** –**[in]** : The n th such field (in the range 0 to 36) contains the value for the link layer channel index n. 0 means channel n is bad. 1 means channel n is unknown. The most significant bits are reserved and shall be set to 0. At least one channel shall be marked as unknown.

返回 - ESP_OK : success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

esp_err_t **esp_gap_ble_set_authorization**(*esp_bd_addr_t* bd_addr, bool authorize)

This function is called to authorized a link after Authentication(MITM protection)

参数
- **bd_addr** –**[in]** BD address of the peer device.
- **authorize** –**[out]** Authorized the link or not.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_read_phy**(*esp_bd_addr_t* bd_addr)

This function is used to read the current transmitter PHY and receiver PHY on the connection identified by remote address.

参数 **bd_addr** –**[in]** : BD address of the peer device

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_preferred_default_phy**(*esp_ble_gap_phy_mask_t* tx_phy_mask, *esp_ble_gap_phy_mask_t* rx_phy_mask)

This function is used to allows the Host to specify its preferred values for the transmitter PHY and receiver PHY to be used for all subsequent connections over the LE transport.

参数
- **tx_phy_mask** –**[in]** : indicates the transmitter PHYs that the Host prefers the Controller to use
- **rx_phy_mask** –**[in]** : indicates the receiver PHYs that the Host prefers the Controller to use

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_preferred_phy**(*esp_bd_addr_t* bd_addr, *esp_ble_gap_all_phys_t* all_phys_mask, *esp_ble_gap_phy_mask_t* tx_phy_mask, *esp_ble_gap_phy_mask_t* rx_phy_mask, *esp_ble_gap_prefer_phy_options_t* phy_options)

This function is used to set the PHY preferences for the connection identified by the remote address. The Controller might not be able to make the change (e.g. because the peer does not support the requested PHY) or may decide that the current PHY is preferable.

参数
- **bd_addr** –**[in]** : remote address
- **all_phys_mask** –**[in]** : a bit field that allows the Host to specify
- **tx_phy_mask** –**[in]** : a bit field that indicates the transmitter PHYs that the Host prefers the Controller to use
- **rx_phy_mask** –**[in]** : a bit field that indicates the receiver PHYs that the Host prefers the Controller to use
- **phy_options** –**[in]** : a bit field that allows the Host to specify options for PHYs

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_ext_adv_set_rand_addr**(uint8_t instance, *esp_bd_addr_t* rand_addr)

This function is used by the Host to set the random device address specified by the Random_Address parameter.

参数
- **instance** –**[in]** : Used to identify an advertising set
- **rand_addr** –**[in]** : Random Device Address

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_ext_adv_set_params**(uint8_t instance, const *esp_ble_gap_ext_adv_params_t* *params)

This function is used by the Host to set the advertising parameters.

参数
- **instance** –**[in]** : identifies the advertising set whose parameters are being configured.
- **params** –**[in]** : advertising parameters

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_config_ext_adv_data_raw**(uint8_t instance, uint16_t length, const uint8_t *data)

This function is used to set the data used in advertising PDUs that have a data field.

参数
- **instance** –**[in]** : identifies the advertising set whose data are being configured
- **length** –**[in]** : data length
- **data** –**[in]** : data information

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_config_ext_scan_rsp_data_raw**(uint8_t instance, uint16_t length, const uint8_t *scan_rsp_data)

This function is used to provide scan response data used in scanning response PDUs.

参数
- **instance** –**[in]** : identifies the advertising set whose response data are being configured.
- **length** –**[in]** : responsedata length
- **scan_rsp_data** –**[in]** : response data information

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_ext_adv_start**(uint8_t num_adv, const *esp_ble_gap_ext_adv_t* *ext_adv)

This function is used to request the Controller to enable one or more advertising sets using the advertising sets identified by the instance parameter.

参数
- **num_adv** –**[in]** : Number of advertising sets to enable or disable
- **ext_adv** –**[in]** : adv parameters

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_ext_adv_stop**(uint8_t num_adv, const uint8_t *ext_adv_inst)

This function is used to request the Controller to disable one or more advertising sets using the advertising sets identified by the instance parameter.

参数
- **num_adv** –**[in]** : Number of advertising sets to enable or disable
- **ext_adv_inst** –**[in]** : ext adv instance

返回 - ESP_OK : success

- other : failed

esp_err_t **esp_ble_gap_ext_adv_set_remove**(uint8_t instance)

This function is used to remove an advertising set from the Controller.

参数 **instance** –**[in]** : Used to identify an advertising set
返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_ext_adv_set_clear**(void)

This function is used to remove all existing advertising sets from the Controller.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_set_params**(uint8_t instance, const *esp_ble_gap_periodic_adv_params_t* *params)

This function is used by the Host to set the parameters for periodic advertising.

参数
- **instance** –**[in]** : identifies the advertising set whose periodic advertising parameters are being configured.
- **params** –**[in]** : periodic adv parameters

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_config_periodic_adv_data_raw**(uint8_t instance, uint16_t length, const uint8_t *data)

This function is used to set the data used in periodic advertising PDUs.

参数
- **instance** –**[in]** : identifies the advertising set whose periodic advertising parameters are being configured.
- **length** –**[in]** : the length of periodic data
- **data** –**[in]** : periodic data information

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_start**(uint8_t instance)

This function is used to request the Controller to enable the periodic advertising for the advertising set specified.

参数 **instance** –**[in]** : Used to identify an advertising set
返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_stop**(uint8_t instance)

This function is used to request the Controller to disable the periodic advertising for the advertising set specified.

参数 **instance** –**[in]** : Used to identify an advertising set
返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_ext_scan_params**(const *esp_ble_ext_scan_params_t* *params)

This function is used to set the extended scan parameters to be used on the advertising channels.

参数 **params** –**[in]** : scan parameters
返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_start_ext_scan**(uint32_t duration, uint16_t period)

This function is used to enable scanning.

参数
- **duration** –**[in]** : Scan duration

- **period** –**[in]** : Time interval from when the Controller started its last Scan Duration until it begins the subsequent Scan Duration.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_stop_ext_scan**(void)

This function is used to disable scanning.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_create_sync**(const *esp_ble_gap_periodic_adv_sync_params_t* *params)

This function is used to synchronize with periodic advertising from an advertiser and begin receiving periodic advertising packets.

参数 **params** –**[in]** : sync parameters
返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_sync_cancel**(void)

This function is used to cancel the LE_Periodic_Advertising_Create_Sync command while it is pending.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_sync_terminate**(uint16_t sync_handle)

This function is used to stop reception of the periodic advertising identified by the Sync Handle parameter.

参数 **sync_handle** –**[in]** : identify the periodic advertiser
返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_add_dev_to_list**(*esp_ble_addr_type_t* addr_type, *esp_bd_addr_t* addr, uint8_t sid)

This function is used to add a single device to the Periodic Advertiser list stored in the Controller.

参数
- **addr_type** –**[in]** : address type
- **addr** –**[in]** : Device Address
- **sid** –**[in]** : Advertising SID subfield in the ADI field used to identify the Periodic Advertising

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_remove_dev_from_list**(*esp_ble_addr_type_t* addr_type, *esp_bd_addr_t* addr, uint8_t sid)

This function is used to remove one device from the list of Periodic Advertisers stored in the Controller. Removals from the Periodic Advertisers List take effect immediately.

参数
- **addr_type** –**[in]** : address type
- **addr** –**[in]** : Device Address
- **sid** –**[in]** : Advertising SID subfield in the ADI field used to identify the Periodic Advertising

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_periodic_adv_clear_dev**(void)

This function is used to remove all devices from the list of Periodic Advertisers in the Controller.

返回 - ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_prefer_ext_connect_params_set**(*esp_bd_addr_t* addr, *esp_ble_gap_phy_mask_t* phy_mask, const *esp_ble_gap_conn_params_t* *phy_1m_conn_params, const *esp_ble_gap_conn_params_t* *phy_2m_conn_params, const *esp_ble_gap_conn_params_t* *phy_coded_conn_params)

This function is used to set aux connection parameters.

参数

- **addr** –**[in]** : device address
- **phy_mask** –**[in]** : indicates the PHY(s) on which the advertising packets should be received on the primary advertising channel and the PHYs for which connection parameters have been specified.
- **phy_1m_conn_params** –**[in]** : Scan connectable advertisements on the LE 1M PHY. Connection parameters for the LE 1M PHY are provided.
- **phy_2m_conn_params** –**[in]** : Connection parameters for the LE 2M PHY are provided.
- **phy_coded_conn_params** –**[in]** : Scan connectable advertisements on the LE Coded PHY. Connection parameters for the LE Coded PHY are provided.

返回 - ESP_OK : success

- other : failed

Unions

union **esp_ble_key_value_t**

#include <esp_gap_ble_api.h> union type of the security key value

Public Members

esp_ble_penc_keys_t **penc_key**

received peer encryption key

esp_ble_pcsrk_keys_t **pcsrk_key**

received peer device SRK

esp_ble_pid_keys_t **pid_key**

peer device ID key

esp_ble_lenc_keys_t **lenc_key**

local encryption reproduction keys LTK = = d1(ER,DIV,0)

esp_ble_lcsrk_keys **lcsrk_key**

local device CSRK = d1(ER,DIV,1)

union **esp_ble_sec_t**

#include <esp_gap_ble_api.h> union associated with ble security

Public Members

esp_ble_sec_key_notif_t **key_notif**

passkey notification

esp_ble_sec_req_t **ble_req**

BLE SMP related request

esp_ble_key_t **ble_key**

BLE SMP keys used when pairing

esp_ble_local_id_keys_t **ble_id_keys**

BLE IR event

esp_ble_local_oob_data_t **oob_data**

BLE SMP secure connection OOB data

esp_ble_auth_cmpl_t **auth_cmpl**

Authentication complete indication.

union **esp_ble_gap_cb_param_t**

#include <esp_gap_ble_api.h> Gap callback parameters union.

Public Members

struct *esp_ble_gap_cb_param_t*::*ble_get_dev_name_cmpl_evt_param* **get_dev_name_cmpl**

Event parameter of ESP_GAP_BLE_GET_DEV_NAME_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_adv_data_cmpl_evt_param* **adv_data_cmpl**

Event parameter of ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_rsp_data_cmpl_evt_param* **scan_rsp_data_cmpl**

Event parameter of ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_param_cmpl_evt_param* **scan_param_cmpl**

Event parameter of ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_result_evt_param* **scan_rst**

Event parameter of ESP_GAP_BLE_SCAN_RESULT_EVT

struct *esp_ble_gap_cb_param_t*::*ble_adv_data_raw_cmpl_evt_param* **adv_data_raw_cmpl**

Event parameter of ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_rsp_data_raw_cmpl_evt_param* **scan_rsp_data_raw_cmpl**

Event parameter of ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_adv_start_cmpl_evt_param* **adv_start_cmpl**

Event parameter of ESP_GAP_BLE_ADV_START_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_start_cmpl_evt_param* **scan_start_cmpl**

Event parameter of ESP_GAP_BLE_SCAN_START_COMPLETE_EVT

esp_ble_sec_t **ble_security**

ble gap security union type

struct *esp_ble_gap_cb_param_t*::*ble_scan_stop_cmpl_evt_param* **scan_stop_cmpl**

Event parameter of ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_adv_stop_cmpl_evt_param* **adv_stop_cmpl**

Event parameter of ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_set_rand_cmpl_evt_param* **set_rand_addr_cmpl**

Event parameter of ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT

struct *esp_ble_gap_cb_param_t*::*ble_update_conn_params_evt_param* **update_conn_params**

Event parameter of ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT

struct *esp_ble_gap_cb_param_t*::*ble_pkt_data_length_cmpl_evt_param* **pkt_data_length_cmpl**

Event parameter of ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_local_privacy_cmpl_evt_param* **local_privacy_cmpl**

Event parameter of ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_remove_bond_dev_cmpl_evt_param* **remove_bond_dev_cmpl**

Event parameter of ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_clear_bond_dev_cmpl_evt_param* **clear_bond_dev_cmpl**

Event parameter of ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_get_bond_dev_cmpl_evt_param* **get_bond_dev_cmpl**

Event parameter of ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_read_rssi_cmpl_evt_param* **read_rssi_cmpl**

Event parameter of ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_update_whitelist_cmpl_evt_param* **update_whitelist_cmpl**

Event parameter of ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_update_duplicate_exceptional_list_cmpl_evt_param* **update_duplicate_exceptional_list_cmpl**

Event parameter of ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_set_channels_evt_param* **ble_set_channels**

Event parameter of ESP_GAP_BLE_SET_CHANNELS_EVT

struct *esp_ble_gap_cb_param_t*::*ble_read_phy_cmpl_evt_param* **read_phy**

Event parameter of ESP_GAP_BLE_READ_PHY_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_set_perf_def_phy_cmpl_evt_param* **set_perf_def_phy**

Event parameter of ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_set_perf_phy_cmpl_evt_param* **set_perf_phy**

Event parameter of ESP_GAP_BLE_SET_PREFERRED_PHY_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_set_rand_addr_cmpl_evt_param* **ext_adv_set_rand_addr**

Event parameter of ESP_GAP_BLE_EXT_ADV_SET_RAND_ADDR_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_set_params_cmpl_evt_param* **ext_adv_set_params**

Event parameter of ESP_GAP_BLE_EXT_ADV_SET_PARAMS_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_data_set_cmpl_evt_param* **ext_adv_data_set**

Event parameter of ESP_GAP_BLE_EXT_ADV_DATA_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_scan_rsp_set_cmpl_evt_param* **scan_rsp_set**

Event parameter of ESP_GAP_BLE_EXT_SCAN_RSP_DATA_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_start_cmpl_evt_param* **ext_adv_start**

Event parameter of ESP_GAP_BLE_EXT_ADV_START_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_stop_cmpl_evt_param* **ext_adv_stop**

Event parameter of ESP_GAP_BLE_EXT_ADV_STOP_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_set_remove_cmpl_evt_param* **ext_adv_remove**

Event parameter of ESP_GAP_BLE_EXT_ADV_SET_REMOVE_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_set_clear_cmpl_evt_param* **ext_adv_clear**

Event parameter of ESP_GAP_BLE_EXT_ADV_SET_CLEAR_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_periodic_adv_set_params_cmpl_param* **peroid_adv_set_params**

Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SET_PARAMS_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_periodic_adv_data_set_cmpl_param* **period_adv_data_set**

Event parameter of ESP_GAP_BLE_PERIODIC_ADV_DATA_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_periodic_adv_start_cmpl_param* **period_adv_start**

Event parameter of ESP_GAP_BLE_PERIODIC_ADV_START_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_periodic_adv_stop_cmpl_param* **period_adv_stop**

Event parameter of ESP_GAP_BLE_PERIODIC_ADV_STOP_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_period_adv_create_sync_cmpl_param* **period_adv_create_sync**

Event parameter of ESP_GAP_BLE_PERIODIC_ADV_CREATE_SYNC_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_period_adv_sync_cancel_cmpl_param* **period_adv_sync_cancel**

Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SYNC_CANCEL_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_period_adv_sync_terminate_cmpl_param* **period_adv_sync_term**
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SYNC_TERMINATE_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_period_adv_add_dev_cmpl_param* **period_adv_add_dev**
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_ADD_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_period_adv_remove_dev_cmpl_param* **period_adv_remove_dev**
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_REMOVE_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_period_adv_clear_dev_cmpl_param* **period_adv_clear_dev**
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_CLEAR_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_set_ext_scan_params_cmpl_param* **set_ext_scan_params**
Event parameter of ESP_GAP_BLE_SET_EXT_SCAN_PARAMS_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_scan_start_cmpl_param* **ext_scan_start**
Event parameter of ESP_GAP_BLE_EXT_SCAN_START_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_scan_stop_cmpl_param* **ext_scan_stop**
Event parameter of ESP_GAP_BLE_EXT_SCAN_STOP_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_conn_params_set_cmpl_param* **ext_conn_params_set**
Event parameter of ESP_GAP_BLE_PREFER_EXT_CONN_PARAMS_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_adv_terminate_param* **adv_terminate**
Event parameter of ESP_GAP_BLE_ADV_TERMINATED_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_req_received_param* **scan_req_received**
Event parameter of ESP_GAP_BLE_SCAN_REQ_RECEIVED_EVT

struct *esp_ble_gap_cb_param_t*::*ble_channel_sel_alg_param* **channel_sel_alg**
Event parameter of ESP_GAP_BLE_CHANNEL_SELECT_ALGORITHM_EVT

struct *esp_ble_gap_cb_param_t*::*ble_periodic_adv_sync_lost_param* **periodic_adv_sync_lost**
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SYNC_LOST_EVT

struct *esp_ble_gap_cb_param_t*::*ble_periodic_adv_sync_estab_param* **periodic_adv_sync_estab**
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SYNC_ESTAB_EVT

struct *esp_ble_gap_cb_param_t*::*ble_phy_update_cmpl_param* **phy_update**
Event parameter of ESP_GAP_BLE_PHY_UPDATE_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_ext_adv_report_param* **ext_adv_report**
Event parameter of ESP_GAP_BLE_EXT_ADV_REPORT_EVT

struct *esp_ble_gap_cb_param_t*::*ble_periodic_adv_report_param* **period_adv_report**
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_REPORT_EVT

struct **ble_adv_data_cmpl_evt_param**
#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set advertising data operation success status

struct **ble_adv_data_raw_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set raw advertising data operation success status

struct **ble_adv_start_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_START_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate advertising start operation success status

struct **ble_adv_stop_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate adv stop operation success status

struct **ble_adv_terminate_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_TERMINATED_EVT.

Public Members

uint8_t **status**

Indicate adv terminate status

uint8_t **adv_instance**

extend advertising handle

uint16_t **conn_idx**

connection index

uint8_t **completed_event**

the number of completed extend advertising events

struct **ble_channel_sel_alg_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_CHANNEL_SELECT_ALGORITHM_EVT.

Public Members

uint16_t **conn_handle**

connection handle

uint8_t **channel_sel_alg**

channel selection algorithm

struct **ble_clear_bond_dev_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the clear bond device operation success status

struct **ble_ext_adv_data_set_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_DATA_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate extend advertising data set status

struct **ble_ext_adv_report_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_REPORT_EVT.

Public Members

esp_ble_gap_ext_adv_reprot_t **params**

extend advertising report parameters

struct **ble_ext_adv_scan_rsp_set_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_SCAN_RSP_DATA_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate extend advertising scan response data set status

struct **ble_ext_adv_set_clear_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_SET_CLEAR_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate advertising stop operation success status

struct **ble_ext_adv_set_params_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_SET_PARAMS_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate extend advertising parameters set status

struct **ble_ext_adv_set_rand_addr_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_SET_RAND_ADDR_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate extend advertising random address set status

struct **ble_ext_adv_set_remove_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_SET_REMOVE_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate advertising stop operation success status

struct **ble_ext_adv_start_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_START_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate advertising start operation success status

struct **ble_ext_adv_stop_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_STOP_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate advertising stop operation success status

struct **ble_ext_conn_params_set_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PREFER_EXT_CONN_PARAMS_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate extend connection parameters set status

struct **ble_ext_scan_start_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_SCAN_START_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate extend advertising start status

struct **ble_ext_scan_stop_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_SCAN_STOP_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate extend advertising stop status

struct **ble_get_bond_dev_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the get bond device operation success status

uint8_t **dev_num**

Indicate the get number device in the bond list

esp_ble_bond_dev_t ***bond_dev**

the pointer to the bond device Structure

struct **ble_get_dev_name_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_GET_DEV_NAME_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the get device name success status

char *name

Name of bluetooth device

struct **ble_local_privacy_cmpl_evt_param**

#include <*esp_gap_ble_api.h*> ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set local privacy operation success status

struct **ble_period_adv_add_dev_cmpl_param**

#include <*esp_gap_ble_api.h*> ESP_GAP_BLE_PERIODIC_ADV_ADD_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising device list add status

struct **ble_period_adv_clear_dev_cmpl_param**

#include <*esp_gap_ble_api.h*> ESP_GAP_BLE_PERIODIC_ADV_CLEAR_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising device list clean status

struct **ble_period_adv_create_sync_cmpl_param**

#include <*esp_gap_ble_api.h*> ESP_GAP_BLE_PERIODIC_ADV_CREATE_SYNC_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising create sync status

struct **ble_period_adv_remove_dev_cmpl_param**

#include <*esp_gap_ble_api.h*> ESP_GAP_BLE_PERIODIC_ADV_REMOVE_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising device list remove status

struct **ble_period_adv_sync_cancel_cmpl_param**

#include <*esp_gap_ble_api.h*> ESP_GAP_BLE_PERIODIC_ADV_SYNC_CANCEL_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising sync cancel status

struct **ble_period_adv_sync_terminate_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_SYNC_TERMINATE_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising sync terminate status

struct **ble_periodic_adv_data_set_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_DATA_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising data set status

struct **ble_periodic_adv_report_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_REPORT_EVT.

Public Members

esp_ble_gap_periodic_adv_report_t **params**

periodic advertising report parameters

struct **ble_periodic_adv_set_params_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_SET_PARAMS_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertisingparameters set status

struct **ble_periodic_adv_start_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_START_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising start status

struct **ble_periodic_adv_stop_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_STOP_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate periodic advertising stop status

struct **ble_periodic_adv_sync_estab_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_SYNC_ESTAB_EVT.

Public Members

uint8_t **status**

periodic advertising sync status

uint16_t **sync_handle**

periodic advertising sync handle

uint8_t **sid**

periodic advertising sid

esp_ble_addr_type_t **adv_addr_type**

periodic advertising address type

esp_bd_addr_t **adv_addr**

periodic advertising address

esp_ble_gap_phy_t **adv_phy**

periodic advertising phy type

uint16_t **period_adv_interval**

periodic advertising interval

uint8_t **adv_clk_accuracy**

periodic advertising clock accuracy

struct **ble_periodic_adv_sync_lost_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_SYNC_LOST_EVT.

Public Members

uint16_t **sync_handle**

sync handle

struct **ble_phy_update_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_PHY_UPDATE_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

phy update status

esp_bd_addr_t **bda**

address

esp_ble_gap_phy_t **tx_phy**

tx phy type

esp_ble_gap_phy_t **rx_phy**

rx phy type

struct **ble_pkt_data_length_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set pkt data length operation success status

esp_ble_pkt_data_length_params_t **params**

pkt data length value

struct **ble_read_phy_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_READ_PHY_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

read phy complete status

esp_bd_addr_t **bda**

read phy address

esp_ble_gap_phy_t **tx_phy**

tx phy type

esp_ble_gap_phy_t **rx_phy**

rx phy type

struct **ble_read_rssi_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the read adv tx power operation success status

int8_t **rssi**

The ble remote device rssi value, the range is from -127 to 20, the unit is dbm, if the RSSI cannot be read, the RSSI metric shall be set to 127.

esp_bd_addr_t **remote_addr**

The remote device address

struct **ble_remove_bond_dev_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the remove bond device operation success status

esp_bd_addr_t **bd_addr**

The device address which has been remove from the bond list

struct **ble_scan_param_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set scan param operation success status

struct **ble_scan_req_received_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_REQ_RECEIVED_EVT.

Public Members

uint8_t **adv_instance**

extend advertising handle

esp_ble_addr_type_t **scan_addr_type**

scanner address type

esp_bd_addr_t **scan_addr**

scanner address

struct **ble_scan_result_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RESULT_EVT.

Public Members

esp_gap_search_evt_t **search_evt**

Search event type

esp_bd_addr_t **bda**

Bluetooth device address which has been searched

esp_bt_dev_type_t **dev_type**

Device type

esp_ble_addr_type_t **ble_addr_type**

Ble device address type

esp_ble_evt_type_t **ble_evt_type**

Ble scan result event type

int **rssi**

Searched device ' s RSSI

uint8_t **ble_adv**[ESP_BLE_ADV_DATA_LEN_MAX + ESP_BLE_SCAN_RSP_DATA_LEN_MAX]

Received EIR

int **flag**

Advertising data flag bit

int **num_resps**

Scan result number

uint8_t **adv_data_len**

Adv data length

uint8_t **scan_rsp_len**

Scan response length

uint32_t **num_dis**

The number of discard packets

struct **ble_scan_rsp_data_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set scan response data operation success status

struct **ble_scan_rsp_data_raw_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set raw advertising data operation success status

struct **ble_scan_start_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_START_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate scan start operation success status

struct **ble_scan_stop_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate scan stop operation success status

struct **ble_set_channels_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_CHANNELS_EVT.

Public Members

esp_bt_status_t **stat**

BLE set channel status

struct **ble_set_ext_scan_params_cmpl_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_EXT_SCAN_PARAMS_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate extend advertising parameters set status

struct **ble_set_perf_def_phy_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate perf default phy set status

struct **ble_set_perf_phy_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_PREFERRED_PHY_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate perf phy set status

struct **ble_set_rand_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT.

Public Members

esp_bt_status_t **status**

Indicate set static rand address operation success status

struct **ble_update_conn_params_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT.

Public Members

esp_bt_status_t **status**

Indicate update connection parameters success status

esp_bd_addr_t **bda**

Bluetooth device address

uint16_t **min_int**

Min connection interval

uint16_t **max_int**

Max connection interval

uint16_t **latency**

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t **conn_int**

Current connection interval

uint16_t **timeout**

Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec

struct **ble_update_duplicate_exceptional_list_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate update duplicate scan exceptional list operation success status

uint8_t **subcode**

Define in esp_bt_duplicate_exceptional_subcode_type_t

uint16_t **length**

The length of device_info

esp_duplicate_info_t **device_info**

device information, when subcode is ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_CLEAN, the value is invalid

struct **ble_update_whitelist_cmpl_evt_param**

#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the add or remove whitelist operation success status

esp_ble_wl_operation_t **wl_operation**

The value is ESP_BLE_WHITELIST_ADD if add address to whitelist operation success, ESP_BLE_WHITELIST_REMOVE if remove address from the whitelist operation success

Structures

struct **esp_ble_adv_params_t**

Advertising parameters.

Public Members

uint16_t **adv_int_min**

Minimum advertising interval for undirected and low duty cycle directed advertising. Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec

uint16_t **adv_int_max**

Maximum advertising interval for undirected and low duty cycle directed advertising. Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec Advertising max interval

esp_ble_adv_type_t **adv_type**

Advertising type

esp_ble_addr_type_t **own_addr_type**

Owner bluetooth device address type

esp_bd_addr_t **peer_addr**

Peer device bluetooth device address

esp_ble_addr_type_t **peer_addr_type**

Peer device bluetooth device address type, only support public address type and random address type

esp_ble_adv_channel_t **channel_map**

Advertising channel map

esp_ble_adv_filter_t **adv_filter_policy**

Advertising filter policy

struct **esp_ble_adv_data_t**

Advertising data content, according to "Supplement to the Bluetooth Core Specification".

Public Members

bool **set_scan_rsp**

Set this advertising data as scan response or not

bool **include_name**

Advertising data include device name or not

bool **include_txpower**

Advertising data include TX power

int **min_interval**

Advertising data show slave preferred connection min interval. The connection interval in the following manner: connIntervalmin = Conn_Interval_Min * 1.25 ms Conn_Interval_Min range: 0x0006 to 0x0C80 Value of 0xFFFF indicates no specific minimum. Values not defined above are reserved for future use.

int **max_interval**

Advertising data show slave preferred connection max interval. The connection interval in the following manner: connIntervalmax = Conn_Interval_Max * 1.25 ms Conn_Interval_Max range: 0x0006 to 0x0C80 Conn_Interval_Max shall be equal to or greater than the Conn_Interval_Min. Value of 0xFFFF indicates no specific maximum. Values not defined above are reserved for future use.

int **appearance**

External appearance of device

uint16_t **manufacturer_len**

Manufacturer data length

uint8_t ***p_manufacturer_data**

Manufacturer data point

uint16_t **service_data_len**

Service data length

uint8_t ***p_service_data**

Service data point

uint16_t **service_uuid_len**

Service uuid length

uint8_t ***p_service_uuid**

Service uuid array point

uint8_t **flag**

Advertising flag of discovery mode, see BLE_ADV_DATA_FLAG detail

struct **esp_ble_scan_params_t**

Ble scan parameters.

Public Members

esp_ble_scan_type_t **scan_type**

Scan type

esp_ble_addr_type_t **own_addr_type**

Owner address type

esp_ble_scan_filter_t **scan_filter_policy**

Scan filter policy

uint16_t **scan_interval**

Scan interval. This is defined as the time interval from when the Controller started its last LE scan until it begins the subsequent LE scan. Range: 0x0004 to 0x4000 Default: 0x0010 (10 ms) Time = N * 0.625 msec Time Range: 2.5 msec to 10.24 seconds

uint16_t **scan_window**

Scan window. The duration of the LE scan. LE_Scan_Window shall be less than or equal to LE_Scan_Interval Range: 0x0004 to 0x4000 Default: 0x0010 (10 ms) Time = N * 0.625 msec Time Range: 2.5 msec to 10240 msec

esp_ble_scan_duplicate_t **scan_duplicate**

The Scan_Duplicates parameter controls whether the Link Layer should filter out duplicate advertising reports (BLE_SCAN_DUPLICATE_ENABLE) to the Host, or if the Link Layer should generate advertising reports for each packet received

struct **esp_gap_conn_params_t**

connection parameters information

Public Members

uint16_t **interval**

connection interval

uint16_t **latency**

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t **timeout**

Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds

struct **esp_ble_conn_update_params_t**

Connection update parameters.

Public Members

esp_bd_addr_t **bda**

Bluetooth device address

uint16_t **min_int**

Min connection interval

uint16_t **max_int**

Max connection interval

uint16_t **latency**

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t **timeout**

Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds

struct **esp_ble_pkt_data_length_params_t**

BLE pkt date length keys.

Public Members

uint16_t **rx_len**

pkt rx data length value

uint16_t **tx_len**

pkt tx data length value

struct **esp_ble_penc_keys_t**

BLE encryption keys.

Public Members

esp_bt_octet16_t **ltk**

The long term key

esp_bt_octet8_t **rand**

The random number

uint16_t **ediv**

The ediv value

uint8_t **sec_level**

The security level of the security link

uint8_t **key_size**

The key size(7~16) of the security link

struct **esp_ble_pcsrk_keys_t**

BLE CSRK keys.

Public Members

uint32_t **counter**

The counter

esp_bt_octet16_t **csrk**

The csrk key

uint8_t **sec_level**

The security level

struct **esp_ble_pid_keys_t**

BLE pid keys.

Public Members

esp_bt_octet16_t **irk**

The irk value

esp_ble_addr_type_t **addr_type**

The address type

esp_bd_addr_t **static_addr**

The static address

struct **esp_ble_lenc_keys_t**

BLE Encryption reproduction keys.

Public Members

esp_bt_octet16_t **ltk**

The long term key

uint16_t **div**

The div value

uint8_t **key_size**

The key size of the security link

uint8_t **sec_level**

The security level of the security link

struct **esp_ble_lcsrk_keys**

BLE SRK keys.

Public Members

uint32_t **counter**

The counter value

uint16_t **div**

The div value

uint8_t **sec_level**

The security level of the security link

esp_bt_octet16_t **csrk**

The csrk key value

struct **esp_ble_sec_key_notif_t**

Structure associated with ESP_KEY_NOTIF_EVT.

Public Members

esp_bd_addr_t **bd_addr**

peer address

uint32_t **passkey**

the numeric value for comparison. If just_works, do not show this number to UI

struct **esp_ble_sec_req_t**

Structure of the security request.

Public Members

esp_bd_addr_t **bd_addr**
peer address

struct **esp_ble_bond_key_info_t**
struct type of the bond key information value

Public Members

esp_ble_key_mask_t **key_mask**
the key mask to indicate witch key is present

esp_ble_penc_keys_t **penc_key**
received peer encryption key

esp_ble_pcsrk_keys_t **pcsrk_key**
received peer device SRK

esp_ble_pid_keys_t **pid_key**
peer device ID key

struct **esp_ble_bond_dev_t**
struct type of the bond device value

Public Members

esp_bd_addr_t **bd_addr**
peer address

esp_ble_bond_key_info_t **bond_key**
the bond key information

struct **esp_ble_key_t**
union type of the security key value

Public Members

esp_bd_addr_t **bd_addr**
peer address

esp_ble_key_type_t **key_type**
key type of the security link

esp_ble_key_value_t **p_key_value**
the pointer to the key value

struct **esp_ble_local_id_keys_t**
structure type of the ble local id keys value

Public Members

esp_bt_octet16_t **ir**
the 16 bits of the ir value

esp_bt_octet16_t **irk**
the 16 bits of the ir key value

esp_bt_octet16_t **dhk**
the 16 bits of the dh key value

struct **esp_ble_local_oob_data_t**
structure type of the ble local oob data value

Public Members

esp_bt_octet16_t **oob_c**
the 128 bits of confirmation value

esp_bt_octet16_t **oob_r**
the 128 bits of randomizer value

struct **esp_ble_auth_cmpl_t**
Structure associated with ESP_AUTH_CMPL_EVT.

Public Members

esp_bd_addr_t **bd_addr**
BD address peer device.

bool **key_present**
Valid link key value in key element

esp_link_key **key**
Link key associated with peer device.

uint8_t **key_type**
The type of Link Key

bool **success**
TRUE of authentication succeeded, FALSE if failed.

uint8_t **fail_reason**
The HCI reason/error code for when success=FALSE

esp_ble_addr_type_t **addr_type**
Peer device address type

esp_bt_dev_type_t **dev_type**

Device type

esp_ble_auth_req_t **auth_mode**

authentication mode

struct **esp_ble_gap_ext_adv_params_t**

ext adv parameters

Public Members

esp_ble_ext_adv_type_mask_t **type**

ext adv type

uint32_t **interval_min**

ext adv minimum interval

uint32_t **interval_max**

ext adv maximum interval

esp_ble_adv_channel_t **channel_map**

ext adv channel map

esp_ble_addr_type_t **own_addr_type**

ext adv own address type

esp_ble_addr_type_t **peer_addr_type**

ext adv peer address type

esp_bd_addr_t **peer_addr**

ext adv peer address

esp_ble_adv_filter_t **filter_policy**

ext adv filter policy

int8_t **tx_power**

ext adv tx power

esp_ble_gap_pri_phy_t **primary_phy**

ext adv primary phy

uint8_t **max_skip**

ext adv maximum skip

esp_ble_gap_phy_t **secondary_phy**

ext adv secondary phy

uint8_t **sid**

ext adv sid

bool **scan_req_notif**

ext adv scan request event notify

struct **esp_ble_ext_scan_cfg_t**

ext scan config

Public Members

esp_ble_scan_type_t **scan_type**

ext scan type

uint16_t **scan_interval**

ext scan interval

uint16_t **scan_window**

ext scan window

struct **esp_ble_ext_scan_params_t**

ext scan parameters

Public Members

esp_ble_addr_type_t **own_addr_type**

ext scan own address type

esp_ble_scan_filter_t **filter_policy**

ext scan filter policy

esp_ble_scan_duplicate_t **scan_duplicate**

ext scan duplicate scan

esp_ble_ext_scan_cfg_mask_t **cfg_mask**

ext scan config mask

esp_ble_ext_scan_cfg_t **uncoded_cfg**

ext scan uncoded config parameters

esp_ble_ext_scan_cfg_t **coded_cfg**

ext scan coded config parameters

struct **esp_ble_gap_conn_params_t**

create extend connection parameters

Public Members

uint16_t **scan_interval**
: init scan interval

uint16_t **scan_window**
: init scan window

uint16_t **interval_min**
: minimum interval

uint16_t **interval_max**
: maximum interval

uint16_t **latency**
: ext scan type

uint16_t **supervision_timeout**
: connection supervision timeout

uint16_t **min_ce_len**
: minimum ce length

uint16_t **max_ce_len**
: maximum ce length

struct **esp_ble_gap_ext_adv_t**
: extend adv enable parameters

Public Members

uint8_t **instance**
: advertising handle

int **duration**
: advertising duration

int **max_events**
: maximum number of extended advertising events

struct **esp_ble_gap_periodic_adv_params_t**
: periodic adv parameters

Public Members

uint16_t **interval_min**
: periodic advertising minimum interval

uint16_t **interval_max**

periodic advertising maximum interval

uint8_t **properties**

periodic advertising properties

struct **esp_ble_gap_periodic_adv_sync_params_t**

periodic adv sync parameters

Public Members

esp_ble_gap_sync_t **filter_policy**

periodic advertising sync filter policy

uint8_t **sid**

periodic advertising sid

esp_ble_addr_type_t **addr_type**

periodic advertising address type

esp_bd_addr_t **addr**

periodic advertising address

uint16_t **skip**

the maximum number of periodic advertising events that can be skipped

uint16_t **sync_timeout**

synchronization timeout

struct **esp_ble_gap_ext_adv_reprot_t**

extend adv report parameters

Public Members

esp_ble_gap_adv_type_t **event_type**

extend advertising type

uint8_t **addr_type**

extend advertising address type

esp_bd_addr_t **addr**

extend advertising address

esp_ble_gap_pri_phy_t **primary_phy**

extend advertising primary phy

esp_ble_gap_phy_t **secondly_phy**
extend advertising secondary phy

uint8_t **sid**
extend advertising sid

uint8_t **tx_power**
extend advertising tx power

int8_t **rssi**
extend advertising rssi

uint16_t **per_adv_interval**
periodic advertising interval

uint8_t **dir_addr_type**
direct address type

esp_bd_addr_t **dir_addr**
direct address

esp_ble_gap_ext_adv_data_status_t **data_status**
data type

uint8_t **adv_data_len**
extend advertising data length

uint8_t **adv_data**[251]
extend advertising data

struct **esp_ble_gap_periodic_adv_report_t**
periodic adv report parameters

Public Members

uint16_t **sync_handle**
periodic advertising train handle

uint8_t **tx_power**
periodic advertising tx power

int8_t **rssi**
periodic advertising rssi

esp_ble_gap_ext_adv_data_status_t **data_status**
periodic advertising data type

uint8_t **data_length**

periodic advertising data length

uint8_t **data**[251]

periodic advertising data

struct **esp_ble_gap_periodic_adv_sync_estab_t**

perodic adv sync establish parameters

Public Members

uint8_t **status**

periodic advertising sync status

uint16_t **sync_handle**

periodic advertising train handle

uint8_t **sid**

periodic advertising sid

esp_ble_addr_type_t **addr_type**

periodic advertising address type

esp_bd_addr_t **adv_addr**

periodic advertising address

esp_ble_gap_phy_t **adv_phy**

periodic advertising adv phy type

uint16_t **period_adv_interval**

periodic advertising interval

uint8_t **adv_clk_accuracy**

periodic advertising clock accuracy

Macros

ESP_BLE_ADV_FLAG_LIMIT_DISC

BLE_ADV_DATA_FLAG data flag bit definition used for advertising data flag.

ESP_BLE_ADV_FLAG_GEN_DISC

ESP_BLE_ADV_FLAG_BREDR_NOT_SPT

ESP_BLE_ADV_FLAG_DMT_CONTROLLER_SPT

ESP_BLE_ADV_FLAG_DMT_HOST_SPT

ESP_BLE_ADV_FLAG_NON_LIMIT_DISC

ESP_LE_KEY_NONE

relate to BTM_LE_KEY_xxx in stack/btm_api.h

No encryption key

ESP_LE_KEY_PENC

encryption key, encryption information of peer device

ESP_LE_KEY_PID

identity key of the peer device

ESP_LE_KEY_PCSRK

peer SRK

ESP_LE_KEY_PLK

Link key

ESP_LE_KEY_LLK

peer link key

ESP_LE_KEY_LENC

master role security information:div

ESP_LE_KEY_LID

master device ID key

ESP_LE_KEY_LCSRK

local CSRK has been deliver to peer

ESP_LE_AUTH_NO_BOND

relate to BTM_LE_AUTH_xxx in stack/btm_api.h

0 no bondingv

ESP_LE_AUTH_BOND

1 « 0 device in the bonding with peer

ESP_LE_AUTH_REQ_MITM

1 « 2 man in the middle attack

ESP_LE_AUTH_REQ_BOND_MITM

0101 banding with man in the middle attack

ESP_LE_AUTH_REQ_SC_ONLY

1 « 3 secure connection

ESP_LE_AUTH_REQ_SC_BOND

1001 secure connection with band

ESP_LE_AUTH_REQ_SC_MITM

1100 secure conn with MITM

ESP_LE_AUTH_REQ_SC_MITM_BOND

1101 SC with MITM and Bonding

ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_DISABLE

authentication disable

ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_ENABLE

authentication enable

ESP_BLE_OOB_DISABLE

disbale the out of bond

ESP_BLE_OOB_ENABLE

enable the out of bond

ESP_IO_CAP_OUT

relate to BTM_IO_CAP_xxx in stack/btm_api.h

DisplayOnly

ESP_IO_CAP_IO

DisplayYesNo

ESP_IO_CAP_IN

KeyboardOnly

ESP_IO_CAP_NONE

NoInputNoOutput

ESP_IO_CAP_KBDISP

Keyboard display

ESP_BLE_APPEARANCE_UNKNOWN

relate to BTM_BLE_APPEARANCE_UNKNOWN in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_PHONE

relate to BTM_BLE_APPEARANCE_GENERIC_PHONE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_COMPUTER

relate to BTM_BLE_APPEARANCE_GENERIC_COMPUTER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_WATCH

relate to BTM_BLE_APPEARANCE_GENERIC_WATCH in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_SPORTS_WATCH

relate to BTM_BLE_APPEARANCE_SPORTS_WATCH in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_CLOCK

relate to BTM_BLE_APPEARANCE_GENERIC_CLOCK in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_DISPLAY

relate to BTM_BLE_APPEARANCE_GENERIC_DISPLAY in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_REMOTE

relate to BTM_BLE_APPEARANCE_GENERIC_REMOTE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_EYEGLASSES

relate to BTM_BLE_APPEARANCE_GENERIC_EYEGLASSES in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_TAG

relate to BTM_BLE_APPEARANCE_GENERIC_TAG in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_KEYRING

relate to BTM_BLE_APPEARANCE_GENERIC_KEYRING in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_MEDIA_PLAYER

relate to BTM_BLE_APPEARANCE_GENERIC_MEDIA_PLAYER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_BARCODE_SCANNER

relate to BTM_BLE_APPEARANCE_GENERIC_BARCODE_SCANNER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_THERMOMETER

relate to BTM_BLE_APPEARANCE_GENERIC_THERMOMETER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_THERMOMETER_EAR

relate to BTM_BLE_APPEARANCE_THERMOMETER_EAR in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_HEART_RATE

relate to BTM_BLE_APPEARANCE_GENERIC_HEART_RATE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HEART_RATE_BELT

relate to BTM_BLE_APPEARANCE_HEART_RATE_BELT in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_BLOOD_PRESSURE

relate to BTM_BLE_APPEARANCE_GENERIC_BLOOD_PRESSURE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_BLOOD_PRESSURE_ARM

relate to BTM_BLE_APPEARANCE_BLOOD_PRESSURE_ARM in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_BLOOD_PRESSURE_WRIST

relate to BTM_BLE_APPEARANCE_BLOOD_PRESSURE_WRIST in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_HID

relate to BTM_BLE_APPEARANCE_GENERIC_HID in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_KEYBOARD

relate to BTM_BLE_APPEARANCE_HID_KEYBOARD in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_MOUSE

relate to BTM_BLE_APPEARANCE_HID_MOUSE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_JOYSTICK

relate to BTM_BLE_APPEARANCE_HID_JOYSTICK in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_GAMEPAD

relate to BTM_BLE_APPEARANCE_HID_GAMEPAD in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_DIGITIZER_TABLET

relate to BTM_BLE_APPEARANCE_HID_DIGITIZER_TABLET in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_CARD_READER

relate to BTM_BLE_APPEARANCE_HID_CARD_READER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_DIGITAL_PEN

relate to BTM_BLE_APPEARANCE_HID_DIGITAL_PEN in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_BARCODE_SCANNER

relate to BTM_BLE_APPEARANCE_HID_BARCODE_SCANNER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_GLUCOSE

relate to BTM_BLE_APPEARANCE_GENERIC_GLUCOSE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_WALKING

relate to BTM_BLE_APPEARANCE_GENERIC_WALKING in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_WALKING_IN_SHOE

relate to BTM_BLE_APPEARANCE_WALKING_IN_SHOE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_WALKING_ON_SHOE

relate to BTM_BLE_APPEARANCE_WALKING_ON_SHOE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_WALKING_ON_HIP

relate to BTM_BLE_APPEARANCE_WALKING_ON_HIP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_CYCLING

relate to BTM_BLE_APPEARANCE_GENERIC_CYCLING in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_COMPUTER

relate to BTM_BLE_APPEARANCE_CYCLING_COMPUTER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_SPEED

relate to BTM_BLE_APPEARANCE_CYCLING_SPEED in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_CADENCE

relate to BTM_BLE_APPEARANCE_CYCLING_CADENCE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_POWER

relate to BTM_BLE_APPEARANCE_CYCLING_POWER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_SPEED_CADENCE

relate to BTM_BLE_APPEARANCE_CYCLING_SPEED_CADENCE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_PULSE_OXIMETER

relate to BTM_BLE_APPEARANCE_GENERIC_PULSE_OXIMETER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_PULSE_OXIMETER_FINGERTIP

relate to BTM_BLE_APPEARANCE_PULSE_OXIMETER_FINGERTIP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_PULSE_OXIMETER_WRIST

relate to BTM_BLE_APPEARANCE_PULSE_OXIMETER_WRIST in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_WEIGHT

relate to BTM_BLE_APPEARANCE_GENERIC_WEIGHT in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_PERSONAL_MOBILITY_DEVICE

relate to BTM_BLE_APPEARANCE_GENERIC_PERSONAL_MOBILITY_DEVICE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_POWERED_WHEELCHAIR

relate to BTM_BLE_APPEARANCE_POWERED_WHEELCHAIR in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_MOBILITY_SCOOTER

relate to BTM_BLE_APPEARANCE_MOBILITY_SCOOTER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_CONTINUOUS_GLUCOSE_MONITOR

relate to BTM_BLE_APPEARANCE_GENERIC_CONTINUOUS_GLUCOSE_MONITOR in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_INSULIN_PUMP

relate to BTM_BLE_APPEARANCE_GENERIC_INSULIN_PUMP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_INSULIN_PUMP_DURABLE_PUMP

relate to BTM_BLE_APPEARANCE_INSULIN_PUMP_DURABLE_PUMP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_INSULIN_PUMP_PATCH_PUMP

relate to BTM_BLE_APPEARANCE_INSULIN_PUMP_PATCH_PUMP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_INSULIN_PEN

relate to BTM_BLE_APPEARANCE_INSULIN_PEN in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_MEDICATION_DELIVERY

relate to BTM_BLE_APPEARANCE_GENERIC_MEDICATION_DELIVERY in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_OUTDOOR_SPORTS

relate to BTM_BLE_APPEARANCE_GENERIC_OUTDOOR_SPORTS in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION

relate to BTM_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_AND_NAV

relate to BTM_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_AND_NAV in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD

relate to BTM_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD_AND_NAV

relate to BTM_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD_AND_NAV in stack/btm_ble_api.h

ESP_GAP_BLE_CHANNELS_LEN

channel length

ESP_GAP_BLE_ADD_WHITELIST_COMPLETE_EVT

This is the old name, just for backwards compatibility.

ESP_BLE_ADV_DATA_LEN_MAX

Advertising data maximum length.

ESP_BLE_SCAN_RSP_DATA_LEN_MAX

Scan response data maximum length.

BLE_BIT(n)

ESP_BLE_GAP_SET_EXT_ADV_PROP_NONCONN_NONSCANNABLE_UNDIRECTED

Non-Connectable and Non-Scannable Undirected advertising

ESP_BLE_GAP_SET_EXT_ADV_PROP_CONNECTABLE

Connectable advertising

ESP_BLE_GAP_SET_EXT_ADV_PROP_SCANNABLE

Scannable advertising

ESP_BLE_GAP_SET_EXT_ADV_PROP_DIRECTED

Directed advertising

ESP_BLE_GAP_SET_EXT_ADV_PROP_HD_DIRECTED

High Duty Cycle Directed Connectable advertising (<= 3.75 ms Advertising Interval)

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY

Use legacy advertising PDUs

ESP_BLE_GAP_SET_EXT_ADV_PROP_ANON_ADV

Omit advertiser’s address from all PDUs (“anonymous advertising”)

ESP_BLE_GAP_SET_EXT_ADV_PROP_INCLUDE_TX_PWR

Include TxPower in the extended header of the advertising PDU

ESP_BLE_GAP_SET_EXT_ADV_PROP_MASK

Reserved for future use If extended advertising PDU types are being used (bit 4 = 0) then: The advertisement shall not be both connectable and scannable. High duty cycle directed connectable advertising (<= 3.75 ms advertising interval) shall not be used (bit 3 = 0) ADV_IND

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_IND

ADV_DIRECT_IND (low duty cycle)

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_LD_DIR

ADV_DIRECT_IND (high duty cycle)

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_HD_DIR

ADV_SCAN_IND

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_SCAN

ADV_NONCONN_IND

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_NONCONN

ESP_BLE_GAP_PHY_1M

Secondery Advertisement PHY is LE1M

ESP_BLE_GAP_PHY_2M

Secondery Advertisement PHY is LE2M

ESP_BLE_GAP_PHY_CODED

Secondery Advertisement PHY is LE Coded

ESP_BLE_GAP_NO_PREFER_TRANSMIT_PHY

No Prefer TX PHY supported by controller

ESP_BLE_GAP_NO_PREFER_RECEIVE_PHY

No Prefer RX PHY supported by controller

ESP_BLE_GAP_PRI_PHY_1M

Primary phy only support 1M and LE coded phy.

Primary Phy is LE1M

ESP_BLE_GAP_PRI_PHY_CODED

Primary Phy is LE CODED

ESP_BLE_GAP_PHY_1M_PREF_MASK

The Host prefers use the LE1M transmitter or reciever PHY

ESP_BLE_GAP_PHY_2M_PREF_MASK

The Host prefers use the LE2M transmitter or reciever PHY

ESP_BLE_GAP_PHY_CODED_PREF_MASK

The Host prefers use the LE CODED transmitter or reciever PHY

ESP_BLE_GAP_PHY_OPTIONS_NO_PREF

The Host has no preferred coding when transmitting on the LE Coded PHY

ESP_BLE_GAP_PHY_OPTIONS_PREF_S2_CODING

The Host prefers that S=2 coding be used when transmitting on the LE Coded PHY

ESP_BLE_GAP_PHY_OPTIONS_PREF_S8_CODING

The Host prefers that S=8 coding be used when transmitting on the LE Coded PHY

ESP_BLE_GAP_EXT_SCAN_CFG_UNCODE_MASK

Scan Advertisements on the LE1M PHY

ESP_BLE_GAP_EXT_SCAN_CFG_CODE_MASK

Scan advertisements on the LE coded PHY

ESP_BLE_GAP_EXT_ADV_DATA_COMPLETE

Advertising data.

extended advertising data compete

ESP_BLE_GAP_EXT_ADV_DATA_INCOMPLETE

extended advertising data incomplete

ESP_BLE_GAP_EXT_ADV_DATA_TRUNCATED

extended advertising data truncated mode

ESP_BLE_GAP_SYNC_POLICY_BY_ADV_INFO

Advertising SYNC policy.

sync policy by advertising info

ESP_BLE_GAP_SYNC_POLICY_BY_PERIODIC_LIST

periodic advertising sync policy

ESP_BLE_ADV_REPORT_EXT_ADV_IND

Advertising report.

advertising report with extended advertising indication type

ESP_BLE_ADV_REPORT_EXT_SCAN_IND

advertising report with extended scan indication type

ESP_BLE_ADV_REPORT_EXT_DIRECT_ADV

advertising report with extended direct advertising indication type

ESP_BLE_ADV_REPORT_EXT_SCAN_RSP
advertising report with extended scan response indication type Bluetooth 5.0, Vol 2, Part E, 7.7.65.13

ESP_BLE_LEGACY_ADV_TYPE_IND
advertising report with legacy advertising indication type

ESP_BLE_LEGACY_ADV_TYPE_DIRECT_IND
advertising report with legacy direct indication type

ESP_BLE_LEGACY_ADV_TYPE_SCAN_IND
advertising report with legacy scan indication type

ESP_BLE_LEGACY_ADV_TYPE_NONCON_IND
advertising report with legacy non connectable indication type

ESP_BLE_LEGACY_ADV_TYPE_SCAN_RSP_TO_ADV_IND
advertising report with legacy scan response indication type

ESP_BLE_LEGACY_ADV_TYPE_SCAN_RSP_TO_ADV_SCAN_IND
advertising report with legacy advertising with scan response indication type

EXT_ADV_TX_PWR_NO_PREFERENCE
Extend advertising tx power, range: [-127, +126] dBm.

host has no preference for tx power

Type Definitions

typedef uint8_t **esp_ble_key_type_t**

typedef uint8_t **esp_ble_auth_req_t**
combination of the above bit pattern

typedef uint8_t **esp_ble_io_cap_t**
combination of the io capability

typedef uint8_t **esp_gap_ble_channels**[ESP_GAP_BLE_CHANNELS_LEN]

typedef uint8_t **esp_duplicate_info_t**[ESP_BD_ADDR_LEN]

typedef uint16_t **esp_ble_ext_adv_type_mask_t**

typedef uint8_t **esp_ble_gap_phy_t**

typedef uint8_t **esp_ble_gap_all_phys_t**

typedef uint8_t **esp_ble_gap_pri_phy_t**

typedef uint8_t **esp_ble_gap_phy_mask_t**

typedef uint16_t **esp_ble_gap_prefer_phy_options_t**

typedef uint8_t **esp_ble_ext_scan_cfg_mask_t**

typedef uint8_t **esp_ble_gap_ext_adv_data_status_t**

typedef uint8_t **esp_ble_gap_sync_t**

typedef uint8_t **esp_ble_gap_adv_type_t**

typedef void (***esp_gap_ble_cb_t**)(*esp_gap_ble_cb_event_t* event, *esp_ble_gap_cb_param_t* *param)

GAP callback function type.

Param event : Event type
Param param : Point to callback parameter, currently is union type

Enumerations

enum **esp_gap_ble_cb_event_t**

GAP BLE callback event type.

Values:

enumerator **ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT**

When advertising data set complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT**

When scan response data set complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT**

When scan parameters set complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_RESULT_EVT**

When one scan result ready, the event comes each time

enumerator **ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT**

When raw advertising data set complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT**

When raw advertising data set complete, the event comes

enumerator **ESP_GAP_BLE_ADV_START_COMPLETE_EVT**

When start advertising complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_START_COMPLETE_EVT**

When start scan complete, the event comes

enumerator **ESP_GAP_BLE_AUTH_CMPL_EVT**

Authentication complete indication.

enumerator **ESP_GAP_BLE_KEY_EVT**

BLE key event for peer device keys

enumerator **ESP_GAP_BLE_SEC_REQ_EVT**

BLE security request

enumerator **ESP_GAP_BLE_PASSKEY_NOTIF_EVT**

passkey notification event

enumerator **ESP_GAP_BLE_PASSKEY_REQ_EVT**

passkey request event

enumerator **ESP_GAP_BLE_OOB_REQ_EVT**

OOB request event

enumerator **ESP_GAP_BLE_LOCAL_IR_EVT**

BLE local IR (identity Root 128-bit random static value used to generate Long Term Key) event

enumerator **ESP_GAP_BLE_LOCAL_ER_EVT**

BLE local ER (Encryption Root vakue used to genrate identity resolving key) event

enumerator **ESP_GAP_BLE_NC_REQ_EVT**

Numeric Comparison request event

enumerator **ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT**

When stop adv complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT**

When stop scan complete, the event comes

enumerator **ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT**

When set the static rand address complete, the event comes

enumerator **ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT**

When update connection parameters complete, the event comes

enumerator **ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT**

When set pkt length complete, the event comes

enumerator **ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT**

When Enable/disable privacy on the local device complete, the event comes

enumerator **ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT**

When remove the bond device complete, the event comes

enumerator **ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT**

When clear the bond device clear complete, the event comes

enumerator **ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT**

When get the bond device list complete, the event comes

enumerator **ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT**

When read the rssi complete, the event comes

enumerator **ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT**

When add or remove whitelist complete, the event comes

enumerator **ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT**

When update duplicate exceptional list complete, the event comes

enumerator **ESP_GAP_BLE_SET_CHANNELS_EVT**

When setting BLE channels complete, the event comes

enumerator **ESP_GAP_BLE_READ_PHY_COMPLETE_EVT**

when reading phy complete, this event comes

enumerator **ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT**

when preferred default phy complete, this event comes

enumerator **ESP_GAP_BLE_SET_PREFERRED_PHY_COMPLETE_EVT**

when preferred phy complete , this event comes

enumerator **ESP_GAP_BLE_EXT_ADV_SET_RAND_ADDR_COMPLETE_EVT**

when extended set random address complete, the event comes

enumerator **ESP_GAP_BLE_EXT_ADV_SET_PARAMS_COMPLETE_EVT**

when extended advertising parameter complete, the event comes

enumerator **ESP_GAP_BLE_EXT_ADV_DATA_SET_COMPLETE_EVT**

when extended advertising data complete, the event comes

enumerator **ESP_GAP_BLE_EXT_SCAN_RSP_DATA_SET_COMPLETE_EVT**

when extended scan response data complete, the event comes

enumerator **ESP_GAP_BLE_EXT_ADV_START_COMPLETE_EVT**

when extended advertising start complete, the event comes

enumerator **ESP_GAP_BLE_EXT_ADV_STOP_COMPLETE_EVT**

when extended advertising stop complete, the event comes

enumerator **ESP_GAP_BLE_EXT_ADV_SET_REMOVE_COMPLETE_EVT**

when extended advertising set remove complete, the event comes

enumerator **ESP_GAP_BLE_EXT_ADV_SET_CLEAR_COMPLETE_EVT**

when extended advertising set clear complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_SET_PARAMS_COMPLETE_EVT**
when periodic advertising parameter complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_DATA_SET_COMPLETE_EVT**
when periodic advertising data complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_START_COMPLETE_EVT**
when periodic advertising start complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_STOP_COMPLETE_EVT**
when periodic advertising stop complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_CREATE_SYNC_COMPLETE_EVT**
when periodic advertising create sync complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_SYNC_CANCEL_COMPLETE_EVT**
when extended advertising sync cancel complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_SYNC_TERMINATE_COMPLETE_EVT**
when extended advertising sync terminate complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_ADD_DEV_COMPLETE_EVT**
when extended advertising add device complete , the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_REMOVE_DEV_COMPLETE_EVT**
when extended advertising remove device complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_CLEAR_DEV_COMPLETE_EVT**
when extended advertising clear device, the event comes

enumerator **ESP_GAP_BLE_SET_EXT_SCAN_PARAMS_COMPLETE_EVT**
when extended scan parameter complete, the event comes

enumerator **ESP_GAP_BLE_EXT_SCAN_START_COMPLETE_EVT**
when extended scan start complete, the event comes

enumerator **ESP_GAP_BLE_EXT_SCAN_STOP_COMPLETE_EVT**
when extended scan stop complete, the event comes

enumerator **ESP_GAP_BLE_PREFER_EXT_CONN_PARAMS_SET_COMPLETE_EVT**
when extended prefer connection parameter set complete, the event comes

enumerator **ESP_GAP_BLE_PHY_UPDATE_COMPLETE_EVT**
when ble phy update complete, the event comes

enumerator **ESP_GAP_BLE_EXT_ADV_REPORT_EVT**
when extended advertising report complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_TIMEOUT_EVT**
when scan timeout complete, the event comes

enumerator **ESP_GAP_BLE_ADV_TERMINATED_EVT**
when advertising terminate data complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_REQ_RECEIVED_EVT**
when scan req received complete, the event comes

enumerator **ESP_GAP_BLE_CHANNEL_SELECT_ALGORITHM_EVT**
when channel select algorithm complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_REPORT_EVT**
when periodic report advertising complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_SYNC_LOST_EVT**
when periodic advertising sync lost complete, the event comes

enumerator **ESP_GAP_BLE_PERIODIC_ADV_SYNC_ESTAB_EVT**
when periodic advertising sync establish complete, the event comes

enumerator **ESP_GAP_BLE_SC_OOB_REQ_EVT**
Secure Connection OOB request event

enumerator **ESP_GAP_BLE_SC_CR_LOC_OOB_EVT**
Secure Connection create OOB data complete event

enumerator **ESP_GAP_BLE_GET_DEV_NAME_COMPLETE_EVT**
When getting BT device name complete, the event comes

enumerator **ESP_GAP_BLE_EVT_MAX**
when maximum advertising event complete, the event comes

enum **esp_ble_adv_data_type**
The type of advertising data(not adv_type)

Values:

enumerator **ESP_BLE_AD_TYPE_FLAG**

enumerator **ESP_BLE_AD_TYPE_16SRV_PART**

enumerator **ESP_BLE_AD_TYPE_16SRV_CMPL**

enumerator **ESP_BLE_AD_TYPE_32SRV_PART**

enumerator **ESP_BLE_AD_TYPE_32SRV_CMPL**

enumerator **ESP_BLE_AD_TYPE_128SRV_PART**

enumerator **ESP_BLE_AD_TYPE_128SRV_CMPL**

enumerator **ESP_BLE_AD_TYPE_NAME_SHORT**

enumerator **ESP_BLE_AD_TYPE_NAME_CMPL**

enumerator **ESP_BLE_AD_TYPE_TX_PWR**

enumerator **ESP_BLE_AD_TYPE_DEV_CLASS**

enumerator **ESP_BLE_AD_TYPE_SM_TK**

enumerator **ESP_BLE_AD_TYPE_SM_OOB_FLAG**

enumerator **ESP_BLE_AD_TYPE_INT_RANGE**

enumerator **ESP_BLE_AD_TYPE_SOL_SRV_UUID**

enumerator **ESP_BLE_AD_TYPE_128SOL_SRV_UUID**

enumerator **ESP_BLE_AD_TYPE_SERVICE_DATA**

enumerator **ESP_BLE_AD_TYPE_PUBLIC_TARGET**

enumerator **ESP_BLE_AD_TYPE_RANDOM_TARGET**

enumerator **ESP_BLE_AD_TYPE_APPEARANCE**

enumerator **ESP_BLE_AD_TYPE_ADV_INT**

enumerator **ESP_BLE_AD_TYPE_LE_DEV_ADDR**

enumerator **ESP_BLE_AD_TYPE_LE_ROLE**

enumerator **ESP_BLE_AD_TYPE_SPAIR_C256**

enumerator **ESP_BLE_AD_TYPE_SPAIR_R256**

enumerator **ESP_BLE_AD_TYPE_32SOL_SRV_UUID**

enumerator **ESP_BLE_AD_TYPE_32SERVICE_DATA**

enumerator **ESP_BLE_AD_TYPE_128SERVICE_DATA**

enumerator **ESP_BLE_AD_TYPE_LE_SECURE_CONFIRM**

enumerator **ESP_BLE_AD_TYPE_LE_SECURE_RANDOM**

enumerator **ESP_BLE_AD_TYPE_URI**

enumerator **ESP_BLE_AD_TYPE_INDOOR_POSITION**

enumerator **ESP_BLE_AD_TYPE_TRANS_DISC_DATA**

enumerator **ESP_BLE_AD_TYPE_LE_SUPPORT_FEATURE**

enumerator **ESP_BLE_AD_TYPE_CHAN_MAP_UPDATE**

enumerator **ESP_BLE_AD_MANUFACTURER_SPECIFIC_TYPE**

enum **esp_ble_adv_type_t**

Advertising mode.

Values:

enumerator **ADV_TYPE_IND**

enumerator **ADV_TYPE_DIRECT_IND_HIGH**

enumerator **ADV_TYPE_SCAN_IND**

enumerator **ADV_TYPE_NONCONN_IND**

enumerator **ADV_TYPE_DIRECT_IND_LOW**

enum **esp_ble_adv_channel_t**

Advertising channel mask.

Values:

enumerator **ADV_CHNL_37**

enumerator **ADV_CHNL_38**

enumerator **ADV_CHNL_39**

enumerator **ADV_CHNL_ALL**

enum **esp_ble_adv_filter_t**

Values:

enumerator **ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY**

Allow both scan and connection requests from anyone.

enumerator **ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY**

Allow both scan req from White List devices only and connection req from anyone.

enumerator **ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST**

Allow both scan req from anyone and connection req from White List devices only.

enumerator **ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST**

Allow scan and connection requests from White List devices only.

enum **esp_ble_sec_act_t**

Values:

enumerator **ESP_BLE_SEC_ENCRYPT**

relate to BTA_DM_BLE_SEC_ENCRYPT in bta/bta_api.h. If the device has already bonded, the stack will used Long Term Key (LTK) to encrypt with the remote device directly. Else if the device hasn't bonded, the stack will used the default authentication request used the esp_ble_gap_set_security_param function set by the user.

enumerator **ESP_BLE_SEC_ENCRYPT_NO_MITM**

relate to BTA_DM_BLE_SEC_ENCRYPT_NO_MITM in bta/bta_api.h. If the device has been already bonded, the stack will check the LTK (Long Term Key) Whether the authentication request has been met, and if met, use the LTK to encrypt with the remote device directly, else re-pair with the remote device. Else if the device hasn't been bonded, the stack will use NO MITM authentication request in the current link instead of using the authreq in the esp_ble_gap_set_security_param function set by the user.

enumerator **ESP_BLE_SEC_ENCRYPT_MITM**

relate to BTA_DM_BLE_SEC_ENCRYPT_MITM in bta/bta_api.h. If the device has been already bonded, the stack will check the LTK (Long Term Key) whether the authentication request has been met, and if met, use the LTK to encrypt with the remote device directly, else re-pair with the remote device. Else if the device hasn't been bonded, the stack will use MITM authentication request in the current link instead of using the authreq in the esp_ble_gap_set_security_param function set by the user.

enum **esp_ble_sm_param_t**

Values:

enumerator **ESP_BLE_SM_PASSKEY**

Authentication requirements of local device

enumerator **ESP_BLE_SM_AUTHEN_REQ_MODE**

The IO capability of local device

enumerator **ESP_BLE_SM_IOCAP_MODE**

Initiator Key Distribution/Generation

enumerator **ESP_BLE_SM_SET_INIT_KEY**

Responder Key Distribution/Generation

enumerator **ESP_BLE_SM_SET_RSP_KEY**

Maximum Encryption key size to support

enumerator **ESP_BLE_SM_MAX_KEY_SIZE**

Minimum Encryption key size requirement from Peer

enumerator **ESP_BLE_SM_MIN_KEY_SIZE**

Set static Passkey

enumerator **ESP_BLE_SM_SET_STATIC_PASSKEY**

Reset static Passkey

enumerator **ESP_BLE_SM_CLEAR_STATIC_PASSKEY**

Accept only specified SMP Authentication requirement

enumerator **ESP_BLE_SM_ONLY_ACCEPT_SPECIFIED_SEC_AUTH**

Enable/Disable OOB support

enumerator **ESP_BLE_SM_OOB_SUPPORT**

Appl encryption key size

enumerator **ESP_BLE_APP_ENC_KEY_SIZE**

authentication max param

enumerator **ESP_BLE_SM_MAX_PARAM**

enum **esp_ble_scan_type_t**

Ble scan type.

Values:

enumerator **BLE_SCAN_TYPE_PASSIVE**

Passive scan

enumerator **BLE_SCAN_TYPE_ACTIVE**

Active scan

enum **esp_ble_scan_filter_t**

Ble scan filter type.

Values:

enumerator **BLE_SCAN_FILTER_ALLOW_ALL**

Accept all :

i. advertisement packets except directed advertising packets not addressed to this device (default).

enumerator **BLE_SCAN_FILTER_ALLOW_ONLY_WLST**

Accept only :

i. advertisement packets from devices where the advertiser’ s address is in the White list.
ii. Directed advertising packets which are not addressed for this device shall be ignored.

enumerator **BLE_SCAN_FILTER_ALLOW_UND_RPA_DIR**

Accept all :

i. undirected advertisement packets, and
ii. directed advertising packets where the initiator address is a resolvable private address, and
iii. directed advertising packets addressed to this device.

enumerator **BLE_SCAN_FILTER_ALLOW_WLIST_RPA_DIR**

Accept all :

i. advertisement packets from devices where the advertiser’ s address is in the White list, and
ii. directed advertising packets where the initiator address is a resolvable private address, and
iii. directed advertising packets addressed to this device.

enum **esp_ble_scan_duplicate_t**

Ble scan duplicate type.

Values:

enumerator **BLE_SCAN_DUPLICATE_DISABLE**

the Link Layer should generate advertising reports to the host for each packet received

enumerator **BLE_SCAN_DUPLICATE_ENABLE**

the Link Layer should filter out duplicate advertising reports to the Host

enumerator **BLE_SCAN_DUPLICATE_MAX**

0x02 –0xFF, Reserved for future use

enum **esp_gap_search_evt_t**

Sub Event of ESP_GAP_BLE_SCAN_RESULT_EVT.

Values:

enumerator **ESP_GAP_SEARCH_INQ_RES_EVT**

Inquiry result for a peer device.

enumerator **ESP_GAP_SEARCH_INQ_CMPL_EVT**

Inquiry complete.

enumerator **ESP_GAP_SEARCH_DISC_RES_EVT**

Discovery result for a peer device.

enumerator **ESP_GAP_SEARCH_DISC_BLE_RES_EVT**

Discovery result for BLE GATT based service on a peer device.

enumerator **ESP_GAP_SEARCH_DISC_CMPL_EVT**

Discovery complete.

enumerator **ESP_GAP_SEARCH_DI_DISC_CMPL_EVT**

Discovery complete.

enumerator **ESP_GAP_SEARCH_SEARCH_CANCEL_CMPL_EVT**

Search cancelled

enumerator **ESP_GAP_SEARCH_INQ_DISCARD_NUM_EVT**

The number of pkt discarded by flow control

enum **esp_ble_evt_type_t**

Ble scan result event type, to indicate the result is scan response or advertising data or other.

Values:

enumerator **ESP_BLE_EVT_CONN_ADV**

Connectable undirected advertising (ADV_IND)

enumerator **ESP_BLE_EVT_CONN_DIR_ADV**

Connectable directed advertising (ADV_DIRECT_IND)

enumerator **ESP_BLE_EVT_DISC_ADV**

Scannable undirected advertising (ADV_SCAN_IND)

enumerator **ESP_BLE_EVT_NON_CONN_ADV**

Non connectable undirected advertising (ADV_NONCONN_IND)

enumerator **ESP_BLE_EVT_SCAN_RSP**

Scan Response (SCAN_RSP)

enum **esp_ble_wl_operation_t**

Values:

enumerator **ESP_BLE_WHITELIST_REMOVE**

remove mac from whitelist

enumerator **ESP_BLE_WHITELIST_ADD**

add address to whitelist

enumerator **ESP_BLE_WHITELIST_CLEAR**

clear all device in whitelist

enum **esp_bt_duplicate_exceptional_subcode_type_t**

Values:

enumerator **ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_ADD**

Add device info into duplicate scan exceptional list

enumerator **ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_REMOVE**

Remove device info from duplicate scan exceptional list

enumerator **ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_CLEAN**

Clean duplicate scan exceptional list

enum **esp_ble_duplicate_exceptional_info_type_t**

Values:

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_ADV_ADDR**

BLE advertising address , device info will be added into ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_LINK_ID**

BLE mesh link ID, it is for BLE mesh, device info will be added into ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_BEACON_TYPE**

BLE mesh beacon AD type, the format is | Len | 0x2B | Beacon Type | Beacon Data |

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_PROV_SRV_ADV**

BLE mesh provisioning service uuid, the format is | 0x02 | 0x01 | flags | 0x03 | 0x03 | 0x1827 | ···. |‘

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_PROXY_SRV_ADV**

BLE mesh adv with proxy service uuid, the format is | 0x02 | 0x01 | flags | 0x03 | 0x03 | 0x1828 | ···. |‘

enum **esp_duplicate_scan_exceptional_list_type_t**

Values:

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST**

duplicate scan exceptional addr list

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST**

duplicate scan exceptional mesh link ID list

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_BEACON_TYPE_LIST**

duplicate scan exceptional mesh beacon type list

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROV_SRV_ADV_LIST**

duplicate scan exceptional mesh adv with provisioning service uuid

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROXY_SRV_ADV_LIST**

duplicate scan exceptional mesh adv with provisioning service uuid

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ALL_LIST**

duplicate scan exceptional all list

GATT DEFINES

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_gatt_defs.h

Unions

union **esp_gatt_rsp_t**

#include <esp_gatt_defs.h> GATT remote read request response type.

Public Members

esp_gatt_value_t **attr_value**

Gatt attribute structure

uint16_t **handle**

Gatt attribute handle

Structures

struct **esp_gatt_id_t**

Gatt id, include uuid and instance id.

Public Members

esp_bt_uuid_t **uuid**

UUID

uint8_t **inst_id**

Instance id

struct **esp_gatt_srvc_id_t**

Gatt service id, include id (uuid and instance id) and primary flag.

Public Members

esp_gatt_id_t **id**

Gatt id, include uuid and instance

bool **is_primary**

This service is primary or not

struct **esp_attr_desc_t**

Attribute description (used to create database)

Public Members

uint16_t **uuid_length**

UUID length

uint8_t ***uuid_p**

UUID value

uint16_t **perm**

Attribute permission

uint16_t **max_length**

Maximum length of the element

uint16_t **length**

Current length of the element

uint8_t ***value**

Element value array

struct **esp_attr_control_t**

attribute auto response flag

Public Members

uint8_t **auto_rsp**

if auto_rsp set to ESP_GATT_RSP_BY_APP, means the response of Write/Read operation will by replied by application. if auto_rsp set to ESP_GATT_AUTO_RSP, means the response of Write/Read operation will be replied by GATT stack automatically.

struct **esp_gatts_attr_db_t**

attribute type added to the gatt server database

Public Members

esp_attr_control_t **attr_control**

The attribute control type

esp_attr_desc_t **att_desc**

The attribute type

struct **esp_attr_value_t**

set the attribute value type

Public Members

uint16_t **attr_max_len**

attribute max value length

uint16_t **attr_len**

attribute current value length

uint8_t ***attr_value**

the pointer to attribute value

struct **esp_gatts_incl_svc_desc_t**

Gatt include service entry element.

Public Members

uint16_t **start_hdl**

Gatt start handle value of included service

uint16_t **end_hdl**

Gatt end handle value of included service

uint16_t **uuid**

Gatt attribute value UUID of included service

struct **esp_gatts_incl128_svc_desc_t**

Gatt include 128 bit service entry element.

Public Members

uint16_t **start_hdl**

Gatt start handle value of included 128 bit service

uint16_t **end_hdl**

Gatt end handle value of included 128 bit service

struct **esp_gatt_value_t**

Gatt attribute value.

Public Members

uint8_t **value**[ESP_GATT_MAX_ATTR_LEN]

Gatt attribute value

uint16_t **handle**

Gatt attribute handle

uint16_t **offset**

Gatt attribute value offset

uint16_t **len**

Gatt attribute value length

uint8_t **auth_req**

Gatt authentication request

struct **esp_gatt_conn_params_t**

Connection parameters information.

Public Members

uint16_t **interval**

connection interval

uint16_t **latency**

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t **timeout**

Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds

struct **esp_gattc_multi_t**

read multiple attribute

Public Members

uint8_t **num_attr**

The number of the attribute

uint16_t **handles**[ESP_GATT_MAX_READ_MULTI_HANDLES]

The handles list

struct **esp_gattc_db_elem_t**

data base attribute element

Public Members

esp_gatt_db_attr_type_t **type**

The attribute type

uint16_t **attribute_handle**

The attribute handle, it's valid for all of the type

uint16_t **start_handle**

The service start handle, it's valid only when the type = ESP_GATT_DB_PRIMARY_SERVICE or ESP_GATT_DB_SECONDARY_SERVICE

uint16_t **end_handle**

The service end handle, it's valid only when the type = ESP_GATT_DB_PRIMARY_SERVICE or ESP_GATT_DB_SECONDARY_SERVICE

esp_gatt_char_prop_t **properties**

The characteristic properties, it's valid only when the type = ESP_GATT_DB_CHARACTERISTIC

esp_bt_uuid_t **uuid**

The attribute uuid, it's valid for all of the type

struct **esp_gattc_service_elem_t**

service element

Public Members

bool **is_primary**

The service flag, true if the service is primary service, else is secondary service

uint16_t **start_handle**

The start handle of the service

uint16_t **end_handle**

The end handle of the service

esp_bt_uuid_t **uuid**

The uuid of the service

struct **esp_gattc_char_elem_t**

characteristic element

Public Members

uint16_t **char_handle**

The characteristic handle

esp_gatt_char_prop_t **properties**

The characteristic properties

esp_bt_uuid_t **uuid**

The characteristic uuid

struct **esp_gattc_descr_elem_t**

descriptor element

Public Members

uint16_t **handle**

The characteristic descriptor handle

esp_bt_uuid_t **uuid**

The characteristic descriptor uuid

struct **esp_gattc_incl_svc_elem_t**

include service element

Public Members

uint16_t **handle**

The include service current attribute handle

uint16_t **incl_srvc_s_handle**

The start handle of the service which has been included

uint16_t **incl_srvc_e_handle**

The end handle of the service which has been included

esp_bt_uuid_t **uuid**

The include service uuid

Macros

ESP_GATT_UUID_IMMEDIATE_ALERT_SVC

All "ESP_GATT_UUID_xxx" is attribute types

ESP_GATT_UUID_LINK_LOSS_SVC

ESP_GATT_UUID_TX_POWER_SVC

ESP_GATT_UUID_CURRENT_TIME_SVC

ESP_GATT_UUID_REF_TIME_UPDATE_SVC

ESP_GATT_UUID_NEXT_DST_CHANGE_SVC

ESP_GATT_UUID_GLUCOSE_SVC

ESP_GATT_UUID_HEALTH_THERMOM_SVC

ESP_GATT_UUID_DEVICE_INFO_SVC

ESP_GATT_UUID_HEART_RATE_SVC

ESP_GATT_UUID_PHONE_ALERT_STATUS_SVC

ESP_GATT_UUID_BATTERY_SERVICE_SVC

ESP_GATT_UUID_BLOOD_PRESSURE_SVC

ESP_GATT_UUID_ALERT_NTF_SVC

ESP_GATT_UUID_HID_SVC

ESP_GATT_UUID_SCAN_PARAMETERS_SVC

ESP_GATT_UUID_RUNNING_SPEED_CADENCE_SVC

ESP_GATT_UUID_Automation_IO_SVC

ESP_GATT_UUID_CYCLING_SPEED_CADENCE_SVC

ESP_GATT_UUID_CYCLING_POWER_SVC

ESP_GATT_UUID_LOCATION_AND_NAVIGATION_SVC

ESP_GATT_UUID_ENVIRONMENTAL_SENSING_SVC

ESP_GATT_UUID_BODY_COMPOSITION

ESP_GATT_UUID_USER_DATA_SVC

ESP_GATT_UUID_WEIGHT_SCALE_SVC

ESP_GATT_UUID_BOND_MANAGEMENT_SVC

ESP_GATT_UUID_CONT_GLUCOSE_MONITOR_SVC

ESP_GATT_UUID_PRI_SERVICE

ESP_GATT_UUID_SEC_SERVICE

ESP_GATT_UUID_INCLUDE_SERVICE

ESP_GATT_UUID_CHAR_DECLARE

ESP_GATT_UUID_CHAR_EXT_PROP

ESP_GATT_UUID_CHAR_DESCRIPTION

ESP_GATT_UUID_CHAR_CLIENT_CONFIG

ESP_GATT_UUID_CHAR_SRVR_CONFIG

ESP_GATT_UUID_CHAR_PRESENT_FORMAT

ESP_GATT_UUID_CHAR_AGG_FORMAT

ESP_GATT_UUID_CHAR_VALID_RANGE

ESP_GATT_UUID_EXT_RPT_REF_DESCR

ESP_GATT_UUID_RPT_REF_DESCR

ESP_GATT_UUID_NUM_DIGITALS_DESCR

ESP_GATT_UUID_VALUE_TRIGGER_DESCR

ESP_GATT_UUID_ENV_SENSING_CONFIG_DESCR

ESP_GATT_UUID_ENV_SENSING_MEASUREMENT_DESCR

ESP_GATT_UUID_ENV_SENSING_TRIGGER_DESCR

ESP_GATT_UUID_TIME_TRIGGER_DESCR

ESP_GATT_UUID_GAP_DEVICE_NAME

ESP_GATT_UUID_GAP_ICON

ESP_GATT_UUID_GAP_PREF_CONN_PARAM

ESP_GATT_UUID_GAP_CENTRAL_ADDR_RESOL

ESP_GATT_UUID_GATT_SRV_CHGD

ESP_GATT_UUID_ALERT_LEVEL

ESP_GATT_UUID_TX_POWER_LEVEL

ESP_GATT_UUID_CURRENT_TIME

ESP_GATT_UUID_LOCAL_TIME_INFO

ESP_GATT_UUID_REF_TIME_INFO

ESP_GATT_UUID_NW_STATUS

ESP_GATT_UUID_NW_TRIGGER

ESP_GATT_UUID_ALERT_STATUS

ESP_GATT_UUID_RINGER_CP

ESP_GATT_UUID_RINGER_SETTING

ESP_GATT_UUID_GM_MEASUREMENT

ESP_GATT_UUID_GM_CONTEXT

ESP_GATT_UUID_GM_CONTROL_POINT

ESP_GATT_UUID_GM_FEATURE

ESP_GATT_UUID_SYSTEM_ID

ESP_GATT_UUID_MODEL_NUMBER_STR

ESP_GATT_UUID_SERIAL_NUMBER_STR

ESP_GATT_UUID_FW_VERSION_STR

ESP_GATT_UUID_HW_VERSION_STR

ESP_GATT_UUID_SW_VERSION_STR

ESP_GATT_UUID_MANU_NAME

ESP_GATT_UUID_IEEE_DATA

ESP_GATT_UUID_PNP_ID

ESP_GATT_UUID_HID_INFORMATION

ESP_GATT_UUID_HID_REPORT_MAP

ESP_GATT_UUID_HID_CONTROL_POINT

ESP_GATT_UUID_HID_REPORT

ESP_GATT_UUID_HID_PROTO_MODE

ESP_GATT_UUID_HID_BT_KB_INPUT

ESP_GATT_UUID_HID_BT_KB_OUTPUT

ESP_GATT_UUID_HID_BT_MOUSE_INPUT

ESP_GATT_HEART_RATE_MEAS
Heart Rate Measurement.

ESP_GATT_BODY_SENSOR_LOCATION
Body Sensor Location.

ESP_GATT_HEART_RATE_CNTL_POINT
Heart Rate Control Point.

ESP_GATT_UUID_BATTERY_LEVEL

ESP_GATT_UUID_SC_CONTROL_POINT

ESP_GATT_UUID_SENSOR_LOCATION

ESP_GATT_UUID_RSC_MEASUREMENT

ESP_GATT_UUID_RSC_FEATURE

ESP_GATT_UUID_CSC_MEASUREMENT

ESP_GATT_UUID_CSC_FEATURE

ESP_GATT_UUID_SCAN_INT_WINDOW

ESP_GATT_UUID_SCAN_REFRESH

ESP_GATT_ILLEGAL_UUID
GATT INVALID UUID.

ESP_GATT_ILLEGAL_HANDLE
GATT INVALID HANDLE.

ESP_GATT_ATTR_HANDLE_MAX
GATT attribute max handle.

ESP_GATT_MAX_READ_MULTI_HANDLES

ESP_GATT_PERM_READ
Attribute permissions.

ESP_GATT_PERM_READ_ENCRYPTED

ESP_GATT_PERM_READ_ENC_MITM

ESP_GATT_PERM_WRITE

ESP_GATT_PERM_WRITE_ENCRYPTED

ESP_GATT_PERM_WRITE_ENC_MITM

ESP_GATT_PERM_WRITE_SIGNED

ESP_GATT_PERM_WRITE_SIGNED_MITM

ESP_GATT_PERM_READ_AUTHORIZATION

ESP_GATT_PERM_WRITE_AUTHORIZATION

ESP_GATT_PERM_ENCRYPT_KEY_SIZE(keysize)

ESP_GATT_CHAR_PROP_BIT_BROADCAST

ESP_GATT_CHAR_PROP_BIT_READ

ESP_GATT_CHAR_PROP_BIT_WRITE_NR

ESP_GATT_CHAR_PROP_BIT_WRITE

ESP_GATT_CHAR_PROP_BIT_NOTIFY

ESP_GATT_CHAR_PROP_BIT_INDICATE

ESP_GATT_CHAR_PROP_BIT_AUTH

ESP_GATT_CHAR_PROP_BIT_EXT_PROP

ESP_GATT_MAX_ATTR_LEN

GATT maximum attribute length.

ESP_GATT_RSP_BY_APP

ESP_GATT_AUTO_RSP

ESP_GATT_IF_NONE

If callback report gattc_if/gatts_if as this macro, means this event is not correspond to any app

Type Definitions

typedef uint16_t **esp_gatt_perm_t**

typedef uint8_t **esp_gatt_char_prop_t**

typedef uint8_t **esp_gatt_if_t**

Gatt interface type, different application on GATT client use different gatt_if

Enumerations

enum **esp_gatt_prep_write_type**

Attribute write data type from the client.

Values:

enumerator **ESP_GATT_PREP_WRITE_CANCEL**

Prepare write cancel

enumerator **ESP_GATT_PREP_WRITE_EXEC**

Prepare write execute

enum **esp_gatt_status_t**

GATT success code and error codes.

Values:

enumerator **ESP_GATT_OK**

enumerator **ESP_GATT_INVALID_HANDLE**

enumerator **ESP_GATT_READ_NOT_PERMIT**

enumerator **ESP_GATT_WRITE_NOT_PERMIT**

enumerator **ESP_GATT_INVALID_PDU**

enumerator **ESP_GATT_INSUF_AUTHENTICATION**

enumerator **ESP_GATT_REQ_NOT_SUPPORTED**

enumerator **ESP_GATT_INVALID_OFFSET**

enumerator **ESP_GATT_INSUF_AUTHORIZATION**

enumerator **ESP_GATT_PREPARE_Q_FULL**

enumerator **ESP_GATT_NOT_FOUND**

enumerator **ESP_GATT_NOT_LONG**

enumerator **ESP_GATT_INSUF_KEY_SIZE**

enumerator **ESP_GATT_INVALID_ATTR_LEN**

enumerator **ESP_GATT_ERR_UNLIKELY**

enumerator **ESP_GATT_INSUF_ENCRYPTION**

enumerator **ESP_GATT_UNSUPPORT_GRP_TYPE**

enumerator **ESP_GATT_INSUF_RESOURCE**

enumerator **ESP_GATT_NO_RESOURCES**

enumerator **ESP_GATT_INTERNAL_ERROR**

enumerator **ESP_GATT_WRONG_STATE**

enumerator **ESP_GATT_DB_FULL**

enumerator **ESP_GATT_BUSY**

enumerator **ESP_GATT_ERROR**

enumerator **ESP_GATT_CMD_STARTED**

enumerator **ESP_GATT_ILLEGAL_PARAMETER**

enumerator **ESP_GATT_PENDING**

enumerator **ESP_GATT_AUTH_FAIL**

enumerator **ESP_GATT_MORE**

enumerator **ESP_GATT_INVALID_CFG**

enumerator **ESP_GATT_SERVICE_STARTED**

enumerator **ESP_GATT_ENCRYPTED_MITM**

enumerator **ESP_GATT_ENCRYPTED_NO_MITM**

enumerator **ESP_GATT_NOT_ENCRYPTED**

enumerator **ESP_GATT_CONGESTED**

enumerator **ESP_GATT_DUP_REG**

enumerator **ESP_GATT_ALREADY_OPEN**

enumerator **ESP_GATT_CANCEL**

enumerator **ESP_GATT_STACK_RSP**

enumerator **ESP_GATT_APP_RSP**

enumerator **ESP_GATT_UNKNOWN_ERROR**

enumerator **ESP_GATT_CCC_CFG_ERR**

enumerator **ESP_GATT_PRC_IN_PROGRESS**

enumerator **ESP_GATT_OUT_OF_RANGE**

enum **esp_gatt_conn_reason_t**

Gatt Connection reason enum.

Values:

enumerator **ESP_GATT_CONN_UNKNOWN**

Gatt connection unknown

enumerator **ESP_GATT_CONN_L2C_FAILURE**

General L2cap failure

enumerator **ESP_GATT_CONN_TIMEOUT**

Connection timeout

enumerator **ESP_GATT_CONN_TERMINATE_PEER_USER**

Connection terminate by peer user

enumerator **ESP_GATT_CONN_TERMINATE_LOCAL_HOST**

Connection terminated by local host

enumerator **ESP_GATT_CONN_FAIL_ESTABLISH**

Connection fail to establish

enumerator **ESP_GATT_CONN_LMP_TIMEOUT**

Connection fail for LMP response tout

enumerator **ESP_GATT_CONN_CONN_CANCEL**

L2CAP connection cancelled

enumerator **ESP_GATT_CONN_NONE**

No connection to cancel

enum **esp_gatt_auth_req_t**

Gatt authentication request type.

Values:

enumerator **ESP_GATT_AUTH_REQ_NONE**

enumerator **ESP_GATT_AUTH_REQ_NO_MITM**

enumerator **ESP_GATT_AUTH_REQ_MITM**

enumerator **ESP_GATT_AUTH_REQ_SIGNED_NO_MITM**

enumerator **ESP_GATT_AUTH_REQ_SIGNED_MITM**

enum **esp_service_source_t**

Values:

enumerator **ESP_GATT_SERVICE_FROM_REMOTE_DEVICE**

enumerator **ESP_GATT_SERVICE_FROM_NVS_FLASH**

enumerator **ESP_GATT_SERVICE_FROM_UNKNOWN**

enum **esp_gatt_write_type_t**

Gatt write type.

Values:

enumerator **ESP_GATT_WRITE_TYPE_NO_RSP**

Gatt write attribute need no response

enumerator **ESP_GATT_WRITE_TYPE_RSP**

Gatt write attribute need remote response

enum **esp_gatt_db_attr_type_t**

the type of attribute element

Values:

enumerator **ESP_GATT_DB_PRIMARY_SERVICE**

Gattc primary service attribute type in the cache

enumerator **ESP_GATT_DB_SECONDARY_SERVICE**

Gattc secondary service attribute type in the cache

enumerator **ESP_GATT_DB_CHARACTERISTIC**

Gattc characteristic attribute type in the cache

enumerator **ESP_GATT_DB_DESCRIPTOR**

Gattc characteristic descriptor attribute type in the cache

enumerator **ESP_GATT_DB_INCLUDED_SERVICE**

Gattc include service attribute type in the cache

enumerator **ESP_GATT_DB_ALL**

Gattc all the attribute (primary service & secondary service & include service & char & descriptor) type in the cache

GATT SERVER API

Application Example Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a GATT sever demo and its tutorial. This demo creates a GATT service with an attribute table, which releases the user from adding attributes one by one. This is the recommended method of adding attributes.
  - bluetooth/bluedroid/ble/gatt_server_service_table
  - GATT Server Service Table Example Walkthrough
- This is a GATT server demo and its tutorial. This demo creates a GATT service by adding attributes one by one as defined by Bluedroid. The recommended method of adding attributes is presented in example above.
  - bluetooth/bluedroid/ble/gatt_server
  - GATT Server Example Walkthrough
- This is a BLE SPP-Like demo. This demo, which acts as a GATT server, can receive data from UART and then send the data to the peer device automatically.
  - bluetooth/bluedroid/ble/ble_spp_server

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_gatts_api.h

Functions

esp_err_t **esp_ble_gatts_register_callback**(*esp_gatts_cb_t* callback)

This function is called to register application callbacks with BTA GATTS module.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_app_register**(uint16_t app_id)

This function is called to register application identifier.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_app_unregister**(*esp_gatt_if_t* gatts_if)

unregister with GATT Server.

参数 **gatts_if** – **[in]** GATT server access interface

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_create_service**(*esp_gatt_if_t* gatts_if, *esp_gatt_srvc_id_t* *service_id, uint16_t num_handle)

Create a service. When service creation is done, a callback event ESP_GATTS_CREATE_EVT is called to report status and service ID to the profile. The service ID obtained in the callback function needs to be used when adding included service and characteristics/descriptors into the service.

参数

- **gatts_if** – **[in]** GATT server access interface

- **service_id** –**[in]** service ID.
- **num_handle** –**[in]** number of handle requested for this service.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_create_attr_tab**(const *esp_gatts_attr_db_t* *gatts_attr_db, *esp_gatt_if_t* gatts_if, uint16_t max_nb_attr, uint8_t srvc_inst_id)

Create a service attribute tab.

参数

- **gatts_attr_db** –**[in]** the pointer to the service attr tab
- **gatts_if** –**[in]** GATT server access interface
- **max_nb_attr** –**[in]** the number of attribute to be added to the service database.
- **srvc_inst_id** –**[in]** the instance id of the service

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_add_included_service**(uint16_t service_handle, uint16_t included_service_handle)

This function is called to add an included service. This function have to be called between 'esp_ble_gatts_create_service' and 'esp_ble_gatts_add_char'. After included service is included, a callback event ESP_GATTS_ADD_INCL_SRVC_EVT is reported the included service ID.

参数

- **service_handle** –**[in]** service handle to which this included service is to be added.
- **included_service_handle** –**[in]** the service ID to be included.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_add_char**(uint16_t service_handle, *esp_bt_uuid_t* *char_uuid, *esp_gatt_perm_t* perm, *esp_gatt_char_prop_t* property, *esp_attr_value_t* *char_val, *esp_attr_control_t* *control)

This function is called to add a characteristic into a service.

参数

- **service_handle** –**[in]** service handle to which this included service is to be added.
- **char_uuid** –**[in]** : Characteristic UUID.
- **perm** –**[in]** : Characteristic value declaration attribute permission.
- **property** –**[in]** : Characteristic Properties
- **char_val** –**[in]** : Characteristic value
- **control** –**[in]** : attribute response control byte

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_add_char_descr**(uint16_t service_handle, *esp_bt_uuid_t* *descr_uuid, *esp_gatt_perm_t* perm, *esp_attr_value_t* *char_descr_val, *esp_attr_control_t* *control)

This function is called to add characteristic descriptor. When it's done, a callback event ESP_GATTS_ADD_DESCR_EVT is called to report the status and an ID number for this descriptor.

参数

- **service_handle** –**[in]** service handle to which this characteristic descriptor is to be added.
- **perm** –**[in]** descriptor access permission.
- **descr_uuid** –**[in]** descriptor UUID.
- **char_descr_val** –**[in]** : Characteristic descriptor value
- **control** –**[in]** : attribute response control byte

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_delete_service**(uint16_t service_handle)

This function is called to delete a service. When this is done, a callback event ESP_GATTS_DELETE_EVT is report with the status.

参数 **service_handle** –**[in]** service_handle to be deleted.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_start_service**(uint16_t service_handle)

This function is called to start a service.

参数 **service_handle** –**[in]** the service handle to be started.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_stop_service**(uint16_t service_handle)

This function is called to stop a service.

参数 **service_handle** –**[in]** - service to be topped.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_send_indicate**(*esp_gatt_if_t* gatts_if, uint16_t conn_id, uint16_t attr_handle, uint16_t value_len, uint8_t *value, bool need_confirm)

Send indicate or notify to GATT client. Set param need_confirm as false will send notification, otherwise indication.

参数

- **gatts_if** –**[in]** GATT server access interface
- **conn_id** –**[in]** - connection id to indicate.
- **attr_handle** –**[in]** - attribute handle to indicate.
- **value_len** –**[in]** - indicate value length.
- **value** –**[in]** value to indicate.
- **need_confirm** –**[in]** - Whether a confirmation is required. false sends a GATT notification, true sends a GATT indication.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_send_response**(*esp_gatt_if_t* gatts_if, uint16_t conn_id, uint32_t trans_id, *esp_gatt_status_t* status, *esp_gatt_rsp_t* *rsp)

This function is called to send a response to a request.

参数

- **gatts_if** –**[in]** GATT server access interface
- **conn_id** –**[in]** - connection identifier.
- **trans_id** –**[in]** - transfer id
- **status** –**[in]** - response status
- **rsp** –**[in]** - response data.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_set_attr_value**(uint16_t attr_handle, uint16_t length, const uint8_t *value)

This function is called to set the attribute value by the application.

参数

- **attr_handle** –**[in]** the attribute handle which to be set
- **length** –**[in]** the value length
- **value** –**[in]** the pointer to the attribute value

返回

- ESP_OK : success
- other : failed

esp_gatt_status_t **esp_ble_gatts_get_attr_value**(uint16_t attr_handle, uint16_t *length, const uint8_t **value)

Retrieve attribute value.

参数

- **attr_handle** –**[in]** Attribute handle.
- **length** –**[out]** pointer to the attribute value length
- **value** –**[out]** Pointer to attribute value payload, the value cannot be modified by user

返回

- ESP_GATT_OK : success
- other : failed

esp_err_t **esp_ble_gatts_open**(*esp_gatt_if_t* gatts_if, *esp_bd_addr_t* remote_bda, bool is_direct)

Open a direct open connection or add a background auto connection.

参数

- **gatts_if** –**[in]** GATT server access interface
- **remote_bda** –**[in]** remote device bluetooth device address.
- **is_direct** –**[in]** direct connection or background auto connection

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_close**(*esp_gatt_if_t* gatts_if, uint16_t conn_id)

Close a connection a remote device.

参数

- **gatts_if** –**[in]** GATT server access interface
- **conn_id** –**[in]** connection ID to be closed.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_send_service_change_indication**(*esp_gatt_if_t* gatts_if, *esp_bd_addr_t* remote_bda)

Send service change indication.

参数

- **gatts_if** –**[in]** GATT server access interface
- **remote_bda** –**[in]** remote device bluetooth device address. If remote_bda is NULL then it will send service change indication to all the connected devices and if not then to a specific device

返回

- ESP_OK : success
- other : failed

Unions

union **esp_ble_gatts_cb_param_t**

#include <esp_gatts_api.h> Gatt server callback parameters union.

Public Members

struct *esp_ble_gatts_cb_param_t*::*gatts_reg_evt_param* **reg**

Gatt server callback param of ESP_GATTS_REG_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_read_evt_param* **read**

Gatt server callback param of ESP_GATTS_READ_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_write_evt_param* **write**

Gatt server callback param of ESP_GATTS_WRITE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_exec_write_evt_param* **exec_write**

Gatt server callback param of ESP_GATTS_EXEC_WRITE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_mtu_evt_param* **mtu**

Gatt server callback param of ESP_GATTS_MTU_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_conf_evt_param* **conf**

Gatt server callback param of ESP_GATTS_CONF_EVT (confirm)

struct *esp_ble_gatts_cb_param_t*::*gatts_create_evt_param* **create**

Gatt server callback param of ESP_GATTS_CREATE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_add_incl_srvc_evt_param* **add_incl_srvc**

Gatt server callback param of ESP_GATTS_ADD_INCL_SRVC_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_add_char_evt_param* **add_char**

Gatt server callback param of ESP_GATTS_ADD_CHAR_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_add_char_descr_evt_param* **add_char_descr**

Gatt server callback param of ESP_GATTS_ADD_CHAR_DESCR_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_delete_evt_param* **del**

Gatt server callback param of ESP_GATTS_DELETE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_start_evt_param* **start**

Gatt server callback param of ESP_GATTS_START_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_stop_evt_param* **stop**

Gatt server callback param of ESP_GATTS_STOP_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_connect_evt_param* **connect**

Gatt server callback param of ESP_GATTS_CONNECT_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_disconnect_evt_param* **disconnect**

Gatt server callback param of ESP_GATTS_DISCONNECT_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_open_evt_param* **open**

Gatt server callback param of ESP_GATTS_OPEN_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_cancel_open_evt_param* **cancel_open**

Gatt server callback param of ESP_GATTS_CANCEL_OPEN_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_close_evt_param* **close**

Gatt server callback param of ESP_GATTS_CLOSE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_congest_evt_param* **congest**

Gatt server callback param of ESP_GATTS_CONGEST_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_rsp_evt_param* **rsp**

Gatt server callback param of ESP_GATTS_RESPONSE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_add_attr_tab_evt_param* **add_attr_tab**

Gatt server callback param of ESP_GATTS_CREAT_ATTR_TAB_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_set_attr_val_evt_param* **set_attr_val**

Gatt server callback param of ESP_GATTS_SET_ATTR_VAL_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_send_service_change_evt_param* **service_change**

Gatt server callback param of ESP_GATTS_SEND_SERVICE_CHANGE_EVT

struct **gatts_add_attr_tab_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_CREAT_ATTR_TAB_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

esp_bt_uuid_t **svc_uuid**

Service uuid type

uint8_t **svc_inst_id**

Service id

uint16_t **num_handle**

The number of the attribute handle to be added to the gatts database

uint16_t ***handles**

The number to the handles

struct **gatts_add_char_descr_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_ADD_CHAR_DESCR_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **attr_handle**

Descriptor attribute handle

uint16_t **service_handle**

Service attribute handle

esp_bt_uuid_t **descr_uuid**

Characteristic descriptor uuid

struct **gatts_add_char_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_ADD_CHAR_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **attr_handle**

Characteristic attribute handle

uint16_t **service_handle**

Service attribute handle

esp_bt_uuid_t **char_uuid**

Characteristic uuid

struct **gatts_add_incl_srvc_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_ADD_INCL_SRVC_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **attr_handle**

Included service attribute handle

uint16_t **service_handle**

Service attribute handle

struct **gatts_cancel_open_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_CANCEL_OPEN_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

struct **gatts_close_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_CLOSE_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

struct **gatts_conf_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_CONF_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

uint16_t **handle**

attribute handle

uint16_t **len**

The indication or notification value length, len is valid when send notification or indication failed

uint8_t ***value**

The indication or notification value , value is valid when send notification or indication failed

struct **gatts_congest_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_LISTEN_EVT.

ESP_GATTS_CONGEST_EVT

Public Members

uint16_t **conn_id**

Connection id

bool **congested**

Congested or not

struct **gatts_connect_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_CONNECT_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint8_t **link_role**

Link role : master role = 0 ; slave role = 1

esp_bd_addr_t **remote_bda**

Remote bluetooth device address

esp_gatt_conn_params_t **conn_params**

current Connection parameters

esp_ble_addr_type_t **ble_addr_type**

Remote BLE device address type

uint16_t **conn_handle**

HCI connection handle

struct **gatts_create_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_UNREG_EVT.

ESP_GATTS_CREATE_EVT

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **service_handle**

Service attribute handle

esp_gatt_srvc_id_t **service_id**

Service id, include service uuid and other information

struct **gatts_delete_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_DELETE_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **service_handle**

Service attribute handle

struct **gatts_disconnect_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_DISCONNECT_EVT.

Public Members

uint16_t **conn_id**

Connection id

esp_bd_addr_t **remote_bda**

Remote bluetooth device address

esp_gatt_conn_reason_t **reason**

Indicate the reason of disconnection

struct **gatts_exec_write_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_EXEC_WRITE_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint32_t **trans_id**

Transfer id

esp_bd_addr_t **bda**

The bluetooth device address which been written

uint8_t **exec_write_flag**

Execute write flag

struct **gatts_mtu_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_MTU_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint16_t **mtu**

MTU size

struct **gatts_open_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_OPEN_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

struct **gatts_read_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_READ_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint32_t **trans_id**

Transfer id

esp_bd_addr_t **bda**

The bluetooth device address which been read

uint16_t **handle**

The attribute handle

uint16_t **offset**

Offset of the value, if the value is too long

bool **is_long**

The value is too long or not

bool **need_rsp**

The read operation need to do response

struct **gatts_reg_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_REG_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **app_id**

Application id which input in register API

struct **gatts_rsp_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_RESPONSE_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **handle**

Attribute handle which send response

struct **gatts_send_service_change_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_SEND_SERVICE_CHANGE_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

struct **gatts_set_attr_val_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_SET_ATTR_VAL_EVT.

Public Members

uint16_t **srvc_handle**

The service handle

uint16_t **attr_handle**

The attribute handle

esp_gatt_status_t **status**

Operation status

struct **gatts_start_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_START_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **service_handle**

Service attribute handle

struct **gatts_stop_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_STOP_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **service_handle**

Service attribute handle

struct **gatts_write_evt_param**

#include <esp_gatts_api.h> ESP_GATTS_WRITE_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint32_t **trans_id**

Transfer id

esp_bd_addr_t **bda**

The bluetooth device address which been written

uint16_t **handle**

The attribute handle

uint16_t **offset**

Offset of the value, if the value is too long

bool **need_rsp**

The write operation need to do response

bool **is_prep**

This write operation is prepare write

uint16_t **len**

The write attribute value length

uint8_t ***value**

The write attribute value

Macros

ESP_GATT_PREP_WRITE_CANCEL

Prepare write flag to indicate cancel prepare write

ESP_GATT_PREP_WRITE_EXEC

Prepare write flag to indicate execute prepare write

Type Definitions

typedef void (***esp_gatts_cb_t**)(*esp_gatts_cb_event_t* event, *esp_gatt_if_t* gatts_if, *esp_ble_gatts_cb_param_t* *param)

GATT Server callback function type.

Param event : Event type
Param gatts_if : GATT server access interface, normally different gatts_if correspond to different profile
Param param : Point to callback parameter, currently is union type

Enumerations

enum **esp_gatts_cb_event_t**

GATT Server callback function events.

Values:

enumerator **ESP_GATTS_REG_EVT**

When register application id, the event comes

enumerator **ESP_GATTS_READ_EVT**

When gatt client request read operation, the event comes

enumerator **ESP_GATTS_WRITE_EVT**

When gatt client request write operation, the event comes

enumerator **ESP_GATTS_EXEC_WRITE_EVT**

When gatt client request execute write, the event comes

enumerator **ESP_GATTS_MTU_EVT**

When set mtu complete, the event comes

enumerator **ESP_GATTS_CONF_EVT**

When receive confirm, the event comes

enumerator **ESP_GATTS_UNREG_EVT**

When unregister application id, the event comes

enumerator **ESP_GATTS_CREATE_EVT**

When create service complete, the event comes

enumerator **ESP_GATTS_ADD_INCL_SRVC_EVT**

When add included service complete, the event comes

enumerator **ESP_GATTS_ADD_CHAR_EVT**

When add characteristic complete, the event comes

enumerator **ESP_GATTS_ADD_CHAR_DESCR_EVT**

When add descriptor complete, the event comes

enumerator **ESP_GATTS_DELETE_EVT**

When delete service complete, the event comes

enumerator **ESP_GATTS_START_EVT**

When start service complete, the event comes

enumerator **ESP_GATTS_STOP_EVT**

When stop service complete, the event comes

enumerator **ESP_GATTS_CONNECT_EVT**

When gatt client connect, the event comes

enumerator **ESP_GATTS_DISCONNECT_EVT**

When gatt client disconnect, the event comes

enumerator **ESP_GATTS_OPEN_EVT**

When connect to peer, the event comes

enumerator **ESP_GATTS_CANCEL_OPEN_EVT**

When disconnect from peer, the event comes

enumerator **ESP_GATTS_CLOSE_EVT**

When gatt server close, the event comes

enumerator **ESP_GATTS_LISTEN_EVT**

When gatt listen to be connected the event comes

enumerator **ESP_GATTS_CONGEST_EVT**

When congest happen, the event comes

enumerator **ESP_GATTS_RESPONSE_EVT**

When gatt send response complete, the event comes

enumerator **ESP_GATTS_CREAT_ATTR_TAB_EVT**

When gatt create table complete, the event comes

enumerator **ESP_GATTS_SET_ATTR_VAL_EVT**

When gatt set attr value complete, the event comes

enumerator **ESP_GATTS_SEND_SERVICE_CHANGE_EVT**

When gatt send service change indication complete, the event comes

GATT CLIENT API

Application Example Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a GATT client demo and its tutorial. This demo can scan for devices, connect to the GATT server and discover its services.
  - bluetooth/bluedroid/ble/gatt_client
  - GATT Client Example Walkthrough
- This is a multiple connection demo and its tutorial. This demo can connect to multiple GATT server devices and discover their services.
  - bluetooth/bluedroid/ble/gattc_multi_connect
  - GATT Client Multi-connection Example Walkthrough
- This is a BLE SPP-Like demo. This demo, which acts as a GATT client, can receive data from UART and then send the data to the peer device automatically.
  - bluetooth/bluedroid/ble/ble_spp_client

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_gattc_api.h

Functions

esp_err_t **esp_ble_gattc_register_callback**(*esp_gattc_cb_t* callback)

This function is called to register application callbacks with GATTC module.

参数 **callback** –**[in]** : pointer to the application callback function.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_app_register**(uint16_t app_id)

This function is called to register application callbacks with GATTC module.

参数 **app_id** –**[in]** : Application Identify (UUID), for different application

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_app_unregister**(*esp_gatt_if_t* gattc_if)

This function is called to unregister an application from GATTC module.

参数 **gattc_if** –**[in]** Gatt client access interface.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_open**(*esp_gatt_if_t* gattc_if, *esp_bd_addr_t* remote_bda, *esp_ble_addr_type_t* remote_addr_type, bool is_direct)

Open a direct connection or add a background auto connection.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **remote_bda** –**[in]** remote device bluetooth device address.
- **remote_addr_type** –**[in]** remote device bluetooth device the address type.
- **is_direct** –**[in]** direct connection or background auto connection(by now, background auto connection is not supported).

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_aux_open**(*esp_gatt_if_t* gattc_if, *esp_bd_addr_t* remote_bda, *esp_ble_addr_type_t* remote_addr_type, bool is_direct)

esp_err_t **esp_ble_gattc_close**(*esp_gatt_if_t* gattc_if, uint16_t conn_id)

Close the virtual connection to the GATT server. gattc may have multiple virtual GATT server connections when multiple app_id registered, this API only close one virtual GATT server connection. if there exist other virtual GATT server connections, it does not disconnect the physical connection. if you want to disconnect the physical connection directly, you can use esp_ble_gap_disconnect(esp_bd_addr_t remote_device).

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID to be closed.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_send_mtu_req**(*esp_gatt_if_t* gattc_if, uint16_t conn_id)

Configure the MTU size in the GATT channel. This can be done only once per connection. Before using, use esp_ble_gatt_set_local_mtu() to configure the local MTU size.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_search_service**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, *esp_bt_uuid_t* *filter_uuid)

This function is called to get service from local cache. This function report service search result by a callback event, and followed by a service search complete event.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID.
- **filter_uuid** –**[in]** a UUID of the service application is interested in. If Null, discover for all services.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_service**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, *esp_bt_uuid_t* *svc_uuid, *esp_gattc_service_elem_t* *result, uint16_t *count, uint16_t offset)

Find all the service with the given service uuid in the gattc cache, if the svc_uuid is NULL, find all the service. Note: It just get service from local cache, won't get from remote devices. If want to get it from remote device, need to used the esp_ble_gattc_cache_refresh, then call esp_ble_gattc_get_service again.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID which identify the server.
- **svc_uuid** –**[in]** the pointer to the service uuid.
- **result** –**[out]** The pointer to the service which has been found in the gattc cache.
- **count** –**[inout]** input the number of service want to find, it will output the number of service has been found in the gattc cache with the given service uuid.
- **offset** –**[in]** Offset of the service position to get.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_all_char**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, *esp_gattc_char_elem_t* *result, uint16_t *count, uint16_t offset)

Find all the characteristic with the given service in the gattc cache Note: It just get characteristic from local cache, won't get from remote devices.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID which identify the server.
- **start_handle** –**[in]** the attribute start handle.
- **end_handle** –**[in]** the attribute end handle
- **result** –**[out]** The pointer to the characteristic in the service.
- **count** –**[inout]** input the number of characteristic want to find, it will output the number of characteristic has been found in the gattc cache with the given service.
- **offset** –**[in]** Offset of the characteristic position to get.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_all_descr**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t char_handle, *esp_gattc_descr_elem_t* *result, uint16_t *count, uint16_t offset)

Find all the descriptor with the given characteristic in the gattc cache Note: It just get descriptor from local cache, won't get from remote devices.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID which identify the server.
- **char_handle** –**[in]** the given characteristic handle
- **result** –**[out]** The pointer to the descriptor in the characteristic.
- **count** –**[inout]** input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.
- **offset** –**[in]** Offset of the descriptor position to get.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_char_by_uuid**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, *esp_bt_uuid_t* char_uuid, *esp_gattc_char_elem_t* *result, uint16_t *count)

Find the characteristic with the given characteristic uuid in the gattc cache Note: It just get characteristic from local cache, won't get from remote devices.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID which identify the server.
- **start_handle** –**[in]** the attribute start handle
- **end_handle** –**[in]** the attribute end handle
- **char_uuid** –**[in]** the characteristic uuid
- **result** –**[out]** The pointer to the characteristic in the service.
- **count** –**[inout]** input the number of characteristic want to find, it will output the number of characteristic has been found in the gattc cache with the given service.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_descr_by_uuid**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, *esp_bt_uuid_t* char_uuid, *esp_bt_uuid_t* descr_uuid, *esp_gattc_descr_elem_t* *result, uint16_t *count)

Find the descriptor with the given characteristic uuid in the gattc cache Note: It just get descriptor from local

cache, won’t get from remote devices.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID which identify the server.
- **start_handle** –**[in]** the attribute start handle
- **end_handle** –**[in]** the attribute end handle
- **char_uuid** –**[in]** the characteristic uuid.
- **descr_uuid** –**[in]** the descriptor uuid.
- **result** –**[out]** The pointer to the descriptor in the given characteristic.
- **count** –**[inout]** input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_descr_by_char_handle**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t char_handle, *esp_bt_uuid_t* descr_uuid, *esp_gattc_descr_elem_t* *result, uint16_t *count)

Find the descriptor with the given characteristic handle in the gattc cache Note: It just get descriptor from local cache, won’t get from remote devices.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID which identify the server.
- **char_handle** –**[in]** the characteristic handle.
- **descr_uuid** –**[in]** the descriptor uuid.
- **result** –**[out]** The pointer to the descriptor in the given characteristic.
- **count** –**[inout]** input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_include_service**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, *esp_bt_uuid_t* *incl_uuid, *esp_gattc_incl_svc_elem_t* *result, uint16_t *count)

Find the include service with the given service handle in the gattc cache Note: It just get include service from local cache, won’t get from remote devices.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID which identify the server.
- **start_handle** –**[in]** the attribute start handle
- **end_handle** –**[in]** the attribute end handle
- **incl_uuid** –**[in]** the include service uuid
- **result** –**[out]** The pointer to the include service in the given service.
- **count** –**[inout]** input the number of include service want to find, it will output the number of include service has been found in the gattc cache with the given service.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_attr_count**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, *esp_gatt_db_attr_type_t* type, uint16_t start_handle, uint16_t end_handle, uint16_t char_handle, uint16_t *count)

Find the attribute count with the given service or characteristic in the gattc cache.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** connection ID which identify the server.
- **type** –**[in]** the attribute type.
- **start_handle** –**[in]** the attribute start handle, if the type is ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore
- **end_handle** –**[in]** the attribute end handle, if the type is ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore
- **char_handle** –**[in]** the characteristic handle, this parameter valid when the type is ESP_GATT_DB_DESCRIPTOR. If the type isn't ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore.
- **count** –**[out]** output the number of attribute has been found in the gattc cache with the given attribute type.

返回

- ESP_OK: success
- other: failed

esp_gatt_status_t **esp_ble_gattc_get_db**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, *esp_gattc_db_elem_t* *db, uint16_t *count)

This function is called to get the GATT database. Note: It just get attribute data base from local cache, won't get from remote devices.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **start_handle** –**[in]** the attribute start handle
- **end_handle** –**[in]** the attribute end handle
- **conn_id** –**[in]** connection ID which identify the server.
- **db** –**[in]** output parameter which will contain the GATT database copy. Caller is responsible for freeing it.
- **count** –**[in]** number of elements in database.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_read_char**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t handle, *esp_gatt_auth_req_t* auth_req)

This function is called to read a service's characteristics of the given characteristic handle.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID.
- **handle** –**[in]** : characteritic handle to read.
- **auth_req** –**[in]** : authenticate request type

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_read_by_type**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, *esp_bt_uuid_t* *uuid, *esp_gatt_auth_req_t* auth_req)

This function is called to read a service's characteristics of the given characteristic UUID.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID.
- **start_handle** –**[in]** : the attribute start handle.
- **end_handle** –**[in]** : the attribute end handle
- **uuid** –**[in]** : The UUID of attribute which will be read.

- **auth_req** –**[in]** : authenticate request type

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_read_multiple**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, *esp_gattc_multi_t* *read_multi, *esp_gatt_auth_req_t* auth_req)

This function is called to read multiple characteristic or characteristic descriptors.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID.
- **read_multi** –**[in]** : pointer to the read multiple parameter.
- **auth_req** –**[in]** : authenticate request type

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_read_char_descr**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t handle, *esp_gatt_auth_req_t* auth_req)

This function is called to read a characteristics descriptor.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID.
- **handle** –**[in]** : descriptor handle to read.
- **auth_req** –**[in]** : authenticate request type

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_write_char**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t handle, uint16_t value_len, uint8_t *value, *esp_gatt_write_type_t* write_type, *esp_gatt_auth_req_t* auth_req)

This function is called to write characteristic value.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID.
- **handle** –**[in]** : characteristic handle to write.
- **value_len** –**[in]** length of the value to be written.
- **value** –**[in]** : the value to be written.
- **write_type** –**[in]** : the type of attribute write operation.
- **auth_req** –**[in]** : authentication request.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_write_char_descr**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t handle, uint16_t value_len, uint8_t *value, *esp_gatt_write_type_t* write_type, *esp_gatt_auth_req_t* auth_req)

This function is called to write characteristic descriptor value.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID
- **handle** –**[in]** : descriptor handle to write.
- **value_len** –**[in]** length of the value to be written.
- **value** –**[in]** : the value to be written.
- **write_type** –**[in]** : the type of attribute write operation.
- **auth_req** –**[in]** : authentication request.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_prepare_write**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t handle, uint16_t offset, uint16_t value_len, uint8_t *value, *esp_gatt_auth_req_t* auth_req)

This function is called to prepare write a characteristic value.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID.
- **handle** –**[in]** : characteristic handle to prepare write.
- **offset** –**[in]** : offset of the write value.
- **value_len** –**[in]** length of the value to be written.
- **value** –**[in]** : the value to be written.
- **auth_req** –**[in]** : authentication request.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_prepare_write_char_descr**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, uint16_t handle, uint16_t offset, uint16_t value_len, uint8_t *value, *esp_gatt_auth_req_t* auth_req)

This function is called to prepare write a characteristic descriptor value.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID.
- **handle** –**[in]** : characteristic descriptor handle to prepare write.
- **offset** –**[in]** : offset of the write value.
- **value_len** –**[in]** length of the value to be written.
- **value** –**[in]** : the value to be written.
- **auth_req** –**[in]** : authentication request.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_execute_write**(*esp_gatt_if_t* gattc_if, uint16_t conn_id, bool is_execute)

This function is called to execute write a prepare write sequence.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **conn_id** –**[in]** : connection ID.
- **is_execute** –**[in]** : execute or cancel.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_register_for_notify**(*esp_gatt_if_t* gattc_if, *esp_bd_addr_t* server_bda, uint16_t handle)

This function is called to register for notification of a service.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **server_bda** –**[in]** : target GATT server.
- **handle** –**[in]** : GATT characteristic handle.

返回

- ESP_OK: registration succeeds
- other: failed

esp_err_t **esp_ble_gattc_unregister_for_notify**(*esp_gatt_if_t* gattc_if, *esp_bd_addr_t* server_bda, uint16_t handle)

This function is called to de-register for notification of a service.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **server_bda** –**[in]** : target GATT server.
- **handle** –**[in]** : GATT characteristic handle.

返回

- ESP_OK: unregister succeeds
- other: failed

esp_err_t **esp_ble_gattc_cache_refresh**(*esp_bd_addr_t* remote_bda)

Refresh the server cache store in the gattc stack of the remote device. If the device is connected, this API will restart the discovery of service information of the remote device.

参数 **remote_bda** –**[in]** remote device BD address.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_cache_assoc**(*esp_gatt_if_t* gattc_if, *esp_bd_addr_t* src_addr, *esp_bd_addr_t* assoc_addr, bool is_assoc)

Add or delete the associated address with the source address. Note: The role of this API is mainly when the client side has stored a server-side database, when it needs to connect another device, but the device ' s attribute database is the same as the server database stored on the client-side, calling this API can use the database that the device has stored used as the peer server database to reduce the attribute database search and discovery process and speed up the connection time. The associated address mains that device want to used the database has stored in the local cache. The source address mains that device want to share the database to the associated address device.

参数

- **gattc_if** –**[in]** Gatt client access interface.
- **src_addr** –**[in]** the source address which provide the attribute table.
- **assoc_addr** –**[in]** the associated device address which went to share the attribute table with the source address.
- **is_assoc** –**[in]** true add the associated device address, false remove the associated device address.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_cache_get_addr_list**(*esp_gatt_if_t* gattc_if)

Get the address list which has store the attribute table in the gattc cache. There will callback ESP_GATTC_GET_ADDR_LIST_EVT event when get address list complete.

参数 **gattc_if** –**[in]** Gatt client access interface.

返回

- ESP_OK: success
- other: failed

esp_err_t **esp_ble_gattc_cache_clean**(*esp_bd_addr_t* remote_bda)

Clean the service cache of this device in the gattc stack,.

参数 **remote_bda** –**[in]** remote device BD address.

返回

- ESP_OK: success
- other: failed

Unions

union **esp_ble_gattc_cb_param_t**

#include <esp_gattc_api.h> Gatt client callback parameters union.

Public Members

struct *esp_ble_gattc_cb_param_t*::*gattc_reg_evt_param* **reg**

Gatt client callback param of ESP_GATTC_REG_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_open_evt_param* **open**

Gatt client callback param of ESP_GATTC_OPEN_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_close_evt_param* **close**

Gatt client callback param of ESP_GATTC_CLOSE_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_cfg_mtu_evt_param* **cfg_mtu**

Gatt client callback param of ESP_GATTC_CFG_MTU_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_search_cmpl_evt_param* **search_cmpl**

Gatt client callback param of ESP_GATTC_SEARCH_CMPL_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_search_res_evt_param* **search_res**

Gatt client callback param of ESP_GATTC_SEARCH_RES_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_read_char_evt_param* **read**

Gatt client callback param of ESP_GATTC_READ_CHAR_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_write_evt_param* **write**

Gatt client callback param of ESP_GATTC_WRITE_DESCR_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_exec_cmpl_evt_param* **exec_cmpl**

Gatt client callback param of ESP_GATTC_EXEC_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_notify_evt_param* **notify**

Gatt client callback param of ESP_GATTC_NOTIFY_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_srvc_chg_evt_param* **srvc_chg**

Gatt client callback param of ESP_GATTC_SRVC_CHG_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_congest_evt_param* **congest**

Gatt client callback param of ESP_GATTC_CONGEST_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_reg_for_notify_evt_param* **reg_for_notify**

Gatt client callback param of ESP_GATTC_REG_FOR_NOTIFY_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_unreg_for_notify_evt_param* **unreg_for_notify**

Gatt client callback param of ESP_GATTC_UNREG_FOR_NOTIFY_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_connect_evt_param* **connect**

Gatt client callback param of ESP_GATTC_CONNECT_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_disconnect_evt_param* **disconnect**

Gatt client callback param of ESP_GATTC_DISCONNECT_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_set_assoc_addr_cmp_evt_param* **set_assoc_cmp**

Gatt client callback param of ESP_GATTC_SET_ASSOC_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_get_addr_list_evt_param* **get_addr_list**

Gatt client callback param of ESP_GATTC_GET_ADDR_LIST_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_queue_full_evt_param* **queue_full**

Gatt client callback param of ESP_GATTC_QUEUE_FULL_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_dis_srvc_cmpl_evt_param* **dis_srvc_cmpl**

Gatt client callback param of ESP_GATTC_DIS_SRVC_CMPL_EVT

struct **gattc_cfg_mtu_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_CFG_MTU_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

uint16_t **mtu**

MTU size

struct **gattc_close_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_CLOSE_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

esp_bd_addr_t **remote_bda**

Remote bluetooth device address

esp_gatt_conn_reason_t **reason**

The reason of gatt connection close

struct **gattc_congest_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_CONGEST_EVT.

Public Members

uint16_t **conn_id**

Connection id

bool **congested**

Congested or not

struct **gattc_connect_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_CONNECT_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint8_t **link_role**

Link role : master role = 0 ; slave role = 1

esp_bd_addr_t **remote_bda**

Remote bluetooth device address

esp_gatt_conn_params_t **conn_params**

current connection parameters

esp_ble_addr_type_t **ble_addr_type**

Remote BLE device address type

uint16_t **conn_handle**

HCI connection handle

struct **gattc_dis_srvc_cmpl_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_DIS_SRVC_CMPL_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

struct **gattc_disconnect_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_DISCONNECT_EVT.

Public Members

esp_gatt_conn_reason_t **reason**

disconnection reason

uint16_t **conn_id**

Connection id

esp_bd_addr_t **remote_bda**

Remote bluetooth device address

struct **gattc_exec_cmpl_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_EXEC_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

struct **gattc_get_addr_list_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_GET_ADDR_LIST_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint8_t **num_addr**

The number of address in the gattc cache address list

esp_bd_addr_t ***addr_list**

The pointer to the address list which has been get from the gattc cache

struct **gattc_notify_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_NOTIFY_EVT.

Public Members

uint16_t **conn_id**
Connection id

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

uint16_t **handle**
The Characteristic or descriptor handle

uint16_t **value_len**
Notify attribute value

uint8_t ***value**
Notify attribute value

bool **is_notify**
True means notify, false means indicate

struct **gattc_open_evt_param**
#include <esp_gattc_api.h> ESP_GATTC_OPEN_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

uint16_t **mtu**
MTU size

struct **gattc_queue_full_evt_param**
#include <esp_gattc_api.h> ESP_GATTC_QUEUE_FULL_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

bool **is_full**

The gattc command queue is full or not

struct **gattc_read_char_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_READ_CHAR_EVT, ESP_GATTC_READ_DESCR_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

uint16_t **handle**

Characteristic handle

uint8_t ***value**

Characteristic value

uint16_t **value_len**

Characteristic value length

struct **gattc_reg_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_REG_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **app_id**

Application id which input in register API

struct **gattc_reg_for_notify_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_REG_FOR_NOTIFY_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **handle**

The characteristic or descriptor handle

struct **gattc_search_cmpl_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_SEARCH_CMPL_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

esp_service_source_t **searched_service_source**

The source of the service information

struct **gattc_search_res_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_SEARCH_RES_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint16_t **start_handle**

Service start handle

uint16_t **end_handle**

Service end handle

esp_gatt_id_t **srvc_id**

Service id, include service uuid and other information

bool **is_primary**

True if this is the primary service

struct **gattc_set_assoc_addr_cmp_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_SET_ASSOC_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

struct **gattc_srvc_chg_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_SRVC_CHG_EVT.

Public Members

esp_bd_addr_t **remote_bda**

Remote bluetooth device address

struct **gattc_unreg_for_notify_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_UNREG_FOR_NOTIFY_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **handle**

The characteristic or descriptor handle

struct **gattc_write_evt_param**

#include <esp_gattc_api.h> ESP_GATTC_WRITE_CHAR_EVT, ESP_GATTC_PREP_WRITE_EVT, ESP_GATTC_WRITE_DESCR_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

uint16_t **handle**

The Characteristic or descriptor handle

uint16_t **offset**

The prepare write offset, this value is valid only when prepare write

Type Definitions

typedef void (***esp_gattc_cb_t**)(*esp_gattc_cb_event_t* event, *esp_gatt_if_t* gattc_if, *esp_ble_gattc_cb_param_t* *param)

GATT Client callback function type.

Param event : Event type
Param gattc_if : GATT client access interface, normally different gattc_if correspond to different profile
Param param : Point to callback parameter, currently is union type

Enumerations

enum **esp_gattc_cb_event_t**

GATT Client callback function events.

Values:

enumerator **ESP_GATTC_REG_EVT**

When GATT client is registered, the event comes

enumerator **ESP_GATTC_UNREG_EVT**

When GATT client is unregistered, the event comes

enumerator **ESP_GATTC_OPEN_EVT**

When GATT virtual connection is set up, the event comes

enumerator **ESP_GATTC_READ_CHAR_EVT**

When GATT characteristic is read, the event comes

enumerator **ESP_GATTC_WRITE_CHAR_EVT**

When GATT characteristic write operation completes, the event comes

enumerator **ESP_GATTC_CLOSE_EVT**

When GATT virtual connection is closed, the event comes

enumerator **ESP_GATTC_SEARCH_CMPL_EVT**

When GATT service discovery is completed, the event comes

enumerator **ESP_GATTC_SEARCH_RES_EVT**

When GATT service discovery result is got, the event comes

enumerator **ESP_GATTC_READ_DESCR_EVT**

When GATT characteristic descriptor read completes, the event comes

enumerator **ESP_GATTC_WRITE_DESCR_EVT**

When GATT characteristic descriptor write completes, the event comes

enumerator **ESP_GATTC_NOTIFY_EVT**

When GATT notification or indication arrives, the event comes

enumerator **ESP_GATTC_PREP_WRITE_EVT**

When GATT prepare-write operation completes, the event comes

enumerator **ESP_GATTC_EXEC_EVT**

When write execution completes, the event comes

enumerator **ESP_GATTC_ACL_EVT**

When ACL connection is up, the event comes

enumerator **ESP_GATTC_CANCEL_OPEN_EVT**

When GATT client ongoing connection is cancelled, the event comes

enumerator **ESP_GATTC_SRVC_CHG_EVT**

When “service changed” occurs, the event comes

enumerator **ESP_GATTC_ENC_CMPL_CB_EVT**

When encryption procedure completes, the event comes

enumerator **ESP_GATTC_CFG_MTU_EVT**

When configuration of MTU completes, the event comes

enumerator **ESP_GATTC_ADV_DATA_EVT**

When advertising of data, the event comes

enumerator **ESP_GATTC_MULT_ADV_ENB_EVT**

When multi-advertising is enabled, the event comes

enumerator **ESP_GATTC_MULT_ADV_UPD_EVT**

When multi-advertising parameters are updated, the event comes

enumerator **ESP_GATTC_MULT_ADV_DATA_EVT**

When multi-advertising data arrives, the event comes

enumerator **ESP_GATTC_MULT_ADV_DIS_EVT**

When multi-advertising is disabled, the event comes

enumerator **ESP_GATTC_CONGEST_EVT**

When GATT connection congestion comes, the event comes

enumerator **ESP_GATTC_BTH_SCAN_ENB_EVT**

When batch scan is enabled, the event comes

enumerator **ESP_GATTC_BTH_SCAN_CFG_EVT**

When batch scan storage is configured, the event comes

enumerator **ESP_GATTC_BTH_SCAN_RD_EVT**

When Batch scan read event is reported, the event comes

enumerator **ESP_GATTC_BTH_SCAN_THR_EVT**

When Batch scan threshold is set, the event comes

enumerator **ESP_GATTC_BTH_SCAN_PARAM_EVT**

When Batch scan parameters are set, the event comes

enumerator **ESP_GATTC_BTH_SCAN_DIS_EVT**

When Batch scan is disabled, the event comes

enumerator **ESP_GATTC_SCAN_FLT_CFG_EVT**

When Scan filter configuration completes, the event comes

enumerator **ESP_GATTC_SCAN_FLT_PARAM_EVT**

When Scan filter parameters are set, the event comes

enumerator **ESP_GATTC_SCAN_FLT_STATUS_EVT**

When Scan filter status is reported, the event comes

enumerator **ESP_GATTC_ADV_VSC_EVT**

When advertising vendor spec content event is reported, the event comes

enumerator **ESP_GATTC_REG_FOR_NOTIFY_EVT**

When register for notification of a service completes, the event comes

enumerator **ESP_GATTC_UNREG_FOR_NOTIFY_EVT**

When unregister for notification of a service completes, the event comes

enumerator **ESP_GATTC_CONNECT_EVT**

When the ble physical connection is set up, the event comes

enumerator **ESP_GATTC_DISCONNECT_EVT**

When the ble physical connection disconnected, the event comes

enumerator **ESP_GATTC_READ_MULTIPLE_EVT**

When the ble characteristic or descriptor multiple complete, the event comes

enumerator **ESP_GATTC_QUEUE_FULL_EVT**

When the gattc command queue full, the event comes

enumerator **ESP_GATTC_SET_ASSOC_EVT**

When the ble gattc set the associated address complete, the event comes

enumerator **ESP_GATTC_GET_ADDR_LIST_EVT**

When the ble get gattc address list in cache finish, the event comes

enumerator **ESP_GATTC_DIS_SRVC_CMPL_EVT**

When the ble discover service complete, the event comes

BLUFI API

Overview BLUFI is a profile based GATT to config ESP32 WIFI to connect/disconnect AP or setup a softap and etc. Use should concern these things:

1. The event sent from profile. Then you need to do something as the event indicate.
2. Security reference. You can write your own Security functions such as symmetrical encryption/decryption and checksum functions. Even you can define the "Key Exchange/Negotiation" procedure.

Application Example Check bluetooth folder in ESP-IDF examples, which contains the following application:

- This is the BLUFI demo. This demo can set ESP32's wifi to softap/station/softap&station mode and config wifi connections - bluetooth/blufi

API Reference

Header File

- components/bt/common/api/include/api/esp_blufi_api.h

Functions

esp_err_t **esp_blufi_register_callbacks**(*esp_blufi_callbacks_t* *callbacks)

This function is called to receive blufi callback event.

参数 **callbacks** –**[in]** callback functions
返回 ESP_OK - success, other - failed

esp_err_t **esp_blufi_profile_init**(void)

This function is called to initialize blufi_profile.

返回 ESP_OK - success, other - failed

esp_err_t **esp_blufi_profile_deinit**(void)

This function is called to de-initialize blufi_profile.

返回 ESP_OK - success, other - failed

esp_err_t **esp_blufi_send_wifi_conn_report**(*wifi_mode_t* opmode, *esp_blufi_sta_conn_state_t* sta_conn_state, uint8_t softap_conn_num, *esp_blufi_extra_info_t* *extra_info)

This function is called to send wifi connection report.

参数

- **opmode** –: wifi opmode
- **sta_conn_state** –: station is already in connection or not
- **softap_conn_num** –: softap connection number
- **extra_info** –: extra information, such as sta_ssid, softap_ssid and etc.

返回 ESP_OK - success, other - failed

esp_err_t **esp_blufi_send_wifi_list**(uint16_t apCount, *esp_blufi_ap_record_t* *list)

This function is called to send wifi list.

参数

- **apCount** –: wifi list count
- **list** –: wifi list

返回 ESP_OK - success, other - failed

uint16_t **esp_blufi_get_version**(void)

Get BLUFI profile version.

返回 Most 8bit significant is Great version, Least 8bit is Sub version

esp_err_t **esp_blufi_send_error_info**(*esp_blufi_error_state_t* state)

This function is called to send blufi error information.

参数 **state** –: error state
返回 ESP_OK - success, other - failed

esp_err_t **esp_blufi_send_custom_data**(uint8_t *data, uint32_t data_len)

This function is called to custom data.

参数

- **data** –: custom data value
- **data_len** –: the length of custom data

返回 ESP_OK - success, other - failed

Unions

union **esp_blufi_cb_param_t**

#include <esp_blufi_api.h> BLUFI callback parameters union.

Public Members

struct *esp_blufi_cb_param_t*::*blufi_init_finish_evt_param* **init_finish**
Blufi callback param of ESP_BLUFI_EVENT_INIT_FINISH

struct *esp_blufi_cb_param_t*::*blufi_deinit_finish_evt_param* **deinit_finish**
Blufi callback param of ESP_BLUFI_EVENT_DEINIT_FINISH

struct *esp_blufi_cb_param_t*::*blufi_set_wifi_mode_evt_param* **wifi_mode**
Blufi callback param of ESP_BLUFI_EVENT_INIT_FINISH

struct *esp_blufi_cb_param_t*::*blufi_connect_evt_param* **connect**
Blufi callback param of ESP_BLUFI_EVENT_CONNECT

struct *esp_blufi_cb_param_t*::*blufi_disconnect_evt_param* **disconnect**
Blufi callback param of ESP_BLUFI_EVENT_DISCONNECT

struct *esp_blufi_cb_param_t*::*blufi_recv_sta_bssid_evt_param* **sta_bssid**
Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_BSSID

struct *esp_blufi_cb_param_t*::*blufi_recv_sta_ssid_evt_param* **sta_ssid**
Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_SSID

struct *esp_blufi_cb_param_t*::*blufi_recv_sta_passwd_evt_param* **sta_passwd**
Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_PASSWD

struct *esp_blufi_cb_param_t*::*blufi_recv_softap_ssid_evt_param* **softap_ssid**
Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_SSID

struct *esp_blufi_cb_param_t*::*blufi_recv_softap_passwd_evt_param* **softap_passwd**
Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD

struct *esp_blufi_cb_param_t*::*blufi_recv_softap_max_conn_num_evt_param* **softap_max_conn_num**
Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM

struct *esp_blufi_cb_param_t*::*blufi_recv_softap_auth_mode_evt_param* **softap_auth_mode**
Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE

struct *esp_blufi_cb_param_t*::*blufi_recv_softap_channel_evt_param* **softap_channel**
Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL

struct *esp_blufi_cb_param_t*::*blufi_recv_username_evt_param* **username**
Blufi callback param of ESP_BLUFI_EVENT_RECV_USERNAME

struct *esp_blufi_cb_param_t*::*blufi_recv_ca_evt_param* **ca**
Blufi callback param of ESP_BLUFI_EVENT_RECV_CA_CERT

struct *esp_blufi_cb_param_t*::*blufi_recv_client_cert_evt_param* **client_cert**
Blufi callback param of ESP_BLUFI_EVENT_RECV_CLIENT_CERT

struct *esp_blufi_cb_param_t*::*blufi_recv_server_cert_evt_param* **server_cert**

Blufi callback param of ESP_BLUFI_EVENT_RECV_SERVER_CERT

struct *esp_blufi_cb_param_t*::*blufi_recv_client_pkey_evt_param* **client_pkey**

Blufi callback param of ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY

struct *esp_blufi_cb_param_t*::*blufi_recv_server_pkey_evt_param* **server_pkey**

Blufi callback param of ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY

struct *esp_blufi_cb_param_t*::*blufi_get_error_evt_param* **report_error**

Blufi callback param of ESP_BLUFI_EVENT_REPORT_ERROR

struct *esp_blufi_cb_param_t*::*blufi_recv_custom_data_evt_param* **custom_data**

Blufi callback param of ESP_BLUFI_EVENT_RECV_CUSTOM_DATA

struct **blufi_connect_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_CONNECT.

Public Members

esp_blufi_bd_addr_t **remote_bda**

Blufi Remote bluetooth device address

uint8_t **server_if**

server interface

uint16_t **conn_id**

Connection id

struct **blufi_deinit_finish_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_DEINIT_FINISH.

Public Members

esp_blufi_deinit_state_t **state**

De-initial status

struct **blufi_disconnect_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_DISCONNECT.

Public Members

esp_blufi_bd_addr_t **remote_bda**

Blufi Remote bluetooth device address

struct **blufi_get_error_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_REPORT_ERROR.

Public Members

esp_blufi_error_state_t **state**

Blufi error state

struct **blufi_init_finish_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_INIT_FINISH.

Public Members

esp_blufi_init_state_t **state**

Initial status

struct **blufi_recv_ca_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CA_CERT.

Public Members

uint8_t ***cert**

CA certificate point

int **cert_len**

CA certificate length

struct **blufi_recv_client_cert_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CLIENT_CERT

Public Members

uint8_t ***cert**

Client certificate point

int **cert_len**

Client certificate length

struct **blufi_recv_client_pkey_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY

Public Members

uint8_t ***pkey**

Client Private Key point, if Client certificate not contain Key

int **pkey_len**

Client Private key length

struct **blufi_recv_custom_data_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CUSTOM_DATA.

Public Members

uint8_t ***data**

Custom data

uint32_t **data_len**

Custom data Length

struct **blufi_recv_server_cert_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SERVER_CERT

Public Members

uint8_t ***cert**

Client certificate point

int **cert_len**

Client certificate length

struct **blufi_recv_server_pkey_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY

Public Members

uint8_t ***pkey**

Client Private Key point, if Client certificate not contain Key

int **pkey_len**

Client Private key length

struct **blufi_recv_softap_auth_mode_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE.

Public Members

wifi_auth_mode_t **auth_mode**

Authentication mode

struct **blufi_recv_softap_channel_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL.

Public Members

uint8_t **channel**

Authentication mode

struct **blufi_recv_softap_max_conn_num_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM.

Public Members

int **max_conn_num**

SSID

struct **blufi_recv_softap_passwd_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD.

Public Members

uint8_t ***passwd**

Password

int **passwd_len**

Password Length

struct **blufi_recv_softap_ssid_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_SSID.

Public Members

uint8_t ***ssid**

SSID

int **ssid_len**

SSID length

struct **blufi_recv_sta_bssid_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_STA_BSSID.

Public Members

uint8_t **bssid**[6]

BSSID

struct **blufi_recv_sta_passwd_evt_param**

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_STA_PASSWD.

Public Members

uint8_t *passwd

Password

int passwd_len

Password Length

struct blufi_recv_sta_ssid_evt_param

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_STA_SSID.

Public Members

uint8_t *ssid

SSID

int ssid_len

SSID length

struct blufi_recv_username_evt_param

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_USERNAME.

Public Members

uint8_t *name

Username point

int name_len

Username length

struct blufi_set_wifi_mode_evt_param

#include <esp_blufi_api.h> ESP_BLUFI_EVENT_SET_WIFI_MODE.

Public Members

wifi_mode_t op_mode

Wifi operation mode

Structures

struct esp_blufi_extra_info_t

BLUFI extra information structure.

Public Members

uint8_t **sta_bssid**[6]
BSSID of station interface

bool **sta_bssid_set**
is BSSID of station interface set

uint8_t ***sta_ssid**
SSID of station interface

int **sta_ssid_len**
length of SSID of station interface

uint8_t ***sta_passwd**
password of station interface

int **sta_passwd_len**
length of password of station interface

uint8_t ***softap_ssid**
SSID of softap interface

int **softap_ssid_len**
length of SSID of softap interface

uint8_t ***softap_passwd**
password of station interface

int **softap_passwd_len**
length of password of station interface

uint8_t **softap_authmode**
authentication mode of softap interface

bool **softap_authmode_set**
is authentication mode of softap interface set

uint8_t **softap_max_conn_num**
max connection number of softap interface

bool **softap_max_conn_num_set**
is max connection number of softap interface set

uint8_t **softap_channel**
channel of softap interface

bool **softap_channel_set**
is channel of softap interface set

uint8_t **sta_max_conn_retry**
max retry of sta establish connection

bool **sta_max_conn_retry_set**
is max retry of sta establish connection set

uint8_t **sta_conn_end_reason**
reason of sta connection end

bool **sta_conn_end_reason_set**
is reason of sta connection end set

int8_t **sta_conn_rssi**
rssi of sta connection

bool **sta_conn_rssi_set**
is rssi of sta connection set

struct **esp_blufi_ap_record_t**
Description of an WiFi AP.

Public Members

uint8_t **ssid**[33]
SSID of AP

int8_t **rssi**
signal strength of AP

struct **esp_blufi_callbacks_t**
BLUFI callback functions type.

Public Members

esp_blufi_event_cb_t **event_cb**
BLUFI event callback

esp_blufi_negotiate_data_handler_t **negotiate_data_handler**
BLUFI negotiate data function for negotiate share key

esp_blufi_encrypt_func_t **encrypt_func**
BLUFI encrypt data function with share key generated by negotiate_data_handler

esp_blufi_decrypt_func_t **decrypt_func**
BLUFI decrypt data function with share key generated by negotiate_data_handler

esp_blufi_checksum_func_t **checksum_func**
BLUFI check sum function (FCS)

Macros

ESP_BLUFI_BD_ADDR_LEN

Bluetooth address length.

Type Definitions

typedef uint8_t **esp_blufi_bd_addr_t**[ESP_BLUFI_BD_ADDR_LEN]

Bluetooth device address.

typedef void (***esp_blufi_event_cb_t**)(*esp_blufi_cb_event_t* event, *esp_blufi_cb_param_t* *param)

BLUFI event callback function type.

Param event : Event type
Param param : Point to callback parameter, currently is union type

typedef void (***esp_blufi_negotiate_data_handler_t**)(uint8_t *data, int len, uint8_t **output_data, int *output_len, bool *need_free)

BLUFI negotiate data handler.

Param data : data from phone
Param len : length of data from phone
Param output_data : data want to send to phone
Param output_len : length of data want to send to phone
Param need_free : output reporting if memory needs to be freed or not *

typedef int (***esp_blufi_encrypt_func_t**)(uint8_t iv8, uint8_t *crypt_data, int crypt_len)

BLUFI encrypt the data after negotiate a share key.

Param iv8 : initial vector(8bit), normally, blufi core will input packet sequence number
Param crypt_data : plain text and encrypted data, the encrypt function must support autochthonous encrypt
Param crypt_len : length of plain text
Return Nonnegative number is encrypted length, if error, return negative number;

typedef int (***esp_blufi_decrypt_func_t**)(uint8_t iv8, uint8_t *crypt_data, int crypt_len)

BLUFI decrypt the data after negotiate a share key.

Param iv8 : initial vector(8bit), normally, blufi core will input packet sequence number
Param crypt_data : encrypted data and plain text, the encrypt function must support autochthonous decrypt
Param crypt_len : length of encrypted text
Return Nonnegative number is decrypted length, if error, return negative number;

typedef uint16_t (***esp_blufi_checksum_func_t**)(uint8_t iv8, uint8_t *data, int len)

BLUFI checksum.

Param iv8 : initial vector(8bit), normally, blufi core will input packet sequence number
Param data : data need to checksum
Param len : length of data

Enumerations

enum **esp_blufi_cb_event_t**

Values:

enumerator **ESP_BLUFI_EVENT_INIT_FINISH**

enumerator **ESP_BLUFI_EVENT_DEINIT_FINISH**

enumerator **ESP_BLUFI_EVENT_SET_WIFI_OPMODE**

enumerator **ESP_BLUFI_EVENT_BLE_CONNECT**

enumerator **ESP_BLUFI_EVENT_BLE_DISCONNECT**

enumerator **ESP_BLUFI_EVENT_REQ_CONNECT_TO_AP**

enumerator **ESP_BLUFI_EVENT_REQ_DISCONNECT_FROM_AP**

enumerator **ESP_BLUFI_EVENT_GET_WIFI_STATUS**

enumerator **ESP_BLUFI_EVENT_DEAUTHENTICATE_STA**

enumerator **ESP_BLUFI_EVENT_RECV_STA_BSSID**

enumerator **ESP_BLUFI_EVENT_RECV_STA_SSID**

enumerator **ESP_BLUFI_EVENT_RECV_STA_PASSWD**

enumerator **ESP_BLUFI_EVENT_RECV_SOFTAP_SSID**

enumerator **ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD**

enumerator **ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM**

enumerator **ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE**

enumerator **ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL**

enumerator **ESP_BLUFI_EVENT_RECV_USERNAME**

enumerator **ESP_BLUFI_EVENT_RECV_CA_CERT**

enumerator **ESP_BLUFI_EVENT_RECV_CLIENT_CERT**

enumerator **ESP_BLUFI_EVENT_RECV_SERVER_CERT**

enumerator **ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY**

enumerator **ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY**

enumerator **ESP_BLUFI_EVENT_RECV_SLAVE_DISCONNECT_BLE**

enumerator **ESP_BLUFI_EVENT_GET_WIFI_LIST**

enumerator **ESP_BLUFI_EVENT_REPORT_ERROR**

enumerator **ESP_BLUFI_EVENT_RECV_CUSTOM_DATA**

enum **esp_blufi_sta_conn_state_t**

BLUFI config status.

Values:

enumerator **ESP_BLUFI_STA_CONN_SUCCESS**

enumerator **ESP_BLUFI_STA_CONN_FAIL**

enumerator **ESP_BLUFI_STA_CONNECTING**

enumerator **ESP_BLUFI_STA_NO_IP**

enum **esp_blufi_init_state_t**

BLUFI init status.

Values:

enumerator **ESP_BLUFI_INIT_OK**

enumerator **ESP_BLUFI_INIT_FAILED**

enum **esp_blufi_deinit_state_t**

BLUFI deinit status.

Values:

enumerator **ESP_BLUFI_DEINIT_OK**

enumerator **ESP_BLUFI_DEINIT_FAILED**

enum **esp_blufi_error_state_t**

Values:

enumerator **ESP_BLUFI_SEQUENCE_ERROR**

enumerator **ESP_BLUFI_CHECKSUM_ERROR**

enumerator **ESP_BLUFI_DECRYPT_ERROR**

enumerator **ESP_BLUFI_ENCRYPT_ERROR**

enumerator **ESP_BLUFI_INIT_SECURITY_ERROR**

enumerator **ESP_BLUFI_DH_MALLOC_ERROR**

enumerator **ESP_BLUFI_DH_PARAM_ERROR**

enumerator **ESP_BLUFI_READ_PARAM_ERROR**

enumerator **ESP_BLUFI_MAKE_PUBLIC_ERROR**

enumerator **ESP_BLUFI_DATA_FORMAT_ERROR**

enumerator **ESP_BLUFI_CALC_MD5_ERROR**

enumerator **ESP_BLUFI_WIFI_SCAN_FAIL**

enumerator **ESP_BLUFI_MSG_STATE_ERROR**

2.3.3 Controller && VHCI

Application Example

Check bluetooth/hci folder in ESP-IDF examples, which contains the following application:

- This is a BLE advertising demo with virtual HCI interface. Send Reset/ADV_PARAM/ADV_DATA/ADV_ENABLE HCI command for BLE advertising - bluetooth/hci/controller_vhci_ble_adv.

API Reference

Header File

- components/bt/include/esp32/include/esp_bt.h

Functions

esp_err_t **esp_ble_tx_power_set**(*esp_ble_power_type_t* power_type, *esp_power_level_t* power_level)

Set BLE TX power Connection Tx power should only be set after connection created.

参数

- **power_type** –: The type of which tx power, could set Advertising/Connection/Default and etc
- **power_level** –Power level(index) corresponding to absolute value(dbm)

返回 ESP_OK - success, other - failed

esp_power_level_t **esp_ble_tx_power_get**(*esp_ble_power_type_t* power_type)

Get BLE TX power Connection Tx power should only be get after connection created.

参数 **power_type** –: The type of which tx power, could set Advertising/Connection/Default and etc

返回 >= 0 - Power level, < 0 - Invalid

esp_err_t **esp_bredr_tx_power_set** (*esp_power_level_t* min_power_level, *esp_power_level_t* max_power_level)

Set BR/EDR TX power BR/EDR power control will use the power in range of minimum value and maximum value. The power level will effect the global BR/EDR TX power, such inquire, page, connection and so on. Please call the function after esp_bt_controller_enable and before any function which cause RF do TX. So you can call the function before doing discovery, profile init and so on. For example, if you want BR/EDR use the new TX power to do inquire, you should call this function before inquire. Another word, If call this function when BR/EDR is in inquire(ING), please do inquire again after call this function. Default minimum power level is ESP_PWR_LVL_N0, and maximum power level is ESP_PWR_LVL_P3.

参数

- **min_power_level** –The minimum power level
- **max_power_level** –The maximum power level

返回 ESP_OK - success, other - failed

esp_err_t **esp_bredr_tx_power_get** (*esp_power_level_t* *min_power_level, *esp_power_level_t* *max_power_level)

Get BR/EDR TX power If the argument is not NULL, then store the corresponding value.

参数

- **min_power_level** –The minimum power level
- **max_power_level** –The maximum power level

返回 ESP_OK - success, other - failed

esp_err_t **esp_bredr_sco_datapath_set** (*esp_sco_data_path_t* data_path)

Set default SCO data path Should be called after controller is enabled, and before (e)SCO link is established.

参数 **data_path** –SCO data path

返回 ESP_OK - success, other - failed

esp_err_t **esp_bt_controller_init** (*esp_bt_controller_config_t* *cfg)

Initialize BT controller to allocate task and other resource. This function should be called only once, before any other BT functions are called.

参数 **cfg** –Initial configuration of BT controller. Different from previous version, there's a mode and some connection configuration in "cfg" to configure controller work mode and allocate the resource which is needed.

返回 ESP_OK - success, other - failed

esp_err_t **esp_bt_controller_deinit** (void)

De-initialize BT controller to free resource and delete task. You should stop advertising and scanning, as well as disconnect all existing connections before de-initializing BT controller.

This function should be called only once, after any other BT functions are called.

返回 ESP_OK - success, other - failed

esp_err_t **esp_bt_controller_enable** (*esp_bt_mode_t* mode)

Enable BT controller. Due to a known issue, you cannot call esp_bt_controller_enable() a second time to change the controller mode dynamically. To change controller mode, call esp_bt_controller_disable() and then call esp_bt_controller_enable() with the new mode.

参数 **mode** –: the mode(BLE/BT/BTDM) to enable. For compatible of API, retain this argument. This mode must be equal as the mode in "cfg" of esp_bt_controller_init().

返回 ESP_OK - success, other - failed

esp_err_t **esp_bt_controller_disable** (void)

Disable BT controller.

返回 ESP_OK - success, other - failed

esp_bt_controller_status_t **esp_bt_controller_get_status** (void)

Get BT controller is initialised/de-initialised/enabled/disabled.

返回 status value

bool **esp_vhci_host_check_send_available**(void)

esp_vhci_host_check_send_available used for check actively if the host can send packet to controller or not.

返回 true for ready to send, false means cannot send packet

void **esp_vhci_host_send_packet**(uint8_t *data, uint16_t len)

esp_vhci_host_send_packet host send packet to controller

Should not call this function from within a critical section or when the scheduler is suspended.

参数

- **data** –the packet point
- **len** –the packet length

esp_err_t **esp_vhci_host_register_callback**(const *esp_vhci_host_callback_t* *callback)

esp_vhci_host_register_callback register the vhci reference callback struct defined by vhci_host_callback structure.

参数 **callback** –*esp_vhci_host_callback* type variable
返回 ESP_OK - success, ESP_FAIL - failed

esp_err_t **esp_bt_controller_mem_release**(*esp_bt_mode_t* mode)

esp_bt_controller_mem_release release the controller memory as per the mode

This function releases the BSS, data and other sections of the controller to heap. The total size is about 70k bytes.

esp_bt_controller_mem_release(mode) should be called only before esp_bt_controller_init() or after esp_bt_controller_deinit().

Note that once BT controller memory is released, the process cannot be reversed. It means you cannot use the bluetooth mode which you have released by this function.

If your firmware will later upgrade the Bluetooth controller mode (BLE -> BT Classic or disabled -> enabled) then do not call this function.

If the app calls esp_bt_controller_enable(ESP_BT_MODE_BLE) to use BLE only then it is safe to call esp_bt_controller_mem_release(ESP_BT_MODE_CLASSIC_BT) at initialization time to free unused BT Classic memory.

If the mode is ESP_BT_MODE_BTDM, then it may be useful to call API esp_bt_mem_release(ESP_BT_MODE_BTDM) instead, which internally calls esp_bt_controller_mem_release(ESP_BT_MODE_BTDM) and additionally releases the BSS and data consumed by the BT/BLE host stack to heap. For more details about usage please refer to the documentation of esp_bt_mem_release() function

参数 **mode** –: the mode want to release memory
返回 ESP_OK - success, other - failed

esp_err_t **esp_bt_mem_release**(*esp_bt_mode_t* mode)

esp_bt_mem_release release controller memory and BSS and data section of the BT/BLE host stack as per the mode

This function first releases controller memory by internally calling esp_bt_controller_mem_release(). Additionally, if the mode is set to ESP_BT_MODE_BTDM, it also releases the BSS and data consumed by the BT/BLE host stack to heap

Note that once BT memory is released, the process cannot be reversed. It means you cannot use the bluetooth mode which you have released by this function.

If your firmware will later upgrade the Bluetooth controller mode (BLE -> BT Classic or disabled -> enabled) then do not call this function.

If you never intend to use bluetooth in a current boot-up cycle, you can call esp_bt_mem_release(ESP_BT_MODE_BTDM) before esp_bt_controller_init or after esp_bt_controller_deinit.

For example, if a user only uses bluetooth for setting the WiFi configuration, and does not use bluetooth in the rest of the product operation" . In such cases, after receiving the WiFi configuration, you can disable/deinit bluetooth and release its memory. Below is the sequence of APIs to be called for such scenarios:

```
esp_bluedroid_disable();
esp_bluedroid_deinit();
esp_bt_controller_disable();
esp_bt_controller_deinit();
esp_bt_mem_release(ESP_BT_MODE_BTDM);
```

备注: In case of NimBLE host, to release BSS and data memory to heap, the mode needs to be set to ESP_BT_MODE_BTDM as controller is dual mode.

参数 **mode** –: the mode whose memory is to be released
返回 ESP_OK - success, other - failed

esp_err_t **esp_bt_sleep_enable**(void)

enable bluetooth to enter modem sleep

Note that this function shall not be invoked before esp_bt_controller_enable()

There are currently two options for bluetooth modem sleep, one is ORIG mode, and another is EVED Mode. EVED Mode is intended for BLE only.

For ORIG mode: Bluetooth modem sleep is enabled in controller start up by default if CONFIG_CTRL_BTDM_MODEM_SLEEP is set and "ORIG mode" is selected. In ORIG modem sleep mode, bluetooth controller will switch off some components and pause to work every now and then, if there is no event to process; and wakeup according to the scheduled interval and resume the work. It can also wakeup earlier upon external request using function "esp_bt_controller_wakeup_request" .

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_bt_sleep_disable**(void)

disable bluetooth modem sleep

Note that this function shall not be invoked before esp_bt_controller_enable()

If esp_bt_sleep_disable() is called, bluetooth controller will not be allowed to enter modem sleep;

If ORIG modem sleep mode is in use, if this function is called, bluetooth controller may not immediately wake up if it is dormant then. In this case, esp_bt_controller_wakeup_request() can be used to shorten the time for wakeup.

返回

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_scan_dupilcate_list_flush**(void)

Manually clear scan duplicate list.

Note that scan duplicate list will be automatically cleared when the maximum amount of device in the filter is reached the amount of device in the filter can be configured in menuconfig.

备注: This function name is incorrectly spelled, it will be fixed in release 5.x version.

返回

- ESP_OK : success
- other : failed

void **esp_wifi_bt_power_domain_on**(void)

bt Wi-Fi power domain power on

void **esp_wifi_bt_power_domain_off**(void)

bt Wi-Fi power domain power off

Structures

struct **esp_bt_controller_config_t**

Controller config options, depend on config mask. Config mask indicate which functions enabled, this means some options or parameters of some functions enabled by config mask.

Public Members

uint16_t **controller_task_stack_size**

Bluetooth controller task stack size

uint8_t **controller_task_prio**

Bluetooth controller task priority

uint8_t **hci_uart_no**

If use UART1/2 as HCI IO interface, indicate UART number

uint32_t **hci_uart_baudrate**

If use UART1/2 as HCI IO interface, indicate UART baudrate

uint8_t **scan_duplicate_mode**

scan duplicate mode

uint8_t **scan_duplicate_type**

scan duplicate type

uint16_t **normal_adv_size**

Normal adv size for scan duplicate

uint16_t **mesh_adv_size**

Mesh adv size for scan duplicate

uint16_t **send_adv_reserved_size**

Controller minimum memory value

uint32_t **controller_debug_flag**

Controller debug log flag

uint8_t **mode**

Controller mode: BR/EDR, BLE or Dual Mode

uint8_t **ble_max_conn**

BLE maximum connection numbers

uint8_t **bt_max_acl_conn**

BR/EDR maximum ACL connection numbers

uint8_t **bt_sco_datapath**

SCO data path, i.e. HCI or PCM module

bool **auto_latency**

BLE auto latency, used to enhance classic BT performance

bool **bt_legacy_auth_vs_evt**

BR/EDR Legacy auth complete event required to protect from BIAS attack

uint8_t **bt_max_sync_conn**

BR/EDR maximum ACL connection numbers. Effective in menuconfig

uint8_t **ble_sca**

BLE low power crystal accuracy index

uint8_t **pcm_role**

PCM role (master & slave)

uint8_t **pcm_polar**

PCM polar trig (falling clk edge & rising clk edge)

bool **hli**

Using high level interrupt or not

uint16_t **dup_list_refresh_period**

Duplicate scan list refresh period

uint32_t **magic**

Magic number

struct **esp_vhci_host_callback**

esp_vhci_host_callback used for vhci call host function to notify what host need to do

Public Members

void (*__notify_host_send_available__)(void)

callback used to notify that the host can send packet to controller

int (*__notify_host_recv__)(uint8_t *data, uint16_t len)

callback used to notify that the controller has a packet to send to the host

Macros

ESP_BT_CONTROLLER_CONFIG_MAGIC_VAL

BT_CONTROLLER_INIT_CONFIG_DEFAULT()

Type Definitions

typedef struct *esp_vhci_host_callback* **esp_vhci_host_callback_t**

esp_vhci_host_callback used for vhci call host function to notify what host need to do

Enumerations

enum **esp_bt_mode_t**

Bluetooth mode for controller enable/disable.

Values:

enumerator **ESP_BT_MODE_IDLE**

Bluetooth is not running

enumerator **ESP_BT_MODE_BLE**

Run BLE mode

enumerator **ESP_BT_MODE_CLASSIC_BT**

Run Classic BT mode

enumerator **ESP_BT_MODE_BTDM**

Run dual mode

enum **[anonymous]**

BLE sleep clock accuracy(SCA), values for ble_sca field in *esp_bt_controller_config_t*, currently only ESP_BLE_SCA_500PPM and ESP_BLE_SCA_250PPM are supported.

Values:

enumerator **ESP_BLE_SCA_500PPM**

BLE SCA at 500ppm

enumerator **ESP_BLE_SCA_250PPM**

BLE SCA at 250ppm

enumerator **ESP_BLE_SCA_150PPM**

BLE SCA at 150ppm

enumerator **ESP_BLE_SCA_100PPM**

BLE SCA at 100ppm

enumerator **ESP_BLE_SCA_75PPM**

BLE SCA at 75ppm

enumerator **ESP_BLE_SCA_50PPM**

BLE SCA at 50ppm

enumerator **ESP_BLE_SCA_30PPM**

BLE SCA at 30ppm

enumerator **ESP_BLE_SCA_20PPM**

BLE SCA at 20ppm

enum **esp_bt_controller_status_t**

Bluetooth controller enable/disable/initialised/de-initialised status.

Values:

enumerator **ESP_BT_CONTROLLER_STATUS_IDLE**

enumerator **ESP_BT_CONTROLLER_STATUS_INITED**

enumerator **ESP_BT_CONTROLLER_STATUS_ENABLED**

enumerator **ESP_BT_CONTROLLER_STATUS_NUM**

enum **esp_ble_power_type_t**

BLE tx power type ESP_BLE_PWR_TYPE_CONN_HDL0-8: for each connection, and only be set after connection completed. when disconnect, the correspond TX power is not effected. ESP_BLE_PWR_TYPE_ADV : for advertising/scan response. ESP_BLE_PWR_TYPE_SCAN : for scan. ESP_BLE_PWR_TYPE_DEFAULT : if each connection ' s TX power is not set, it will use this default value. if neither in scan mode nor in adv mode, it will use this default value. If none of power type is set, system will use ESP_PWR_LVL_P3 as default for ADV/SCAN/CONN0-9.

Values:

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL0**

For connection handle 0

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL1**

For connection handle 1

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL2**

For connection handle 2

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL3**

For connection handle 3

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL4**

For connection handle 4

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL5**

For connection handle 5

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL6**

For connection handle 6

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL7**

For connection handle 7

enumerator **ESP_BLE_PWR_TYPE_CONN_HDL8**

For connection handle 8

enumerator **ESP_BLE_PWR_TYPE_ADV**

For advertising

enumerator **ESP_BLE_PWR_TYPE_SCAN**

For scan

enumerator **ESP_BLE_PWR_TYPE_DEFAULT**

For default, if not set other, it will use default value

enumerator **ESP_BLE_PWR_TYPE_NUM**

TYPE numbers

enum **esp_power_level_t**

Bluetooth TX power level(index), it’ s just a index corresponding to power(dbm).

Values:

enumerator **ESP_PWR_LVL_N12**

Corresponding to -12dbm

enumerator **ESP_PWR_LVL_N9**

Corresponding to -9dbm

enumerator **ESP_PWR_LVL_N6**

Corresponding to -6dbm

enumerator **ESP_PWR_LVL_N3**

Corresponding to -3dbm

enumerator **ESP_PWR_LVL_N0**

Corresponding to 0dbm

enumerator **ESP_PWR_LVL_P3**

Corresponding to +3dbm

enumerator **ESP_PWR_LVL_P6**

Corresponding to +6dbm

enumerator **ESP_PWR_LVL_P9**

Corresponding to +9dbm

enumerator **ESP_PWR_LVL_N14**

Backward compatibility! Setting to -14dbm will actually result to -12dbm

enumerator **ESP_PWR_LVL_N11**

Backward compatibility! Setting to -11dbm will actually result to -9dbm

enumerator **ESP_PWR_LVL_N8**

Backward compatibility! Setting to -8dbm will actually result to -6dbm

enumerator **ESP_PWR_LVL_N5**

Backward compatibility! Setting to -5dbm will actually result to -3dbm

enumerator **ESP_PWR_LVL_N2**

Backward compatibility! Setting to -2dbm will actually result to 0dbm

enumerator **ESP_PWR_LVL_P1**

Backward compatibility! Setting to +1dbm will actually result to +3dbm

enumerator **ESP_PWR_LVL_P4**

Backward compatibility! Setting to +4dbm will actually result to +6dbm

enumerator **ESP_PWR_LVL_P7**

Backward compatibility! Setting to +7dbm will actually result to +9dbm

enum **esp_sco_data_path_t**

Bluetooth audio data transport path.

Values:

enumerator **ESP_SCO_DATA_PATH_HCI**

data over HCI transport

enumerator **ESP_SCO_DATA_PATH_PCM**

data over PCM interface

2.3.4 NimBLE-based host APIs

Overview

Apache MyNewt NimBLE is a highly configurable and BT SIG qualifiable BLE stack providing both host and controller functionalities. ESP-IDF supports NimBLE host stack which is specifically ported for ESP32 platform and FreeRTOS. The underlying controller is still the same (as in case of Bluedroid) providing VHCI interface. Refer to NimBLE user guide for a complete list of features and additional information on NimBLE stack. Most features of NimBLE including BLE Mesh are supported by ESP-IDF. The porting layer is kept cleaner by maintaining all the existing APIs of NimBLE along with a single ESP-NimBLE API for initialization, making it simpler for the application developers.

Architecture

Currently, NimBLE host and controller support different transports such as UART and RAM between them. However, RAM transport cannot be used as is in case of ESP as ESP controller supports VHCI interface and buffering schemes used by NimBLE host is incompatible with that used by ESP controller. Therefore, a new transport between NimBLE host and ESP controller has been added. This is depicted in the figure below. This layer is responsible for maintaining pool of transport buffers and formatting buffers exchanges between host and controller as per the requirements.

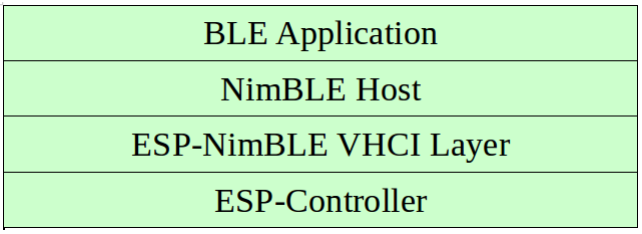


图 1: ESP NimBLE Stack

Threading Model

The NimBLE host can run inside the application thread or can have its own independent thread. This flexibility is inherently provided by NimBLE design. By default, a thread is spawned by the porting function `nimble_port_freertos_init`. This behavior can be changed by overriding the same function. For BLE Mesh, additional thread (advertising thread) is used which keeps on feeding advertisement events to the main thread.

Programming Sequence

To begin with, make sure that the NimBLE stack is enabled from menuconfig *choose NimBLE for the Bluetooth host*.

Typical programming sequence with NimBLE stack consists of the following steps:

- Initialize NVS flash using *`nvs_flash_init()`* API. This is because ESP controller uses NVS during initialization.
- Initialize the host and controller stack using `nimble_port_init`.
- Initialize the required NimBLE host configuration parameters and callbacks
- Perform application specific tasks/initialization
- Run the thread for host stack using `nimble_port_freertos_init`

This documentation does not cover NimBLE APIs. Refer to NimBLE tutorial for more details on the programming sequence/NimBLE APIs for different scenarios.

API Reference

Header File

- components/bt/host/nimble/esp-hci/include/esp_nimble_hci.h

Functions

esp_err_t **esp_nimble_hci_init**(void)

Initialize VHCI transport layer between NimBLE Host and ESP Bluetooth controller.

This function initializes the transport buffers to be exchanged between NimBLE host and ESP controller. It also registers required host callbacks with the controller.

返回

- ESP_OK if the initialization is successful
- Appropriate error code from esp_err_t in case of an error

esp_err_t **esp_nimble_hci_deinit**(void)

Deinitialize VHCI transport layer between NimBLE Host and ESP Bluetooth controller.

备注: This function should be called after the NimBLE host is deinitialized.

返回

- ESP_OK if the deinitialization is successful
- Appropriate error codes from esp_err_t in case of an error

Macros

BLE_HCI_UART_H4_NONE

BLE_HCI_UART_H4_CMD

BLE_HCI_UART_H4_ACL

BLE_HCI_UART_H4_SCO

BLE_HCI_UART_H4_EVT

ESP-IDF 目前支持两个主机堆栈。基于 Bluedroid 的堆栈（默认）支持传统蓝牙和 BLE，而基于 Apache NimBLE 的堆栈仅支持 BLE。用户可参考如下信息进行选择：

- 对于同时涉及传统蓝牙和 BLE 的用例，应该选用 Bluedroid。
- 对于仅涉及 BLE 的用例，建议选用 NimBLE。在代码占用和运行时，NimBLE 对内存的要求较低，因此适用于此类场景。

蓝牙 API 的示例代码存放于 ESP-IDF 示例项目的 bluetooth/bluedroid 目录下。

下面的示例给出了详细介绍：

- GATT 客户端示例
- GATT 服务端服务表格示例
- GATT 服务端示例
- GATT 客户端安全性示例
- GATT 服务端安全性示例
- GATT 客户端多连接示例

2.4 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)**: There are items remained to retrieve

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_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_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_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 occured 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

2.5 连网 API

2.5.1 Wi-Fi

ESP-NOW

概述 ESP-NOW 是一种由乐鑫公司定义的无连接 Wi-Fi 通信协议。在 ESP-NOW 中，应用程序数据被封装在各个供应商的动作帧中，然后在无连接的情况下，从一个 Wi-Fi 设备传输到另一个 Wi-Fi 设备。CTR 与 CBC-MAC 协议 (CCMP) 可用来保护动作帧的安全。ESP-NOW 广泛应用于智能照明、远程控制、传感器等领域。

帧格式 ESP-NOW 使用各个供应商的动作帧传输数据，默认比特率为 1 Mbps。各个供应商的动作帧格式为：

MAC 报头	分类代码	组织标识符	随机值	供应商特定内容	FCS
24 字节	1 字节	3 字节	4 字节	7~257 字节	4 字节

- 分类代码：分类代码字段可用于指示各个供应商的类别（比如 127）。
- 组织标识符：组织标识符包含一个唯一标识符 (比如 0x18fe34)，为乐鑫指定的 MAC 地址的前三个字节。
- 随机值：防止重放攻击。
- 供应商特定内容：供应商特定内容包含供应商特定字段，如下所示：

元素 ID	长度	组织标识符	类型	版本	正文
1 字节	1 字节	3 字节	1 字节	1 字节	0 ~ 250 字节

- 元素 ID：元素 ID 字段可用于指示特定于供应商的元素。
- 长度：长度是组织标识符、类型、版本和正文的总长度。
- 组织标识符：组织标识符包含一个唯一标识符 (比如 0x18fe34)，为乐鑫指定的 MAC 地址的前三个字节。
- 类型：类型字段设置为 4，代表 ESP-NOW。
- 版本：版本字段设置为 ESP-NOW 的版本。
- 正文：正文包含 ESP-NOW 数据。

由于 ESP-NOW 是无连接的，因此 MAC 报头与标准帧略有不同。FrameControl 字段的 FromDS 和 ToDS 位均为 0。第一个地址字段用于配置目标地址。第二个地址字段用于配置源地址。第三个地址字段用于配置广播地址 (0xff:0xff:0xff:0xff:0xff:0xff)。

安全

ESP-NOW 采用 CCMP 方法保护供应商特定动作帧的安全，具体可参考 IEEE Std. 802.11-2012。Wi-Fi 设备维护一个初始

- PMK 可使用 AES-128 算法加密 LMK。请调用 *esp_now_set_pmk()* 设置 PMK。如果未设置 PMK，将使用默认 PMK。
- LMK 可通过 CCMP 方法对供应商特定的动作帧进行加密，最多拥有 6 个不同的 LMK。如果未设置配对设备的 LMK，则动作帧不进行加密。

目前，不支持加密组播供应商特定的动作帧。

初始化和反初始化 调用 *esp_now_init()* 初始化 ESP-NOW，调用 *esp_now_deinit()* 反初始化 ESP-NOW。ESP-NOW 数据必须在 Wi-Fi 启动后传输，因此建议在初始化 ESP-NOW 之前启动 Wi-Fi，并在反初始化 ESP-NOW 之后停止 Wi-Fi。当调用 *esp_now_deinit()* 时，配对设备的所有信息都将被删除。

添加配对设备 在将数据发送到其他设备之前，请先调用*esp_now_add_peer()* 将其添加到配对设备列表中。如果启用了加密，则必须设置 LMK。ESP-NOW 数据可以从 Station 或 Softap 接口发送。确保在发送 ESP-NOW 数据之前已启用该接口。

配对设备的最大数量是 20，其中加密设备的数量不超过 4，默认值是 2。如果想要修改加密设备的数量，在 Wi-Fi menuconfig 设置*CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM*。

在发送广播数据之前必须添加具有广播 MAC 地址的设备。配对设备的信道范围是从 0 ~ 14。如果信道设置为 0，数据将在当前信道上发送。否则，必须使用本地设备所在的通道。

发送 ESP-NOW 数据 调用*esp_now_send()* 发送 ESP-NOW 数据，调用*esp_now_register_send_cb()* 注册发送回调函数。如果 MAC 层成功接收到数据，则该函数将返回 *ESP_NOW_SEND_SUCCESS* 事件。否则，它将返回 *ESP_NOW_SEND_FAIL*。ESP-NOW 数据发送失败可能有几种原因，比如目标设备不存在、设备的信道不相同、动作帧在传输过程中丢失等。应用层并不一定可以总能接收到数据。如果需要，应用层可在接收 ESP-NOW 数据时发回一个应答 (ACK) 数据。如果接收 ACK 数据超时，则将重新传输 ESP-NOW 数据。可以为 ESP-NOW 数据设置序列号，从而删除重复的数据。

如果有大量 ESP-NOW 数据要发送，调用 `esp_now_send()` 时需注意单次发送的数据不能超过 250 字节。请注意，两个 ESP-NOW 数据包的发送间隔太短可能导致回调函数返回混乱。因此，建议在等到上一次回调函数返回 ACK 后再发送下一个 ESP-NOW 数据。发送回调函数从高优先级的 Wi-Fi 任务中运行。因此，不要在回调函数中执行冗长的操作。相反，将必要的数据发布到队列，并交给优先级较低的任务处理。

接收 ESP-NOW 数据 调用*esp_now_register_recv_cb()* 注册接收回调函数。当接收 ESP-NOW 数据时，需要调用接收回调函数。接收回调函数也在 Wi-Fi 任务任务中运行。因此，不要在回调函数中执行冗长的操作。相反，将必要的数据发布到队列，并交给优先级较低的任务处理。

配置 ESP-NOW 速率 调用*esp_wifi_config_espnow_rate()* 配置指定接口的 ESPNOW 速率。确保在配置速率之前使能接口。这个 API 应该在*esp_wifi_start()* 之后调用。

配置 ESP-NOW 功耗参数 当且仅当 ESP32-C2 配置为 STA 模式时，允许其进行休眠。

进行休眠时，调用*esp_now_set_wake_window()* 为 ESP-NOW 收包配置 Window。默认情况下 Window 为最大值，将允许一直收包。

如果对 ESP-NOW 进功耗管理，也需要调用*esp_wifi_connectionless_module_set_wake_interval()*。

请参考非连接模块功耗管理 获取更多信息。

应用示例

- 如何在设备间传输 ESP-NOW 数据：wifi/espnow。
- 了解更多 ESP-NOW 的应用示例，请参考 README.md 文件。

API 参考

Header File

- components/esp_wifi/include/esp_now.h

Functions

esp_err_t **esp_now_init**(void)

Initialize ESPNOW function.

返回

- ESP_OK : succeed

- ESP_ERR_ESPNOW_INTERNAL : Internal error

esp_err_t **esp_now_deinit**(void)

De-initialize ESPNOW function.

返回

- ESP_OK : succeed

esp_err_t **esp_now_get_version**(uint32_t *version)

Get the version of ESPNOW.

参数 **version** –ESPNOW version

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_ARG : invalid argument

esp_err_t **esp_now_register_recv_cb**(*esp_now_recv_cb_t* cb)

Register callback function of receiving ESPNOW data.

参数 **cb** –callback function of receiving ESPNOW data

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_INTERNAL : internal error

esp_err_t **esp_now_unregister_recv_cb**(void)

Unregister callback function of receiving ESPNOW data.

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

esp_err_t **esp_now_register_send_cb**(*esp_now_send_cb_t* cb)

Register callback function of sending ESPNOW data.

参数 **cb** –callback function of sending ESPNOW data

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_INTERNAL : internal error

esp_err_t **esp_now_unregister_send_cb**(void)

Unregister callback function of sending ESPNOW data.

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

esp_err_t **esp_now_send**(const uint8_t *peer_addr, const uint8_t *data, size_t len)

Send ESPNOW data.

Attention 1. If peer_addr is not NULL, send data to the peer whose MAC address matches peer_addr
Attention 2. If peer_addr is NULL, send data to all of the peers that are added to the peer list
Attention 3. The maximum length of data must be less than ESP_NOW_MAX_DATA_LEN
Attention 4. The buffer pointed to by data argument does not need to be valid after esp_now_send returns

参数

- **peer_addr** –peer MAC address
- **data** –data to send
- **len** –length of data

返回

- ESP_OK : succeed

- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_INTERNAL : internal error
- ESP_ERR_ESPNOW_NO_MEM : out of memory, when this happens, you can delay a while before sending the next data
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found
- ESP_ERR_ESPNOW_IF : current WiFi interface doesn't match that of peer

esp_err_t **esp_now_add_peer**(const *esp_now_peer_info_t* *peer)

Add a peer to peer list.

参数 **peer** –peer information

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_FULL : peer list is full
- ESP_ERR_ESPNOW_NO_MEM : out of memory
- ESP_ERR_ESPNOW_EXIST : peer has existed

esp_err_t **esp_now_del_peer**(const uint8_t *peer_addr)

Delete a peer from peer list.

参数 **peer_addr** –peer MAC address

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

esp_err_t **esp_now_mod_peer**(const *esp_now_peer_info_t* *peer)

Modify a peer.

参数 **peer** –peer information

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_FULL : peer list is full

esp_err_t **esp_wifi_config_espnow_rate**(*wifi_interface_t* ifx, *wifi_phy_rate_t* rate)

Config ESPNOW rate of specified interface.

Deprecated:

please use *esp_now_set_peer_rate_config()* instead.

Attention 1. This API should be called after esp_wifi_start().

Attention 2. This API only work when not use Wi-Fi 6 and *esp_now_set_peer_rate_config()* not called.

参数

- **ifx** –Interface to be configured.
- **rate** –Phy rate to be configured.

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_now_set_peer_rate_config**(const uint8_t *peer_addr, *esp_now_rate_config_t* *config)

Set ESPNOW rate config for each peer.

Attention 1. This API should be called after esp_wifi_start() and *esp_now_init()*.

参数

- **peer_addr** –peer MAC address
- **config** –rate config to be configured.

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_INTERNAL : internal error

esp_err_t **esp_now_get_peer**(const uint8_t *peer_addr, *esp_now_peer_info_t* *peer)

Get a peer whose MAC address matches peer_addr from peer list.

参数

- **peer_addr** –peer MAC address
- **peer** –peer information

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

esp_err_t **esp_now_fetch_peer**(bool from_head, *esp_now_peer_info_t* *peer)

Fetch a peer from peer list. Only return the peer which address is unicast, for the multicast/broadcast address, the function will ignore and try to find the next in the peer list.

参数

- **from_head** –fetch from head of list or not
- **peer** –peer information

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

bool **esp_now_is_peer_exist**(const uint8_t *peer_addr)

Peer exists or not.

参数 **peer_addr** –peer MAC address

返回

- true : peer exists
- false : peer not exists

esp_err_t **esp_now_get_peer_num**(*esp_now_peer_num_t* *num)

Get the number of peers.

参数 **num** –number of peers

返回

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument

esp_err_t **esp_now_set_pmk**(const uint8_t *pmk)

Set the primary master key.

Attention 1. primary master key is used to encrypt local master key

参数 **pmk** –primary master key
返回
- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument

esp_err_t **esp_now_set_wake_window**(uint16_t window)

Set wake window for esp_now to wake up in interval unit.

Attention 1. This configuration could work at connected status. When ESP_WIFI_STA_DISCONNECTED_PM_ENABLE is enabled, this configuration could work at disconnected status.
Attention 2. Default value is the maximum.

参数 **window** –Milliseconds would the chip keep waked each interval, from 0 to 65535.
返回
- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

Structures

struct **esp_now_peer_info**

ESPNOW peer information parameters.

Public Members

uint8_t **peer_addr**[ESP_NOW_ETH_ALEN]

ESPNOW peer MAC address that is also the MAC address of station or softap

uint8_t **lmk**[ESP_NOW_KEY_LEN]

ESPNOW peer local master key that is used to encrypt data

uint8_t **channel**

Wi-Fi channel that peer uses to send/receive ESPNOW data. If the value is 0, use the current channel which station or softap is on. Otherwise, it must be set as the channel that station or softap is on.

wifi_interface_t **ifidx**

Wi-Fi interface that peer uses to send/receive ESPNOW data

bool **encrypt**

ESPNOW data that this peer sends/receives is encrypted or not

void ***priv**

ESPNOW peer private data

struct **esp_now_peer_num**

Number of ESPNOW peers which exist currently.

Public Members

int **total_num**

Total number of ESPNOW peers, maximum value is ESP_NOW_MAX_TOTAL_PEER_NUM

int **encrypt_num**

Number of encrypted ESPNOW peers, maximum value is ESP_NOW_MAX_ENCRYPT_PEER_NUM

struct **esp_now_recv_info**

ESPNOW packet information.

Public Members

uint8_t ***src_addr**

Source address of ESPNOW packet

uint8_t ***des_addr**

Destination address of ESPNOW packet

wifi_pkt_rx_ctrl_t ***rx_ctrl**

Rx control info of ESPNOW packet

struct **esp_now_rate_config**

ESPNOW rate config.

Public Members

wifi_phy_mode_t **phymode**

ESPNOW phymode of specified interface

wifi_phy_rate_t **rate**

ESPNOW rate of specified interface

bool **ersu**

ESPNOW using ersu send frame

Macros

ESP_ERR_ESPNOW_BASE

ESPNOW error number base.

ESP_ERR_ESPNOW_NOT_INIT

ESPNOW is not initialized.

ESP_ERR_ESPNOW_ARG

Invalid argument

ESP_ERR_ESPNOW_NO_MEM

Out of memory

ESP_ERR_ESPNOW_FULL

ESPNOW peer list is full

ESP_ERR_ESPNOW_NOT_FOUND

ESPNOW peer is not found

ESP_ERR_ESPNOW_INTERNAL

Internal error

ESP_ERR_ESPNOW_EXIST

ESPNOW peer has existed

ESP_ERR_ESPNOW_IF

Interface error

ESP_NOW_ETH_ALEN

Length of ESPNOW peer MAC address

ESP_NOW_KEY_LEN

Length of ESPNOW peer local master key

ESP_NOW_MAX_TOTAL_PEER_NUM

Maximum number of ESPNOW total peers

ESP_NOW_MAX_ENCRYPT_PEER_NUM

Maximum number of ESPNOW encrypted peers

ESP_NOW_MAX_DATA_LEN

Maximum length of ESPNOW data which is sent very time

Type Definitions

typedef struct *esp_now_peer_info* **esp_now_peer_info_t**

ESPNOW peer information parameters.

typedef struct *esp_now_peer_num* **esp_now_peer_num_t**

Number of ESPNOW peers which exist currently.

typedef struct *esp_now_recv_info* **esp_now_recv_info_t**

ESPNOW packet information.

typedef struct *esp_now_rate_config* **esp_now_rate_config_t**

ESPNOW rate config.

typedef void (***esp_now_recv_cb_t**)(const *esp_now_recv_info_t* *esp_now_info, const uint8_t *data, int data_len)

Callback function of receiving ESPNOW data.

Attention esp_now_info is a local variable, it can only be used in the callback.

Param esp_now_info received ESPNOW packet information
Param data received data
Param data_len length of received data

typedef void (***esp_now_send_cb_t**)(const uint8_t *mac_addr, *esp_now_send_status_t* status)

Callback function of sending ESPNOW data.

Param mac_addr peer MAC address
Param status status of sending ESPNOW data (succeed or fail)

Enumerations

enum **esp_now_send_status_t**

Status of sending ESPNOW data .

Values:

enumerator **ESP_NOW_SEND_SUCCESS**

Send ESPNOW data successfully

enumerator **ESP_NOW_SEND_FAIL**

Send ESPNOW data fail

SmartConfig

The SmartConfig™ is a provisioning technology developed by TI to connect a new Wi-Fi device to a Wi-Fi network. It uses a mobile app to broadcast the network credentials from a smartphone, or a tablet, to an un-provisioned Wi-Fi device.

The advantage of this technology is that the device does not need to directly know SSID or password of an Access Point (AP). This information is provided using the smartphone. This is particularly important to headless device and systems, due to their lack of a user interface.

If you are looking for other options to provision your ESP32-C2 devices, check 配网 *API*.

Application Example Connect ESP32-C2 to target AP using SmartConfig: wifi/smart_config.

API Reference

Header File

- components/esp_wifi/include/esp_smartconfig.h

Functions

const char ***esp_smartconfig_get_version**(void)

Get the version of SmartConfig.

返回

- SmartConfig version const char.

esp_err_t **esp_smartconfig_start**(const *smartconfig_start_config_t* *config)

Start SmartConfig, config ESP device to connect AP. You need to broadcast information by phone APP. Device sniffer special packets from the air that containing SSID and password of target AP.

Attention 1. This API can be called in station or softAP-station mode.
Attention 2. Can not call esp_smartconfig_start twice before it finish, please call esp_smartconfig_stop first.

参数 **config** –pointer to smartconfig start configure structure
返回
- ESP_OK: succeed
- others: fail

esp_err_t **esp_smartconfig_stop**(void)

Stop SmartConfig, free the buffer taken by esp_smartconfig_start.

Attention Whether connect to AP succeed or not, this API should be called to free memory taken by smartconfig_start.

返回
- ESP_OK: succeed
- others: fail

esp_err_t **esp_esptouch_set_timeout**(uint8_t time_s)

Set timeout of SmartConfig process.

Attention Timing starts from SC_STATUS_FIND_CHANNEL status. SmartConfig will restart if timeout.

参数 **time_s** –range 15s~255s, offset:45s.
返回
- ESP_OK: succeed
- others: fail

esp_err_t **esp_smartconfig_set_type**(*smartconfig_type_t* type)

Set protocol type of SmartConfig.

Attention If users need to set the SmartConfig type, please set it before calling esp_smartconfig_start.

参数 **type** –Choose from the smartconfig_type_t.
返回
- ESP_OK: succeed
- others: fail

esp_err_t **esp_smartconfig_fast_mode**(bool enable)

Set mode of SmartConfig. default normal mode.

Attention 1. Please call it before API esp_smartconfig_start.
Attention 2. Fast mode have corresponding APP(phone).
Attention 3. Two mode is compatible.

参数 **enable** –false-disable(default); true-enable;
返回
- ESP_OK: succeed

- others: fail

esp_err_t **esp_smartconfig_get_rvd_data**(uint8_t *rvd_data, uint8_t len)

Get reserved data of ESPTouch v2.

参数

- **rvd_data** –reserved data
- **len** –length of reserved data

返回

- ESP_OK: succeed
- others: fail

Structures

struct **smartconfig_event_got_ssid_pswd_t**

Argument structure for SC_EVENT_GOT_SSID_PSWD event

Public Members

uint8_t **ssid**[32]

SSID of the AP. Null terminated string.

uint8_t **password**[64]

Password of the AP. Null terminated string.

bool **bssid_set**

whether set MAC address of target AP or not.

uint8_t **bssid**[6]

MAC address of target AP.

smartconfig_type_t **type**

Type of smartconfig(ESPTouch or AirKiss).

uint8_t **token**

Token from cellphone which is used to send ACK to cellphone.

uint8_t **cellphone_ip**[4]

IP address of cellphone.

struct **smartconfig_start_config_t**

Configure structure for esp_smartconfig_start

Public Members

bool **enable_log**

Enable smartconfig logs.

bool **esp_touch_v2_enable_crypt**

Enable ESPTouch v2 crypt.

char ***esp_touch_v2_key**

ESPTouch v2 crypt key, len should be 16.

Macros

SMARTCONFIG_START_CONFIG_DEFAULT()

Enumerations

enum **smartconfig_type_t**

Values:

enumerator **SC_TYPE_ESPTOUCH**

protocol: ESPTouch

enumerator **SC_TYPE_AIRKISS**

protocol: AirKiss

enumerator **SC_TYPE_ESPTOUCH_AIRKISS**

protocol: ESPTouch and AirKiss

enumerator **SC_TYPE_ESPTOUCH_V2**

protocol: ESPTouch v2

enum **smartconfig_event_t**

Smartconfig event declarations

Values:

enumerator **SC_EVENT_SCAN_DONE**

Station smartconfig has finished to scan for APs

enumerator **SC_EVENT_FOUND_CHANNEL**

Station smartconfig has found the channel of the target AP

enumerator **SC_EVENT_GOT_SSID_PSWD**

Station smartconfig got the SSID and password

enumerator **SC_EVENT_SEND_ACK_DONE**

Station smartconfig has sent ACK to cellphone

Wi-Fi 库

概述 Wi-Fi 库支持配置及监控 ESP32-C2 Wi-Fi 连网功能。支持配置：

- station 模式（即 STA 模式或 Wi-Fi 客户端模式），此时 ESP32-C2 连接到接入点 (AP)。
- AP 模式（即 Soft-AP 模式或接入点模式），此时基站连接到 ESP32-C2。
- station/AP 共存模式（ESP32-C2 既是接入点，同时又作为基站连接到另外一个接入点）。
- 上述模式的各种安全模式（WPA、WPA2 及 WEP 等）。
- 扫描接入点（包括主动扫描及被动扫描）。
- 使用混杂模式监控 IEEE802.11 Wi-Fi 数据包。

应用示例 ESP-IDF 仓库的 wifi 目录下提供了演示 Wi-Fi 库功能的几个应用示例，请查看 README 了解更多详细信息。

API 参考

Header File

- components/esp_wifi/include/esp_wifi.h

Functions

esp_err_t **esp_wifi_init** (const *wifi_init_config_t* *config)

Initialize WiFi Allocate resource for WiFi driver, such as WiFi control structure, RX/TX buffer, WiFi NVS structure etc. This WiFi also starts WiFi task.

Attention 1. This API must be called before all other WiFi API can be called

Attention 2. Always use WIFI_INIT_CONFIG_DEFAULT macro to initialize the configuration to default values, this can guarantee all the fields get correct value when more fields are added into *wifi_init_config_t* in future release. If you want to set your own initial values, overwrite the default values which are set by WIFI_INIT_CONFIG_DEFAULT. Please be notified that the field 'magic' of *wifi_init_config_t* should always be WIFI_INIT_CONFIG_MAGIC!

参数 **config** –pointer to WiFi initialized configuration structure; can point to a temporary variable.

返回

- ESP_OK: succeed
- ESP_ERR_NO_MEM: out of memory
- others: refer to error code esp_err.h

esp_err_t **esp_wifi_deinit** (void)

Deinit WiFi Free all resource allocated in esp_wifi_init and stop WiFi task.

Attention 1. This API should be called if you want to remove WiFi driver from the system

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_set_mode** (*wifi_mode_t* mode)

Set the WiFi operating mode.

```
Set the WiFi operating mode as station, soft-AP, station+soft-AP or NAN.
The default mode is station mode.
```

参数 **mode** –WiFi operating mode

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error code in esp_err.h

esp_err_t **esp_wifi_get_mode**(*wifi_mode_t* *mode)

Get current operating mode of WiFi.

参数 **mode** –**[out]** store current WiFi mode

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_start**(void)

Start WiFi according to current configuration If mode is WIFI_MODE_STA, it creates station control block and starts station If mode is WIFI_MODE_AP, it creates soft-AP control block and starts soft-AP If mode is WIFI_MODE_APSTA, it creates soft-AP and station control block and starts soft-AP and station If mode is WIFI_MODE_NAN, it creates NAN control block and starts NAN.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NO_MEM: out of memory
- ESP_ERR_WIFI_CONN: WiFi internal error, station or soft-AP control block wrong
- ESP_FAIL: other WiFi internal errors

esp_err_t **esp_wifi_stop**(void)

Stop WiFi If mode is WIFI_MODE_STA, it stops station and frees station control block If mode is WIFI_MODE_AP, it stops soft-AP and frees soft-AP control block If mode is WIFI_MODE_APSTA, it stops station/soft-AP and frees station/soft-AP control block If mode is WIFI_MODE_NAN, it stops NAN and frees NAN control block.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_restore**(void)

Restore WiFi stack persistent settings to default values.

This function will reset settings made using the following APIs:

- esp_wifi_set_bandwidth,
- esp_wifi_set_protocol,
- esp_wifi_set_config related
- esp_wifi_set_mode

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_connect**(void)

Connect WiFi station to the AP.

Attention 1. This API only impact WIFI_MODE_STA or WIFI_MODE_APSTA mode

Attention 2. If station interface is connected to an AP, call esp_wifi_disconnect to disconnect.

Attention 3. The scanning triggered by esp_wifi_scan_start() will not be effective until connection between device and the AP is established. If device is scanning and connecting at the same time, it will abort scanning and return a warning message and error number ESP_ERR_WIFI_STATE. If you want to do reconnection after device received disconnect event, remember to add the maximum retry time, otherwise the called scan will not work. This is especially true when the AP doesn't exist, and you still try reconnection after device received disconnect event with the reason code WIFI_REASON_NO_AP_FOUND.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_CONN: WiFi internal error, station or soft-AP control block wrong
- ESP_ERR_WIFI_SSID: SSID of AP which station connects is invalid

esp_err_t **esp_wifi_disconnect**(void)

Disconnect WiFi station from the AP.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi was not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
- ESP_FAIL: other WiFi internal errors

esp_err_t **esp_wifi_clear_fast_connect**(void)

Currently this API is just an stub API.

返回

- ESP_OK: succeed
- others: fail

esp_err_t **esp_wifi_deauth_sta**(uint16_t aid)

deauthenticate all stations or associated id equals to aid

参数 **aid** –when aid is 0, deauthenticate all stations, otherwise deauthenticate station whose associated id is aid

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_MODE: WiFi mode is wrong

esp_err_t **esp_wifi_scan_start**(const *wifi_scan_config_t* *config, bool block)

Scan all available APs.

Attention If this API is called, the found APs are stored in WiFi driver dynamic allocated memory and the will be freed in esp_wifi_scan_get_ap_records, so generally, call esp_wifi_scan_get_ap_records to cause the memory to be freed once the scan is done

Attention The values of maximum active scan time and passive scan time per channel are limited to 1500 milliseconds. Values above 1500ms may cause station to disconnect from AP and are not recommended.

参数

- **config** –configuration of scanning
- **block** –if block is true, this API will block the caller until the scan is done, otherwise it will return immediately

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
- ESP_ERR_WIFI_TIMEOUT: blocking scan is timeout
- ESP_ERR_WIFI_STATE: wifi still connecting when invoke esp_wifi_scan_start
- others: refer to error code in esp_err.h

esp_err_t **esp_wifi_scan_stop**(void)

Stop the scan in process.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start

esp_err_t **esp_wifi_scan_get_ap_num**(uint16_t *number)

Get number of APs found in last scan.

Attention This API can only be called when the scan is completed, otherwise it may get wrong value.

参数 **number** –**[out]** store number of APIs found in last scan

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_scan_get_ap_records**(uint16_t *number, *wifi_ap_record_t* *ap_records)

Get AP list found in last scan.

参数

- **number** –**[inout]** As input param, it stores max AP number ap_records can hold. As output param, it receives the actual AP number this API returns.
- **ap_records** –*wifi_ap_record_t* array to hold the found APs

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NO_MEM: out of memory

esp_err_t **esp_wifi_clear_ap_list**(void)

Clear AP list found in last scan.

Attention When the obtained ap list fails,bss info must be cleared,otherwise it may cause memory leakage.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_sta_get_ap_info**(*wifi_ap_record_t* *ap_info)

Get information of AP to which the device is associated with.

Attention When the obtained country information is empty, it means that the AP does not carry country information

参数 **ap_info** –the *wifi_ap_record_t* to hold AP information sta can get the connected ap' s phy mode info through the struct member phy_11b, phy_11g, phy_11n, phy_lr in the *wifi_ap_record_t* struct. For example, phy_11b = 1 imply that ap support 802.11b mode

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_CONN: The station interface don' t initialized
- ESP_ERR_WIFI_NOT_CONNECT: The station is in disconnect status

esp_err_t **esp_wifi_set_ps**(*wifi_ps_type_t* type)

Set current WiFi power save type.

Attention Default power save type is WIFI_PS_MIN_MODEM.

参数 **type** –power save type
返回 ESP_OK: succeed

esp_err_t **esp_wifi_get_ps**(*wifi_ps_type_t* *type)

Get current WiFi power save type.

Attention Default power save type is WIFI_PS_MIN_MODEM.

参数 **type** –**[out]** store current power save type
返回 ESP_OK: succeed

esp_err_t **esp_wifi_set_protocol**(*wifi_interface_t* ifx, uint8_t protocol_bitmap)

Set protocol type of specified interface The default protocol is (WIFI_PROTOCOL_11B|WIFI_PROTOCOL_11G|WIFI_PROT
if CONFIG_SOC_WIFI_HE_SUPPORT, the default protocol is (WIFI_PROTOCOL_11B|WIFI_PROTOCOL_11G|WIFI_PR

Attention Support 802.11b or 802.11bg or 802.11bgn or 802.11bgnax or LR mode

参数
- **ifx** –interfaces
- **protocol_bitmap** –WiFi protocol bitmap

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- others: refer to error codes in esp_err.h

esp_err_t **esp_wifi_get_protocol**(*wifi_interface_t* ifx, uint8_t *protocol_bitmap)

Get the current protocol bitmap of the specified interface.

参数
- **ifx** –interface
- **protocol_bitmap** –**[out]** store current WiFi protocol bitmap of interface ifx

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error codes in esp_err.h

esp_err_t **esp_wifi_set_bandwidth**(*wifi_interface_t* ifx, *wifi_bandwidth_t* bw)

Set the bandwidth of specified interface.

Attention 1. API return false if try to configure an interface that is not enabled
Attention 2. WIFI_BW_HT40 is supported only when the interface support 11N

参数
- **ifx** –interface to be configured
- **bw** –bandwidth

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error codes in esp_err.h

esp_err_t **esp_wifi_get_bandwidth**(*wifi_interface_t* ifx, *wifi_bandwidth_t* *bw)

Get the bandwidth of specified interface.

Attention 1. API return false if try to get a interface that is not enable

参数

- **ifx** –interface to be configured
- **bw** –**[out]** store bandwidth of interface ifx

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_set_channel**(uint8_t primary, *wifi_second_chan_t* second)

Set primary/secondary channel of device.

Attention 1. This API should be called after esp_wifi_start()

Attention 2. When device is in STA mode, this API should not be called when STA is scanning or connecting to an external AP

Attention 3. When device is in softAP mode, this API should not be called when softAP has connected to external STAs

Attention 4. When device is in STA+softAP mode, this API should not be called when in the scenarios described above

Attention 5. The channel info set by this API will not be stored in NVS. So If you want to remeber the channel used before wifi stop, you need to call this API again after wifi start, or you can call `esp_wifi_set_config()` to store the channel info in NVS.

参数

- **primary** –for HT20, primary is the channel number, for HT40, primary is the primary channel
- **second** –for HT20, second is ignored, for HT40, second is the second channel

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_get_channel**(uint8_t *primary, *wifi_second_chan_t* *second)

Get the primary/secondary channel of device.

Attention 1. API return false if try to get a interface that is not enable

参数

- **primary** –store current primary channel
- **second** –**[out]** store current second channel

返回

- ESP_OK: succeed

- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_set_country**(const *wifi_country_t* *country)

configure country info

Attention 1. It is discouraged to call this API since this doesn't validate the per-country rules, it's up to the user to fill in all fields according to local regulations. Please use esp_wifi_set_country_code instead.

Attention 2. The default country is "01" (world safe mode) {.cc=" 01" , .schan=1, .nchan=11, .policy=WIFI_COUNTRY_POLICY_AUTO}.

Attention 3. The third octect of country code string is one of the following: ' ', 'O', 'I', 'X', otherwise it is considered as ' '.

Attention 4. When the country policy is WIFI_COUNTRY_POLICY_AUTO, the country info of the AP to which the station is connected is used. E.g. if the configured country info is {.cc=" US" , .schan=1, .nchan=11} and the country info of the AP to which the station is connected is {.cc=" JP" , .schan=1, .nchan=14} then the country info that will be used is {.cc=" JP" , .schan=1, .nchan=14}. If the station disconnected from the AP the country info is set back to the country info of the station automatically, {.cc=" US" , .schan=1, .nchan=11} in the example.

Attention 5. When the country policy is WIFI_COUNTRY_POLICY_MANUAL, then the configured country info is used always.

Attention 6. When the country info is changed because of configuration or because the station connects to a different external AP, the country IE in probe response/beacon of the soft-AP is also changed.

Attention 7. The country configuration is stored into flash.

Attention 8. When this API is called, the PHY init data will switch to the PHY init data type corresponding to the country info.

参数 **country** –the configured country info

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_get_country**(*wifi_country_t* *country)

get the current country info

参数 **country** –country info

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_set_mac**(*wifi_interface_t* ifx, const uint8_t mac[6])

Set MAC address of WiFi station, soft-AP or NAN interface.

Attention 1. This API can only be called when the interface is disabled

Attention 2. Above mentioned interfaces have different MAC addresses, do not set them to be the same.

Attention 3. The bit 0 of the first byte of MAC address can not be 1. For example, the MAC address can set to be "1a:XX:XX:XX:XX:XX" , but can not be "15:XX:XX:XX:XX:XX" .

参数

- **ifx** –interface
- **mac** –the MAC address

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_WIFI_MAC: invalid mac address
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- others: refer to error codes in esp_err.h

esp_err_t **esp_wifi_get_mac**(*wifi_interface_t* ifx, uint8_t mac[6])

Get mac of specified interface.

参数

- **ifx** –interface
- **mac** –**[out]** store mac of the interface ifx

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_IF: invalid interface

esp_err_t **esp_wifi_set_promiscuous_rx_cb**(*wifi_promiscuous_cb_t* cb)

Register the RX callback function in the promiscuous mode.

Each time a packet is received, the registered callback function will be called.

参数 **cb** –callback

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_set_promiscuous**(bool en)

Enable the promiscuous mode.

参数 **en** –false - disable, true - enable

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_get_promiscuous**(bool *en)

Get the promiscuous mode.

参数 **en** –**[out]** store the current status of promiscuous mode

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_set_promiscuous_filter**(const *wifi_promiscuous_filter_t* *filter)

Enable the promiscuous mode packet type filter.

备注: The default filter is to filter all packets except WIFI_PKT_MISC

参数 **filter** –the packet type filtered in promiscuous mode.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_get_promiscuous_filter**(*wifi_promiscuous_filter_t* *filter)

Get the promiscuous filter.

参数 **filter** –**[out]** store the current status of promiscuous filter

返回

- ESP_OK: succeed

- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_set_promiscuous_ctrl_filter**(const *wifi_promiscuous_filter_t* *filter)

Enable subtype filter of the control packet in promiscuous mode.

备注: The default filter is to filter none control packet.

参数 **filter** –the subtype of the control packet filtered in promiscuous mode.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_get_promiscuous_ctrl_filter**(*wifi_promiscuous_filter_t* *filter)

Get the subtype filter of the control packet in promiscuous mode.

参数 **filter** –**[out]** store the current status of subtype filter of the control packet in promiscuous mode

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument

esp_err_t **esp_wifi_set_config**(*wifi_interface_t* interface, *wifi_config_t* *conf)

Set the configuration of the STA, AP or NAN.

Attention 1. This API can be called only when specified interface is enabled, otherwise, API fail

Attention 2. For station configuration, bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

Attention 3. ESP devices are limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the station.

Attention 4. The configuration will be stored in NVS for station and soft-AP

参数

- **interface** –interface
- **conf** –station, soft-AP or NAN configuration

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_WIFI_MODE: invalid mode
- ESP_ERR_WIFI_PASSWORD: invalid password
- ESP_ERR_WIFI_NVS: WiFi internal NVS error
- others: refer to the erro code in esp_err.h

esp_err_t **esp_wifi_get_config**(*wifi_interface_t* interface, *wifi_config_t* *conf)

Get configuration of specified interface.

参数

- **interface** –interface
- **conf** –**[out]** station or soft-AP configuration

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_IF: invalid interface

esp_err_t **esp_wifi_ap_get_sta_list**(*wifi_sta_list_t* *sta)

Get STAs associated with soft-AP.

Attention SSC only API

参数 **sta** –**[out]** station list ap can get the connected sta' s phy mode info through the struct member phy_11b, phy_11g, phy_11n, phy_lr in the *wifi_sta_info_t* struct. For example, phy_11b = 1 imply that sta support 802.11b mode

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- ESP_ERR_WIFI_CONN: WiFi internal error, the station/soft-AP control block is invalid

esp_err_t **esp_wifi_ap_get_sta_aid**(const uint8_t mac[6], uint16_t *aid)

Get AID of STA connected with soft-AP.

参数

- **mac** –STA' s mac address
- **aid** –**[out]** Store the AID corresponding to STA mac

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NOT_FOUND: Requested resource not found
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- ESP_ERR_WIFI_CONN: WiFi internal error, the station/soft-AP control block is invalid

esp_err_t **esp_wifi_set_storage**(*wifi_storage_t* storage)

Set the WiFi API configuration storage type.

Attention 1. The default value is WIFI_STORAGE_FLASH

参数 **storage** –: storage type

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **esp_wifi_set_vendor_ie**(bool enable, *wifi_vendor_ie_type_t* type, *wifi_vendor_ie_id_t* idx, const void *vnd_ie)

Set 802.11 Vendor-Specific Information Element.

参数

- **enable** –If true, specified IE is enabled. If false, specified IE is removed.
- **type** –Information Element type. Determines the frame type to associate with the IE.
- **idx** –Index to set or clear. Each IE type can be associated with up to two elements (indices 0 & 1).
- **vnd_ie** –Pointer to vendor specific element data. First 6 bytes should be a header with fields matching *vendor_ie_data_t*. If enable is false, this argument is ignored and can be NULL. Data does not need to remain valid after the function returns.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init()
- ESP_ERR_INVALID_ARG: Invalid argument, including if first byte of vnd_ie is not WIFI_VENDOR_IE_ELEMENT_ID (0xDD) or second byte is an invalid length.

- ESP_ERR_NO_MEM: Out of memory

esp_err_t **esp_wifi_set_vendor_ie_cb**(*esp_vendor_ie_cb_t* cb, void *ctx)

Register Vendor-Specific Information Element monitoring callback.

参数

- **cb** –Callback function
- **ctx** –Context argument, passed to callback function.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_set_max_tx_power**(int8_t power)

Set maximum transmitting power after WiFi start.

Attention 1. Maximum power before wifi startup is limited by PHY init data bin.

Attention 2. The value set by this API will be mapped to the max_tx_power of the structure *wifi_country_t* variable.

Attention 3. Mapping Table {Power, max_tx_power} = {{8, 2}, {20, 5}, {28, 7}, {34, 8}, {44, 11}, {52, 13}, {56, 14}, {60, 15}, {66, 16}, {72, 18}, {80, 20}}.

Attention 4. Param power unit is 0.25dBm, range is [8, 84] corresponding to 2dBm - 20dBm.

Attention 5. Relationship between set value and actual value. As follows: {set value range, actual value} = {{[8, 19],8}, {[20, 27],20}, {[28, 33],28}, {[34, 43],34}, {[44, 51],44}, {[52, 55],52}, {[56, 59],56}, {[60, 65],60}, {[66, 71],66}, {[72, 79],72}, {[80, 84],80}}.

参数 **power** –Maximum WiFi transmitting power.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is out of range

esp_err_t **esp_wifi_get_max_tx_power**(int8_t *power)

Get maximum transmiting power after WiFi start.

参数 **power** –Maximum WiFi transmitting power, unit is 0.25dBm.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_ARG: invalid argument

esp_err_t **esp_wifi_set_event_mask**(uint32_t mask)

Set mask to enable or disable some WiFi events.

Attention 1. Mask can be created by logical OR of various WIFI_EVENT_MASK_ constants. Events which have corresponding bit set in the mask will not be delivered to the system event handler.

Attention 2. Default WiFi event mask is WIFI_EVENT_MASK_AP_PROBEREQRECVED.

Attention 3. There may be lots of stations sending probe request data around. Don't unmask this event unless you need to receive probe request data.

参数 **mask** –WiFi event mask.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_get_event_mask**(uint32_t *mask)

Get mask of WiFi events.

参数 **mask** –WiFi event mask.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument

esp_err_t **esp_wifi_80211_tx**(*wifi_interface_t* ifx, const void *buffer, int len, bool en_sys_seq)

Send raw ieee80211 data.

Attention Currently only support for sending beacon/probe request/probe response/action and non-QoS data frame

参数

- **ifx** –interface if the Wi-Fi mode is Station, the ifx should be WIFI_IF_STA. If the Wi-Fi mode is SoftAP, the ifx should be WIFI_IF_AP. If the Wi-Fi mode is Station+SoftAP, the ifx should be WIFI_IF_STA or WIFI_IF_AP. If the ifx is wrong, the API returns ESP_ERR_WIFI_IF.
- **buffer** –raw ieee80211 buffer
- **len** –the length of raw buffer, the len must be <= 1500 Bytes and >= 24 Bytes
- **en_sys_seq** –indicate whether use the internal sequence number. If en_sys_seq is false, the sequence in raw buffer is unchanged, otherwise it will be overwritten by WiFi driver with the system sequence number. Generally, if esp_wifi_80211_tx is called before the Wi-Fi connection has been set up, both en_sys_seq==true and en_sys_seq==false are fine. However, if the API is called after the Wi-Fi connection has been set up, en_sys_seq must be true, otherwise ESP_ERR_WIFI_ARG is returned.

返回

- ESP_OK: success
- ESP_ERR_WIFI_IF: Invalid interface
- ESP_ERR_INVALID_ARG: Invalid parameter
- ESP_ERR_WIFI_NO_MEM: out of memory

esp_err_t **esp_wifi_set_csi_rx_cb**(*wifi_csi_cb_t* cb, void *ctx)

Register the RX callback function of CSI data.

```
Each time a CSI data is received, the callback function will be called.
```

参数

- **cb** –callback
- **ctx** –context argument, passed to callback function

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_set_csi_config**(const *wifi_csi_config_t* *config)

Set CSI data configuration.

return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start or promiscuous mode is not enabled

- ESP_ERR_INVALID_ARG: invalid argument

参数 config –configuration

esp_err_t **esp_wifi_set_csi**(bool en)

Enable or disable CSI.

return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start or promiscuous mode is not enabled
- ESP_ERR_INVALID_ARG: invalid argument

参数 en –true - enable, false - disable

esp_err_t **esp_wifi_set_ant_gpio**(const *wifi_ant_gpio_config_t* *config)

Set antenna GPIO configuration.

参数 config –Antenna GPIO configuration.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: Invalid argument, e.g. parameter is NULL, invalid GPIO number etc

esp_err_t **esp_wifi_get_ant_gpio**(*wifi_ant_gpio_config_t* *config)

Get current antenna GPIO configuration.

参数 config –Antenna GPIO configuration.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is NULL

esp_err_t **esp_wifi_set_ant**(const *wifi_ant_config_t* *config)

Set antenna configuration.

参数 config –Antenna configuration.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: Invalid argument, e.g. parameter is NULL, invalid antenna mode or invalid GPIO number

esp_err_t **esp_wifi_get_ant**(*wifi_ant_config_t* *config)

Get current antenna configuration.

参数 config –Antenna configuration.

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is NULL

int64_t **esp_wifi_get_tsf_time**(*wifi_interface_t* interface)

Get the TSF time In Station mode or SoftAP+Station mode if station is not connected or station doesn't receive at least one beacon after connected, will return 0.

Attention Enabling power save may cause the return value inaccurate, except WiFi modem sleep

参数 **interface** –The interface whose tsf_time is to be retrieved.
返回 0 or the TSF time

esp_err_t **esp_wifi_set_inactive_time**(*wifi_interface_t* ifx, uint16_t sec)

Set the inactive time of the STA or AP.

Attention 1. For Station, If the station does not receive a beacon frame from the connected SoftAP during the inactive time, disconnect from SoftAP. Default 6s.
Attention 2. For SoftAP, If the softAP doesn’ t receive any data from the connected STA during inactive time, the softAP will force deauth the STA. Default is 300s.
Attention 3. The inactive time configuration is not stored into flash

参数
- **ifx** –interface to be configured.
- **sec** –Inactive time. Unit seconds.

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_ARG: invalid argument, For Station, if sec is less than 3. For SoftAP, if sec is less than 10.

esp_err_t **esp_wifi_get_inactive_time**(*wifi_interface_t* ifx, uint16_t *sec)

Get inactive time of specified interface.

参数
- **ifx** –Interface to be configured.
- **sec** –Inactive time. Unit seconds.

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_ARG: invalid argument

esp_err_t **esp_wifi_statis_dump**(uint32_t modules)

Dump WiFi statistics.

参数 **modules** –statistic modules to be dumped
返回
- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_set_rssi_threshold**(int32_t rssi)

Set RSSI threshold below which APP will get an event.

Attention This API needs to be called every time after WIFI_EVENT_STA_BSS_RSSI_LOW event is received.

参数 **rssi** –threshold value in dbm between -100 to 0
返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument

esp_err_t **esp_wifi_ftm_initiate_session**(*wifi_ftm_initiator_cfg_t* *cfg)

Start an FTM Initiator session by sending FTM request If successful, event WIFI_EVENT_FTM_REPORT is generated with the result of the FTM procedure.

Attention 1. Use this API only in Station mode.

Attention 2. If FTM is initiated on a different channel than Station is connected in or internal SoftAP is started in, FTM defaults to a single burst in ASAP mode.

参数 **cfg** –FTM Initiator session configuration

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_ftm_end_session**(void)

End the ongoing FTM Initiator session.

Attention This API works only on FTM Initiator

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_ftm_resp_set_offset**(int16_t offset_cm)

Set offset in cm for FTM Responder. An equivalent offset is calculated in picoseconds and added in TOD of FTM Measurement frame (T1).

Attention Use this API only in AP mode before performing FTM as responder

参数 **offset_cm** –T1 Offset to be added in centimeters

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_config_11b_rate**(*wifi_interface_t* ifx, bool disable)

Enable or disable 11b rate of specified interface.

Attention 1. This API should be called after esp_wifi_init() and before esp_wifi_start().

Attention 2. Only when really need to disable 11b rate call this API otherwise don't call this.

参数

- **ifx** –Interface to be configured.
- **disable** –true means disable 11b rate while false means enable 11b rate.

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_connectionless_module_set_wake_interval**(uint16_t wake_interval)

Set wake interval for connectionless modules to wake up periodically.

Attention 1. Only one wake interval for all connectionless modules.

Attention 2. This configuration could work at connected status. When ESP_WIFI_STA_DISCONNECTED_PM_ENABLE is enabled, this configuration could work at disconnected status.

Attention 3. Event WIFI_EVENT_CONNECTIONLESS_MODULE_WAKE_INTERVAL_START would be posted each time wake interval starts.

Attention 4. Recommend to configure interval in multiples of hundred. (e.g. 100ms)

Attention 5. Recommend to configure interval to ESP_WIFI_CONNECTIONLESS_INTERVAL_DEFAULT_MODE to get stable performance at coexistence mode.

参数 `wake_interval` –Milliseconds after would the chip wake up, from 1 to 65535.

esp_err_t **`esp_wifi_set_country_code`**(const char *country, bool ieee80211d_enabled)

configure country

Attention 1. When ieee80211d_enabled, the country info of the AP to which the station is connected is used. E.g. if the configured country is US and the country info of the AP to which the station is connected is JP then the country info that will be used is JP. If the station disconnected from the AP the country info is set back to the country info of the station automatically, US in the example.

Attention 2. When ieee80211d_enabled is disabled, then the configured country info is used always.

Attention 3. When the country info is changed because of configuration or because the station connects to a different external AP, the country IE in probe response/beacon of the soft-AP is also changed.

Attention 4. The country configuration is stored into flash.

Attention 5. When this API is called, the PHY init data will switch to the PHY init data type corresponding to the country info.

Attention 6. Supported country codes are "01" (world safe mode) "AT" ," AU" ," BE" ," BG" ," BR" , "CA" ," CH" ," CN" ," CY" ," CZ" ," DE" ," DK" ," EE" ," ES" ," FI" ," FR" ," GB" ," GR" ," HK" ," HR" ," HU" , "IE" ," IN" ," IS" ," IT" ," JP" ," KR" ," LI" ," LT" ," LU" ," LV" ," MT" ," MX" ," NL" ," NO" ," NZ" ," PL" ," PT" , "RO" ," SE" ," SI" ," SK" ," TW" ," US"

Attention 7. When country code "01" (world safe mode) is set, SoftAP mode won't contain country IE.

Attention 8. The default country is "01" (world safe mode) and ieee80211d_enabled is TRUE.

Attention 9. The third octect of country code string is one of the following: ' ', 'O', 'I', 'X', otherwise it is considered as ' '.

参数

- **`country`** –the configured country ISO code
- **`ieee80211d_enabled`** –802.11d is enabled or not

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **`esp_wifi_get_country_code`**(char *country)

get the current country code

参数 `country` –country code

返回

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

esp_err_t **`esp_wifi_config_80211_tx_rate`**(*wifi_interface_t* ifx, *wifi_phy_rate_t* rate)

Config 80211 tx rate of specified interface.

Attention 1. This API should be called after esp_wifi_init() and before esp_wifi_start().

参数

- **ifx** –Interface to be configured.
- **rate** –Phy rate to be configured.

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_disable_pmf_config**(*wifi_interface_t* ifx)

Disable PMF configuration for specified interface.

Attention This API should be called after esp_wifi_set_config() and before esp_wifi_start().

参数 **ifx** –Interface to be configured.

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_sta_get_aid**(uint16_t *aid)

Get the Association id assigned to STA by AP.

Attention aid = 0 if station is not connected to AP.

参数 **aid** –**[out]** store the aid

返回

- ESP_OK: succeed

esp_err_t **esp_wifi_sta_get_negotiated_phymode**(*wifi_phy_mode_t* *phymode)

Get the negotiated phymode after connection.

Attention Operation phy mode, BIT[5]: indicate whether LR enabled, BIT[0-4]: wifi_phy_mode_t

参数 **phymode** –**[out]** store the negotiated phymode.

返回

- ESP_OK: succeed

esp_err_t **esp_wifi_set_dynamic_cs**(bool enabled)

Config dynamic carrier sense.

Attention This API should be called after esp_wifi_start().

参数 **enabled** –Dynamic carrier sense is enabled or not.

返回

- ESP_OK: succeed
- others: failed

Structures

struct **wifi_init_config_t**

WiFi stack configuration parameters passed to esp_wifi_init call.

Public Members

wifi_osi_funcs_t *__osi_funcs__

WiFi OS functions

wpa_crypto_funcs_t __wpa_crypto_funcs__

WiFi station crypto functions when connect

int __static_rx_buf_num__

WiFi static RX buffer number

int __dynamic_rx_buf_num__

WiFi dynamic RX buffer number

int __tx_buf_type__

WiFi TX buffer type

int __static_tx_buf_num__

WiFi static TX buffer number

int __dynamic_tx_buf_num__

WiFi dynamic TX buffer number

int __cache_tx_buf_num__

WiFi TX cache buffer number

int __csi_enable__

WiFi channel state information enable flag

int __ampdu_rx_enable__

WiFi AMPDU RX feature enable flag

int __ampdu_tx_enable__

WiFi AMPDU TX feature enable flag

int __amsdu_tx_enable__

WiFi AMSDU TX feature enable flag

int __nvs_enable__

WiFi NVS flash enable flag

int __nano_enable__

Nano option for printf/scan family enable flag

int __rx_ba_win__

WiFi Block Ack RX window size

int __wifi_task_core_id__

WiFi Task Core ID

int **beacon_max_len**

WiFi softAP maximum length of the beacon

int **mgmt_sbuf_num**

WiFi management short buffer number, the minimum value is 6, the maximum value is 32

uint64_t **feature_caps**

Enables additional WiFi features and capabilities

bool **sta_disconnected_pm**

WiFi Power Management for station at disconnected status

int **espnow_max_encrypt_num**

Maximum encrypt number of peers supported by espnow

int **magic**

WiFi init magic number, it should be the last field

Macros

ESP_ERR_WIFI_NOT_INIT

WiFi driver was not installed by esp_wifi_init

ESP_ERR_WIFI_NOT_STARTED

WiFi driver was not started by esp_wifi_start

ESP_ERR_WIFI_NOT_STOPPED

WiFi driver was not stopped by esp_wifi_stop

ESP_ERR_WIFI_IF

WiFi interface error

ESP_ERR_WIFI_MODE

WiFi mode error

ESP_ERR_WIFI_STATE

WiFi internal state error

ESP_ERR_WIFI_CONN

WiFi internal control block of station or soft-AP error

ESP_ERR_WIFI_NVS

WiFi internal NVS module error

ESP_ERR_WIFI_MAC

MAC address is invalid

ESP_ERR_WIFI_SSID

SSID is invalid

ESP_ERR_WIFI_PASSWORD

Password is invalid

ESP_ERR_WIFI_TIMEOUT

Timeout error

ESP_ERR_WIFI_WAKE_FAIL

WiFi is in sleep state(RF closed) and wakeup fail

ESP_ERR_WIFI_WOULD_BLOCK

The caller would block

ESP_ERR_WIFI_NOT_CONNECT

Station still in disconnect status

ESP_ERR_WIFI_POST

Failed to post the event to WiFi task

ESP_ERR_WIFI_INIT_STATE

Invalid WiFi state when init/deinit is called

ESP_ERR_WIFI_STOP_STATE

Returned when WiFi is stopping

ESP_ERR_WIFI_NOT_ASSOC

The WiFi connection is not associated

ESP_ERR_WIFI_TX_DISALLOW

The WiFi TX is disallowed

ESP_ERR_WIFI_TWT_FULL

no available flow id

ESP_ERR_WIFI_TWT_SETUP_TIMEOUT

Timeout of receiving twt setup response frame, timeout times can be set during twt setup

WIFI_STATIC_TX_BUFFER_NUM

WIFI_CACHE_TX_BUFFER_NUM

WIFI_DYNAMIC_TX_BUFFER_NUM

WIFI_CSI_ENABLED

WIFI_AMPDU_RX_ENABLED

WIFI_AMPDU_TX_ENABLED

WIFI_AMSDU_TX_ENABLED

WIFI_NVS_ENABLED

WIFI_NANO_FORMAT_ENABLED

WIFI_INIT_CONFIG_MAGIC

WIFI_DEFAULT_RX_BA_WIN

WIFI_TASK_CORE_ID

WIFI_SOFTAP_BEACON_MAX_LEN

WIFI_MGMT_SBUF_NUM

WIFI_STA_DISCONNECTED_PM_ENABLED

CONFIG_FEATURE_WPA3_SAE_BIT

CONFIG_FEATURE_CACHE_TX_BUF_BIT

CONFIG_FEATURE_FTM_INITIATOR_BIT

CONFIG_FEATURE_FTM_RESPONDER_BIT

WIFI_INIT_CONFIG_DEFAULT()

ESP_WIFI_CONNECTIONLESS_INTERVAL_DEFAULT_MODE

Type Definitions

typedef void (***wifi_promiscuous_cb_t**)(void *buf, *wifi_promiscuous_pkt_type_t* type)

The RX callback function in the promiscuous mode. Each time a packet is received, the callback function will be called.

Param buf Data received. Type of data in buffer (*wifi_promiscuous_pkt_t* or *wifi_pkt_rx_ctrl_t*) indicated by 'type' parameter.
Param type promiscuous packet type.

typedef void (***esp_vendor_ie_cb_t**)(void *ctx, *wifi_vendor_ie_type_t* type, const uint8_t sa[6], const *vendor_ie_data_t* *vnd_ie, int rssi)

Function signature for received Vendor-Specific Information Element callback.

Param ctx Context argument, as passed to esp_wifi_set_vendor_ie_cb() when registering callback.
Param type Information element type, based on frame type received.
Param sa Source 802.11 address.
Param vnd_ie Pointer to the vendor specific element data received.
Param rssi Received signal strength indication.

typedef void (***wifi_csi_cb_t**)(void *ctx, *wifi_csi_info_t* *data)

The RX callback function of Channel State Information(CSI) data.

```
Each time a CSI data is received, the callback function will be called.
```

Param ctx context argument, passed to esp_wifi_set_csi_rx_cb() when registering callback function.

Param data CSI data received. The memory that it points to will be deallocated after callback function returns.

Header File

- components/esp_wifi/include/esp_wifi_types.h

Unions

union **wifi_config_t**

#include <esp_wifi_types.h> Configuration data for device' s AP or STA or NAN.

The usage of this union (for ap, sta or nan configuration) is determined by the accompanying interface argument passed to esp_wifi_set_config() or esp_wifi_get_config()

Public Members

wifi_ap_config_t **ap**

configuration of AP

wifi_sta_config_t **sta**

configuration of STA

wifi_nan_config_t **nan**

configuration of NAN

Structures

struct **wifi_country_t**

Structure describing WiFi country-based regional restrictions.

Public Members

char **cc**[3]

country code string

uint8_t **schan**

start channel

uint8_t **nchan**

total channel number

int8_t **max_tx_power**

This field is used for getting WiFi maximum transmitting power, call esp_wifi_set_max_tx_power to set the maximum transmitting power.

wifi_country_policy_t **policy**

country policy

struct **wifi_active_scan_time_t**

Range of active scan times per channel.

Public Members

uint32_t **min**

minimum active scan time per channel, units: millisecond

uint32_t **max**

maximum active scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

struct **wifi_scan_time_t**

Aggregate of active & passive scan time per channel.

Public Members

wifi_active_scan_time_t **active**

active scan time per channel, units: millisecond.

uint32_t **passive**

passive scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

struct **wifi_scan_config_t**

Parameters for an SSID scan.

Public Members

uint8_t ***ssid**

SSID of AP

uint8_t ***bssid**

MAC address of AP

uint8_t **channel**

channel, scan the specific channel

bool **show_hidden**

enable to scan AP whose SSID is hidden

wifi_scan_type_t **scan_type**

scan type, active or passive

wifi_scan_time_t **scan_time**

scan time per channel

struct **wifi_he_ap_info_t**

Description of a WiFi AP HE Info.

Public Members

uint8_t **bss_color**

an unsigned integer whose value is the BSS Color of the BSS corresponding to the AP

uint8_t **partial_bss_color**

indicate if an AID assignment rule based on the BSS color

uint8_t **bss_color_disabled**

indicate if the use of BSS color is disabled

uint8_t **bssid_index**

in M-BSSID set, identifies the nontransmitted BSSID

struct **wifi_ap_record_t**

Description of a WiFi AP.

Public Members

uint8_t **bssid**[6]

MAC address of AP

uint8_t **ssid**[33]

SSID of AP

uint8_t **primary**

channel of AP

wifi_second_chan_t **second**

secondary channel of AP

int8_t **rssi**

signal strength of AP

wifi_auth_mode_t **authmode**

authmode of AP

wifi_cipher_type_t **pairwise_cipher**

pairwise cipher of AP

wifi_cipher_type_t **group_cipher**

group cipher of AP

wifi_ant_t **ant**

antenna used to receive beacon from AP

uint32_t **phy_11b**

bit: 0 flag to identify if 11b mode is enabled or not

uint32_t **phy_11g**

bit: 1 flag to identify if 11g mode is enabled or not

uint32_t **phy_11n**

bit: 2 flag to identify if 11n mode is enabled or not

uint32_t **phy_lr**

bit: 3 flag to identify if low rate is enabled or not

uint32_t **phy_11ax**

bit: 4 flag to identify if 11ax mode is enabled or not

uint32_t **wps**

bit: 5 flag to identify if WPS is supported or not

uint32_t **ftm_responder**

bit: 6 flag to identify if FTM is supported in responder mode

uint32_t **ftm_initiator**

bit: 7 flag to identify if FTM is supported in initiator mode

uint32_t **reserved**

bit: 8..31 reserved

wifi_country_t **country**

country information of AP

wifi_he_ap_info_t **he_ap**

HE AP info

struct **wifi_scan_threshold_t**

Structure describing parameters for a WiFi fast scan.

Public Members

int8_t **rssi**

The minimum rssi to accept in the fast scan mode

wifi_auth_mode_t **authmode**

The weakest authmode to accept in the fast scan mode Note: Incase this value is not set and password is set as per WPA2 standards(password len >= 8), it will be defaulted to WPA2 and device won't connect to deprecated WEP/WPA networks. Please set authmode threshold as WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK to connect to WEP/WPA networks

struct **wifi_pmf_config_t**

Configuration structure for Protected Management Frame

Public Members

bool **capable**

Deprecated variable. Device will always connect in PMF mode if other device also advertizes PMF capability.

bool **required**

Advertizes that Protected Management Frame is required. Device will not associate to non-PMF capable devices.

struct **wifi_ap_config_t**

Soft-AP configuration settings for the device.

Public Members

uint8_t **ssid**[32]

SSID of soft-AP. If ssid_len field is 0, this must be a Null terminated string. Otherwise, length is set according to ssid_len.

uint8_t **password**[64]

Password of soft-AP.

uint8_t **ssid_len**

Optional length of SSID field.

uint8_t **channel**

Channel of soft-AP

wifi_auth_mode_t **authmode**

Auth mode of soft-AP. Do not support AUTH_WEP, AUTH_WAPI_PSK and AUTH_OWE in soft-AP mode. When the auth mode is set to WPA2_PSK, WPA2_WPA3_PSK or WPA3_PSK, the pairwise cipher will be overwritten with WIFI_CIPHER_TYPE_CCMP.

uint8_t **ssid_hidden**

Broadcast SSID or not, default 0, broadcast the SSID

uint8_t **max_connection**

Max number of stations allowed to connect in

uint16_t **beacon_interval**

Beacon interval which should be multiples of 100. Unit: TU(time unit, 1 TU = 1024 us). Range: 100 ~ 60000. Default value: 100

wifi_cipher_type_t **pairwise_cipher**

Pairwise cipher of SoftAP, group cipher will be derived using this. Cipher values are valid starting from WIFI_CIPHER_TYPE_TKIP, enum values before that will be considered as invalid and default cipher suites(TKIP+CCMP) will be used. Valid cipher suites in softAP mode are WIFI_CIPHER_TYPE_TKIP, WIFI_CIPHER_TYPE_CCMP and WIFI_CIPHER_TYPE_TKIP_CCMP.

bool **ftm_responder**

Enable FTM Responder mode

wifi_pmf_config_t **pmf_cfg**

Configuration for Protected Management Frame

wifi_sae_pwe_method_t **sae_pwe_h2e**

Configuration for SAE PWE derivation method

struct **wifi_sta_config_t**

STA configuration settings for the device.

Public Members

uint8_t **ssid**[32]

SSID of target AP.

uint8_t **password**[64]

Password of target AP.

wifi_scan_method_t **scan_method**

do all channel scan or fast scan

bool **bssid_set**

whether set MAC address of target AP or not. Generally, station_config.bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

uint8_t **bssid**[6]

MAC address of target AP

uint8_t **channel**

channel of target AP. Set to 1~13 to scan starting from the specified channel before connecting to AP. If the channel of AP is unknown, set it to 0.

uint16_t **listen_interval**

Listen interval for ESP32 station to receive beacon when WIFI_PS_MAX_MODEM is set. Units: AP beacon intervals. Defaults to 3 if set to 0.

wifi_sort_method_t **sort_method**

sort the connect AP in the list by rssi or security mode

wifi_scan_threshold_t **threshold**

When sort_method is set, only APs which have an auth mode that is more secure than the selected auth mode and a signal stronger than the minimum RSSI will be used.

wifi_pmf_config_t **pmf_cfg**

Configuration for Protected Management Frame. Will be advertised in RSN Capabilities in RSN IE.

uint32_t **rm_enabled**

Whether Radio Measurements are enabled for the connection

uint32_t **btm_enabled**

Whether BSS Transition Management is enabled for the connection

uint32_t **mbo_enabled**

Whether MBO is enabled for the connection

uint32_t **ft_enabled**

Whether FT is enabled for the connection

uint32_t **owe_enabled**

Whether OWE is enabled for the connection

uint32_t **transition_disable**

Whether to enable transition disable feature

uint32_t **reserved**

Reserved for future feature set

wifi_sae_pwe_method_t **sae_pwe_h2e**

Configuration for SAE PWE derivation method

wifi_sae_pk_mode_t **sae_pk_mode**

Configuration for SAE-PK (Public Key) Authentication method

uint8_t **failure_retry_cnt**

Number of connection retries station will do before moving to next AP. scan_method should be set as WIFI_ALL_CHANNEL_SCAN to use this config. Note: Enabling this may cause connection time to increase incase best AP doesn’ t behave properly.

uint32_t **he_dcm_set**

Whether DCM max.constellation for transmission and reception is set.

uint32_t **he_dcm_max_constellation_tx**

Indicate the max.constellation for DCM in TB PPDU the STA supported. 0: not supported. 1: BPSK, 2: QPSK, 3: 16-QAM. The default value is 3.

uint32_t **he_dcm_max_constellation_rx**

Indicate the max.constellation for DCM in both Data field and HE-SIG-B field the STA supported. 0: not supported. 1: BPSK, 2: QPSK, 3: 16-QAM. The default value is 3.

uint32_t **he_mcs9_enabled**

Whether to support HE-MCS 0 to 9. The default value is 0.

uint32_t **he_su_beamformee_disabled**

Whether to disable support for operation as an SU beamformee.

uint32_t **he_trig_su_bmforming_feedback_disabled**

Whether to disable support the transmission of SU feedback in an HE TB sounding sequence.

uint32_t **he_trig_mu_bmforming_partial_feedback_disabled**

Whether to disable support the transmission of partial-bandwidth MU feedback in an HE TB sounding sequence.

uint32_t **he_trig_cqi_feedback_disabled**

Whether to disable support the transmission of CQI feedback in an HE TB sounding sequence.

uint32_t **he_reserved**

Reserved for future feature set

uint8_t **sae_h2e_identifier**[SAE_H2E_IDENTIFIER_LEN]

Password identifier for H2E. this needs to be null terminated string

struct **wifi_nan_config_t**

NAN Discovery start configuration.

Public Members

uint8_t **op_channel**

NAN Discovery operating channel

uint8_t **master_pref**

Device’ s preference value to serve as NAN Master

uint8_t **scan_time**

Scan time in seconds while searching for a NAN cluster

uint16_t **warm_up_sec**

Warm up time before assuming NAN Anchor Master role

struct **wifi_sta_info_t**

Description of STA associated with AP.

Public Members

uint8_t **mac**[6]
mac address

int8_t **rssi**
current average rssi of sta connected

uint32_t **phy_11b**
bit: 0 flag to identify if 11b mode is enabled or not

uint32_t **phy_11g**
bit: 1 flag to identify if 11g mode is enabled or not

uint32_t **phy_11n**
bit: 2 flag to identify if 11n mode is enabled or not

uint32_t **phy_lr**
bit: 3 flag to identify if low rate is enabled or not

uint32_t **phy_11ax**
bit: 4 flag to identify if 11ax mode is enabled or not

uint32_t **is_mesh_child**
bit: 5 flag to identify mesh child

uint32_t **reserved**
bit: 6..31 reserved

struct **wifi_sta_list_t**
List of stations associated with the Soft-AP.

Public Members

wifi_sta_info_t **sta**[ESP_WIFI_MAX_CONN_NUM]
station list

int **num**
number of stations in the list (other entries are invalid)

struct **vendor_ie_data_t**
Vendor Information Element header.

The first bytes of the Information Element will match this header. Payload follows.

Public Members

uint8_t **element_id**

Should be set to WIFI_VENDOR_IE_ELEMENT_ID (0xDD)

uint8_t **length**

Length of all bytes in the element data following this field. Minimum 4.

uint8_t **vendor_oui**[3]

Vendor identifier (OUI).

uint8_t **vendor_oui_type**

Vendor-specific OUI type.

uint8_t **payload**[0]

Payload. Length is equal to value in ‘length’ field, minus 4.

struct **wifi_pkt_rx_ctrl_t**

Received packet radio metadata header, this is the common header at the beginning of all promiscuous mode RX callback buffers.

Public Members

signed **rssi**

Received Signal Strength Indicator(RSSI) of packet. unit: dBm

unsigned **rate**

PHY rate encoding of the packet. Only valid for non HT(11bg) packet

unsigned **__pad0__**

reserved

unsigned **sig_mode**

0: non HT(11bg) packet; 1: HT(11n) packet; 3: VHT(11ac) packet

unsigned **__pad1__**

reserved

unsigned **mcs**

Modulation Coding Scheme. If is HT(11n) packet, shows the modulation, range from 0 to 76(MSC0 ~ MCS76)

unsigned **cwb**

Channel Bandwidth of the packet. 0: 20MHz; 1: 40MHz

unsigned **__pad2__**

reserved

unsigned **smoothing**

reserved

unsigned **not_sounding**

reserved

unsigned **__pad3__**

reserved

unsigned **aggregation**

Aggregation. 0: MPDU packet; 1: AMPDU packet

unsigned **stbc**

Space Time Block Code(STBC). 0: non STBC packet; 1: STBC packet

unsigned **fec_coding**

Flag is set for 11n packets which are LDPC

unsigned **sgi**

Short Guide Interval(SGI). 0: Long GI; 1: Short GI

unsigned **__pad4__**

reserved

unsigned **ampdu_cnt**

ampdu cnt

unsigned **channel**

primary channel on which this packet is received

unsigned **secondary_channel**

secondary channel on which this packet is received. 0: none; 1: above; 2: below

unsigned **__pad5__**

reserved

unsigned **timestamp**

timestamp. The local time when this packet is received. It is precise only if modem sleep or light sleep is not enabled. unit: microsecond

unsigned **__pad6__**

reserved

signed **noise_floor**

noise floor of Radio Frequency Module(RF). unit: dBm

unsigned **__pad7__**

reserved

unsigned **__pad8__**

reserved

unsigned **__pad9__**

reserved

unsigned **ant**

antenna number from which this packet is received. 0: WiFi antenna 0; 1: WiFi antenna 1

unsigned **__pad10__**

reserved

unsigned **__pad11__**

reserved

unsigned **__pad12__**

reserved

unsigned **sig_len**

length of packet including Frame Check Sequence(FCS)

unsigned **__pad13__**

reserved

unsigned **rx_state**

state of the packet. 0: no error; others: error numbers which are not public

struct **wifi_promiscuous_pkt_t**

Payload passed to ‘buf’ parameter of promiscuous mode RX callback.

Public Members

wifi_pkt_rx_ctrl_t **rx_ctrl**

metadata header

uint8_t **payload**[0]

Data or management payload. Length of payload is described by rx_ctrl.sig_len. Type of content determined by packet type argument of callback.

struct **wifi_promiscuous_filter_t**

Mask for filtering different packet types in promiscuous mode.

Public Members

uint32_t **filter_mask**

OR of one or more filter values WIFI_PROMIS_FILTER_*

struct **wifi_csi_config_t**

Channel state information(CSI) configuration type.

Public Members

bool **lltf_en**

enable to receive legacy long training field(lltf) data. Default enabled

bool **htltf_en**

enable to receive HT long training field(htltf) data. Default enabled

bool **stbc_htltf2_en**

enable to receive space time block code HT long training field(stbc-htltf2) data. Default enabled

bool **ltf_merge_en**

enable to generate htlft data by averaging lltf and ht_ltf data when receiving HT packet. Otherwise, use ht_ltf data directly. Default enabled

bool **channel_filter_en**

enable to turn on channel filter to smooth adjacent sub-carrier. Disable it to keep independence of adjacent sub-carrier. Default enabled

bool **manu_scale**

manually scale the CSI data by left shifting or automatically scale the CSI data. If set true, please set the shift bits. false: automatically. true: manually. Default false

uint8_t **shift**

manually left shift bits of the scale of the CSI data. The range of the left shift bits is 0~15

struct **wifi_csi_info_t**

CSI data type.

Public Members

wifi_pkt_rx_ctrl_t **rx_ctrl**

received packet radio metadata header of the CSI data

uint8_t **mac**[6]

source MAC address of the CSI data

uint8_t **dmac**[6]

destination MAC address of the CSI data

bool **first_word_invalid**

first four bytes of the CSI data is invalid or not

int8_t ***buf**

buffer of CSI data

uint16_t **len**

length of CSI data

struct **wifi_ant_gpio_t**

WiFi GPIO configuration for antenna selection.

Public Members

uint8_t **gpio_select**

Whether this GPIO is connected to external antenna switch

uint8_t **gpio_num**

The GPIO number that connects to external antenna switch

struct **wifi_ant_gpio_config_t**

WiFi GPIOs configuration for antenna selection.

Public Members

wifi_ant_gpio_t **gpio_cfg**[4]

The configurations of GPIOs that connect to external antenna switch

struct **wifi_ant_config_t**

WiFi antenna configuration.

Public Members

wifi_ant_mode_t **rx_ant_mode**

WiFi antenna mode for receiving

wifi_ant_t **rx_ant_default**

Default antenna mode for receiving, it’ s ignored if rx_ant_mode is not WIFI_ANT_MODE_AUTO

wifi_ant_mode_t **tx_ant_mode**

WiFi antenna mode for transmission, it can be set to WIFI_ANT_MODE_AUTO only if rx_ant_mode is set to WIFI_ANT_MODE_AUTO

uint8_t **enabled_ant0**

Index (in antenna GPIO configuration) of enabled WIFI_ANT_MODE_ANT0

uint8_t **enabled_ant1**

Index (in antenna GPIO configuration) of enabled WIFI_ANT_MODE_ANT1

struct **wifi_action_tx_req_t**

Action Frame Tx Request.

Public Members

wifi_interface_t **ifx**

WiFi interface to send request to

uint8_t **dest_mac**[6]

Destination MAC address

bool **no_ack**

Indicates no ack required

wifi_action_rx_cb_t **rx_cb**

Rx Callback to receive any response

uint32_t **data_len**

Length of the appended Data

uint8_t **data**[0]

Appended Data payload

struct **wifi_ftm_initiator_cfg_t**

FTM Initiator configuration.

Public Members

uint8_t **resp_mac**[6]

MAC address of the FTM Responder

uint8_t **channel**

Primary channel of the FTM Responder

uint8_t **frm_count**

No. of FTM frames requested in terms of 4 or 8 bursts (allowed values - 0(No pref), 16, 24, 32, 64)

uint16_t **burst_period**

Requested time period between consecutive FTM bursts in 100’s of milliseconds (0 - No pref)

struct **wifi_beacon_monitor_config_t**

WiFi beacon monitor parameter configuration.

Public Members

bool **enable**

Enable or disable beacon monitor

uint8_t **loss_timeout**

Beacon lost timeout

uint8_t **loss_threshold**

Maximum number of consecutive lost beacons allowed

uint8_t **delta_intr_early**

Delta early time for RF PHY on

uint8_t **delta_loss_timeout**

Delta timeout time for RF PHY off

struct **wifi_nan_publish_cfg_t**

NAN Publish service configuration parameters.

Public Members

char **service_name**[ESP_WIFI_MAX_SVC_NAME_LEN]

Service name identifier

wifi_nan_service_type_t **type**

Service type

char **matching_filter**[ESP_WIFI_MAX_FILTER_LEN]

Comma separated filters for filtering services

char **svc_info**[ESP_WIFI_MAX_SVC_INFO_LEN]

Service info shared in Publish frame

uint8_t **single_replied_event**

Give single Replied event or every time

uint8_t **datapath_reqd**

NAN Datapath required for the service

uint8_t **reserved**

Reserved

struct **wifi_nan_subscribe_cfg_t**

NAN Subscribe service configuration parameters.

Public Members

char **service_name**[ESP_WIFI_MAX_SVC_NAME_LEN]

Service name identifier

wifi_nan_service_type_t **type**

Service type

char **matching_filter**[ESP_WIFI_MAX_FILTER_LEN]

Comma separated filters for filtering services

char **svc_info**[ESP_WIFI_MAX_SVC_INFO_LEN]

Service info shared in Subscribe frame

uint8_t **single_match_event**

Give single Match event or every time

uint8_t **reserved**

Reserved

struct **wifi_nan_followup_params_t**

NAN Follow-up parameters.

Public Members

uint8_t **inst_id**

Own service instance id

uint8_t **peer_inst_id**

Peer’ s service instance id

uint8_t **peer_mac**[6]

Peer’ s MAC address

char **svc_info**[ESP_WIFI_MAX_SVC_INFO_LEN]

Service info(or message) to be shared

struct **wifi_nan_datapath_req_t**

NAN Datapath Request parameters.

Public Members

uint8_t **pub_id**

Publisher’ s service instance id

uint8_t **peer_mac**[6]

Peer’ s MAC address

bool **confirm_required**

NDP Confirm frame required

struct **wifi_nan_datapath_resp_t**

NAN Datapath Response parameters.

Public Members

bool **accept**

True - Accept incoming NDP, False - Reject it

uint8_t **ndp_id**

NAN Datapath Identifier

uint8_t **peer_mac**[6]

Peer’ s MAC address

struct **wifi_nan_datapath_end_req_t**

NAN Datapath End parameters.

Public Members

uint8_t **ndp_id**

NAN Datapath Identifier

uint8_t **peer_mac**[6]

Peer’ s MAC address

struct **wifi_event_sta_scan_done_t**

Argument structure for WIFI_EVENT_SCAN_DONE event

Public Members

uint32_t **status**

status of scanning APs: 0 —success, 1 - failure

uint8_t **number**

number of scan results

uint8_t **scan_id**

scan sequence number, used for block scan

struct **wifi_event_sta_connected_t**

Argument structure for WIFI_EVENT_STA_CONNECTED event

Public Members

uint8_t **ssid**[32]

SSID of connected AP

uint8_t **ssid_len**

SSID length of connected AP

uint8_t **bssid**[6]

BSSID of connected AP

uint8_t **channel**

channel of connected AP

wifi_auth_mode_t **authmode**

authentication mode used by AP

uint16_t **aid**

authentication id assigned by the connected AP

struct **wifi_event_sta_disconnected_t**

Argument structure for WIFI_EVENT_STA_DISCONNECTED event

Public Members

uint8_t **ssid**[32]

SSID of disconnected AP

uint8_t **ssid_len**

SSID length of disconnected AP

uint8_t **bssid**[6]

BSSID of disconnected AP

uint8_t **reason**

reason of disconnection

int8_t **rssi**

rssi of disconnection

struct **wifi_event_sta_authmode_change_t**

Argument structure for WIFI_EVENT_STA_AUTHMODE_CHANGE event

Public Members

wifi_auth_mode_t **old_mode**

the old auth mode of AP

wifi_auth_mode_t **new_mode**

the new auth mode of AP

struct **wifi_event_sta_wps_er_pin_t**

Argument structure for WIFI_EVENT_STA_WPS_ER_PIN event

Public Members

uint8_t **pin_code**[8]
PIN code of station in enrollee mode

struct **wifi_event_sta_wps_er_success_t**
Argument structure for WIFI_EVENT_STA_WPS_ER_SUCCESS event

Public Members

uint8_t **ap_cred_cnt**
Number of AP credentials received

uint8_t **ssid**[MAX_SSID_LEN]
SSID of AP

uint8_t **passphrase**[MAX_PASSPHRASE_LEN]
Passphrase for the AP

struct *wifi_event_sta_wps_er_success_t*::[anonymous] **ap_cred**[MAX_WPS_AP_CRED]
All AP credentials received from WPS handshake

struct **wifi_event_ap_staconnected_t**
Argument structure for WIFI_EVENT_AP_STACONNECTED event

Public Members

uint8_t **mac**[6]
MAC address of the station connected to Soft-AP

uint8_t **aid**
the aid that soft-AP gives to the station connected to

bool **is_mesh_child**
flag to identify mesh child

struct **wifi_event_ap_stadisconnected_t**
Argument structure for WIFI_EVENT_AP_STADISCONNECTED event

Public Members

uint8_t **mac**[6]
MAC address of the station disconnects to soft-AP

uint8_t **aid**
the aid that soft-AP gave to the station disconnects to

bool **is_mesh_child**

flag to identify mesh child

struct **wifi_event_ap_probe_req_rx_t**

Argument structure for WIFI_EVENT_AP_PROBEREQRECVED event

Public Members

int **rssi**

Received probe request signal strength

uint8_t **mac**[6]

MAC address of the station which send probe request

struct **wifi_event_bss_rssi_low_t**

Argument structure for WIFI_EVENT_STA_BSS_RSSI_LOW event

Public Members

int32_t **rssi**

RSSI value of bss

struct **wifi_ftm_report_entry_t**

Argument structure for

Public Members

uint8_t **dlog_token**

Dialog Token of the FTM frame

int8_t **rssi**

RSSI of the FTM frame received

uint32_t **rtt**

Round Trip Time in pSec with a peer

uint64_t **t1**

Time of departure of FTM frame from FTM Responder in pSec

uint64_t **t2**

Time of arrival of FTM frame at FTM Initiator in pSec

uint64_t **t3**

Time of departure of ACK from FTM Initiator in pSec

uint64_t **t4**

Time of arrival of ACK at FTM Responder in pSec

struct **wifi_event_ftm_report_t**

Argument structure for WIFI_EVENT_FTM_REPORT event

Public Members

uint8_t **peer_mac**[6]

MAC address of the FTM Peer

wifi_ftm_status_t **status**

Status of the FTM operation

uint32_t **rtt_raw**

Raw average Round-Trip-Time with peer in Nano-Seconds

uint32_t **rtt_est**

Estimated Round-Trip-Time with peer in Nano-Seconds

uint32_t **dist_est**

Estimated one-way distance in Centi-Meters

wifi_ftm_report_entry_t ***ftm_report_data**

Pointer to FTM Report with multiple entries, should be freed after use

uint8_t **ftm_report_num_entries**

Number of entries in the FTM Report data

struct **wifi_event_action_tx_status_t**

Argument structure for WIFI_EVENT_ACTION_TX_STATUS event

Public Members

wifi_interface_t **ifx**

WiFi interface to send request to

uint32_t **context**

Context to identify the request

uint8_t **da**[6]

Destination MAC address

uint8_t **status**

Status of the operation

struct **wifi_event_roc_done_t**

Argument structure for WIFI_EVENT_ROC_DONE event

Public Members

uint32_t **context**

Context to identify the request

struct **wifi_event_ap_wps_rg_pin_t**

Argument structure for WIFI_EVENT_AP_WPS_RG_PIN event

Public Members

uint8_t **pin_code**[8]

PIN code of station in enrollee mode

struct **wifi_event_ap_wps_rg_fail_reason_t**

Argument structure for WIFI_EVENT_AP_WPS_RG_FAILED event

Public Members

wps_fail_reason_t **reason**

WPS failure reason wps_fail_reason_t

uint8_t **peer_macaddr**[6]

Enrollee mac address

struct **wifi_event_ap_wps_rg_success_t**

Argument structure for WIFI_EVENT_AP_WPS_RG_SUCCESS event

Public Members

uint8_t **peer_macaddr**[6]

Enrollee mac address

struct **wifi_event_nan_svc_match_t**

Argument structure for WIFI_EVENT_NAN_SVC_MATCH event

Public Members

uint8_t **subscribe_id**

Subscribe Service Identifier

uint8_t **publish_id**

Publish Service Identifier

uint8_t **pub_if_mac**[6]

NAN Interface MAC of the Publisher

struct **wifi_event_nan_replied_t**

Argument structure for WIFI_EVENT_NAN_REPLIED event

Public Members

uint8_t **publish_id**

Publish Service Identifier

uint8_t **subscribe_id**

Subscribe Service Identifier

uint8_t **sub_if_mac**[6]

NAN Interface MAC of the Subscriber

struct **wifi_event_nan_receive_t**

Argument structure for WIFI_EVENT_NAN_RECEIVE event

Public Members

uint8_t **inst_id**

Our Service Identifier

uint8_t **peer_inst_id**

Peer’s Service Identifier

uint8_t **peer_if_mac**[6]

Peer’s NAN Interface MAC

uint8_t **peer_svc_info**[ESP_WIFI_MAX_SVC_INFO_LEN]

Peer Service Info

struct **wifi_event_ndp_indication_t**

Argument structure for WIFI_EVENT_NDP_INDICATION event

Public Members

uint8_t **publish_id**

Publish Id for NAN Service

uint8_t **ndp_id**

NDP instance id

uint8_t **peer_nmi**[6]

Peer’s NAN Management Interface MAC

uint8_t **peer_ndi**[6]

Peer’ s NAN Data Interface MAC

uint8_t **svc_info**[ESP_WIFI_MAX_SVC_INFO_LEN]

Service Specific Info

struct **wifi_event_ndp_confirm_t**

Argument structure for WIFI_EVENT_NDP_CONFIRM event

Public Members

uint8_t **status**

NDP status code

uint8_t **ndp_id**

NDP instance id

uint8_t **peer_nmi**[6]

Peer’ s NAN Management Interface MAC

uint8_t **peer_ndi**[6]

Peer’ s NAN Data Interface MAC

uint8_t **own_ndi**[6]

Own NAN Data Interface MAC

uint8_t **svc_info**[ESP_WIFI_MAX_SVC_INFO_LEN]

Service Specific Info

struct **wifi_event_ndp_terminated_t**

Argument structure for WIFI_EVENT_NDP_TERMINATED event

Public Members

uint8_t **reason**

Termination reason code

uint8_t **ndp_id**

NDP instance id

uint8_t **init_ndi**[6]

Initiator’ s NAN Data Interface MAC

Macros

WIFI_OFFCHAN_TX_REQ

WIFI_OFFCHAN_TX_CANCEL

WIFI_ROC_REQ

WIFI_ROC_CANCEL

WIFI_PROTOCOL_11B

WIFI_PROTOCOL_11G

WIFI_PROTOCOL_11N

WIFI_PROTOCOL_LR

WIFI_PROTOCOL_11AX

SAE_H2E_IDENTIFIER_LEN

ESP_WIFI_MAX_CONN_NUM
max number of stations which can connect to ESP32C2 soft-AP

WIFI_VENDOR_IE_ELEMENT_ID

WIFI_PROMIS_FILTER_MASK_ALL
filter all packets

WIFI_PROMIS_FILTER_MASK_MGMT
filter the packets with type of WIFI_PKT_MGMT

WIFI_PROMIS_FILTER_MASK_CTRL
filter the packets with type of WIFI_PKT_CTRL

WIFI_PROMIS_FILTER_MASK_DATA
filter the packets with type of WIFI_PKT_DATA

WIFI_PROMIS_FILTER_MASK_MISC
filter the packets with type of WIFI_PKT_MISC

WIFI_PROMIS_FILTER_MASK_DATA_MPDU
filter the MPDU which is a kind of WIFI_PKT_DATA

WIFI_PROMIS_FILTER_MASK_DATA_AMPDU
filter the AMPDU which is a kind of WIFI_PKT_DATA

WIFI_PROMIS_FILTER_MASK_FCSFAIL
filter the FCS failed packets, do not open it in general

WIFI_PROMIS_CTRL_FILTER_MASK_ALL
filter all control packets

WIFI_PROMIS_CTRL_FILTER_MASK_WRAPPER
filter the control packets with subtype of Control Wrapper

WIFI_PROMIS_CTRL_FILTER_MASK_BAR
filter the control packets with subtype of Block Ack Request

WIFI_PROMIS_CTRL_FILTER_MASK_BA
filter the control packets with subtype of Block Ack

WIFI_PROMIS_CTRL_FILTER_MASK_PSPOLL
filter the control packets with subtype of PS-Poll

WIFI_PROMIS_CTRL_FILTER_MASK_RTS
filter the control packets with subtype of RTS

WIFI_PROMIS_CTRL_FILTER_MASK_CTS
filter the control packets with subtype of CTS

WIFI_PROMIS_CTRL_FILTER_MASK_ACK
filter the control packets with subtype of ACK

WIFI_PROMIS_CTRL_FILTER_MASK_CFEND
filter the control packets with subtype of CF-END

WIFI_PROMIS_CTRL_FILTER_MASK_CFENDACK
filter the control packets with subtype of CF-END+CF-ACK

WIFI_EVENT_MASK_ALL
mask all WiFi events

WIFI_EVENT_MASK_NONE
mask none of the WiFi events

WIFI_EVENT_MASK_AP_PROBEREQRECVED
mask SYSTEM_EVENT_AP_PROBEREQRECVED event

ESP_WIFI_NAN_MAX_SVC_SUPPORTED

ESP_WIFI_NAN_DATAPATH_MAX_PEERS

ESP_WIFI_NDP_ROLE_INITIATOR

ESP_WIFI_NDP_ROLE_RESPONDER

ESP_WIFI_MAX_SVC_NAME_LEN

ESP_WIFI_MAX_FILTER_LEN

ESP_WIFI_MAX_SVC_INFO_LEN

MAX_SSID_LEN

MAX_PASSPHRASE_LEN

MAX_WPS_AP_CRED

WIFI_STATIS_BUFFER

WIFI_STATIS_RXTX

WIFI_STATIS_HW

WIFI_STATIS_DIAG

WIFI_STATIS_PS

WIFI_STATIS_ALL

Type Definitions

typedef int (***wifi_action_rx_cb_t**)(uint8_t *hdr, uint8_t *payload, size_t len, uint8_t channel)

The Rx callback function of Action Tx operations.

Param hdr pointer to the IEEE 802.11 Header structure
Param payload pointer to the Payload following 802.11 Header
Param len length of the Payload
Param channel channel number the frame is received on

Enumerations

enum **wifi_mode_t**

Values:

enumerator **WIFI_MODE_NULL**

null mode

enumerator **WIFI_MODE_STA**

WiFi station mode

enumerator **WIFI_MODE_AP**

WiFi soft-AP mode

enumerator **WIFI_MODE_APSTA**

WiFi station + soft-AP mode

enumerator **WIFI_MODE_NAN**
WiFi NAN mode

enumerator **WIFI_MODE_MAX**

enum **wifi_interface_t**
Values:

enumerator **WIFI_IF_STA**

enumerator **WIFI_IF_AP**

enumerator **WIFI_IF_MAX**

enum **wifi_country_policy_t**
Values:

enumerator **WIFI_COUNTRY_POLICY_AUTO**
Country policy is auto, use the country info of AP to which the station is connected

enumerator **WIFI_COUNTRY_POLICY_MANUAL**
Country policy is manual, always use the configured country info

enum **wifi_auth_mode_t**
Values:

enumerator **WIFI_AUTH_OPEN**
authenticate mode : open

enumerator **WIFI_AUTH_WEP**
authenticate mode : WEP

enumerator **WIFI_AUTH_WPA_PSK**
authenticate mode : WPA_PSK

enumerator **WIFI_AUTH_WPA2_PSK**
authenticate mode : WPA2_PSK

enumerator **WIFI_AUTH_WPA_WPA2_PSK**
authenticate mode : WPA_WPA2_PSK

enumerator **WIFI_AUTH_WPA2_ENTERPRISE**
authenticate mode : WPA2_ENTERPRISE

enumerator **WIFI_AUTH_WPA3_PSK**
authenticate mode : WPA3_PSK

enumerator **WIFI_AUTH_WPA2_WPA3_PSK**
authenticate mode : WPA2_WPA3_PSK

enumerator **WIFI_AUTH_WAPI_PSK**

authenticate mode : WAPI_PSK

enumerator **WIFI_AUTH_OWE**

authenticate mode : OWE

enumerator **WIFI_AUTH_MAX**

enum **wifi_err_reason_t**

Values:

enumerator **WIFI_REASON_UNSPECIFIED**

enumerator **WIFI_REASON_AUTH_EXPIRE**

enumerator **WIFI_REASON_AUTH_LEAVE**

enumerator **WIFI_REASON_ASSOC_EXPIRE**

enumerator **WIFI_REASON_ASSOC_TOOMANY**

enumerator **WIFI_REASON_NOT_AUTHED**

enumerator **WIFI_REASON_NOT_ASSOCED**

enumerator **WIFI_REASON_ASSOC_LEAVE**

enumerator **WIFI_REASON_ASSOC_NOT_AUTHED**

enumerator **WIFI_REASON_DISASSOC_PWRCAP_BAD**

enumerator **WIFI_REASON_DISASSOC_SUPCHAN_BAD**

enumerator **WIFI_REASON_BSS_TRANSITION_DISASSOC**

enumerator **WIFI_REASON_IE_INVALID**

enumerator **WIFI_REASON_MIC_FAILURE**

enumerator **WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT**

enumerator **WIFI_REASON_GROUP_KEY_UPDATE_TIMEOUT**

enumerator **WIFI_REASON_IE_IN_4WAY_DIFFERS**

enumerator **WIFI_REASON_GROUP_CIPHER_INVALID**

enumerator **WIFI_REASON_PAIRWISE_CIPHER_INVALID**

enumerator **WIFI_REASON_AKMP_INVALID**

enumerator **WIFI_REASON_UNSUPP_RSN_IE_VERSION**

enumerator **WIFI_REASON_INVALID_RSN_IE_CAP**

enumerator **WIFI_REASON_802_1X_AUTH_FAILED**

enumerator **WIFI_REASON_CIPHER_SUITE_REJECTED**

enumerator **WIFI_REASON_TDLS_PEER_UNREACHABLE**

enumerator **WIFI_REASON_TDLS_UNSPECIFIED**

enumerator **WIFI_REASON_SSP_REQUESTED_DISASSOC**

enumerator **WIFI_REASON_NO_SSP_ROAMING_AGREEMENT**

enumerator **WIFI_REASON_BAD_CIPHER_OR_AKM**

enumerator **WIFI_REASON_NOT_AUTHORIZED_THIS_LOCATION**

enumerator **WIFI_REASON_SERVICE_CHANGE_PERCLUDES_TS**

enumerator **WIFI_REASON_UNSPECIFIED_QOS**

enumerator **WIFI_REASON_NOT_ENOUGH_BANDWIDTH**

enumerator **WIFI_REASON_MISSING_ACKS**

enumerator **WIFI_REASON_EXCEEDED_TXOP**

enumerator **WIFI_REASON_STA_LEAVING**

enumerator **WIFI_REASON_END_BA**

enumerator **WIFI_REASON_UNKNOWN_BA**

enumerator **WIFI_REASON_TIMEOUT**

enumerator **WIFI_REASON_PEER_INITIATED**

enumerator **WIFI_REASON_AP_INITIATED**

enumerator **WIFI_REASON_INVALID_FT_ACTION_FRAME_COUNT**

enumerator **WIFI_REASON_INVALID_PMKID**

enumerator **WIFI_REASON_INVALID_MDE**

enumerator **WIFI_REASON_INVALID_FTE**

enumerator **WIFI_REASON_TRANSMISSION_LINK_ESTABLISH_FAILED**

enumerator **WIFI_REASON_ALTERATIVE_CHANNEL_OCCUPIED**

enumerator **WIFI_REASON_BEACON_TIMEOUT**

enumerator **WIFI_REASON_NO_AP_FOUND**

enumerator **WIFI_REASON_AUTH_FAIL**

enumerator **WIFI_REASON_ASSOC_FAIL**

enumerator **WIFI_REASON_HANDSHAKE_TIMEOUT**

enumerator **WIFI_REASON_CONNECTION_FAIL**

enumerator **WIFI_REASON_AP_TSF_RESET**

enumerator **WIFI_REASON_ROAMING**

enumerator **WIFI_REASON_ASSOC_COMEBACK_TIME_TOO_LONG**

enumerator **WIFI_REASON_SA_QUERY_TIMEOUT**

enum **wifi_second_chan_t**

Values:

enumerator **WIFI_SECOND_CHAN_NONE**

the channel width is HT20

enumerator **WIFI_SECOND_CHAN_ABOVE**

the channel width is HT40 and the secondary channel is above the primary channel

enumerator **WIFI_SECOND_CHAN_BELOW**

the channel width is HT40 and the secondary channel is below the primary channel

enum **wifi_scan_type_t**

Values:

enumerator **WIFI_SCAN_TYPE_ACTIVE**

active scan

enumerator **WIFI_SCAN_TYPE_PASSIVE**

passive scan

enum **wifi_cipher_type_t**

Values:

enumerator **WIFI_CIPHER_TYPE_NONE**

the cipher type is none

enumerator **WIFI_CIPHER_TYPE_WEP40**

the cipher type is WEP40

enumerator **WIFI_CIPHER_TYPE_WEP104**

the cipher type is WEP104

enumerator **WIFI_CIPHER_TYPE_TKIP**

the cipher type is TKIP

enumerator **WIFI_CIPHER_TYPE_CCMP**

the cipher type is CCMP

enumerator **WIFI_CIPHER_TYPE_TKIP_CCMP**

the cipher type is TKIP and CCMP

enumerator **WIFI_CIPHER_TYPE_AES_CMAC128**

the cipher type is AES-CMAC-128

enumerator **WIFI_CIPHER_TYPE_SMS4**

the cipher type is SMS4

enumerator **WIFI_CIPHER_TYPE_GCMP**

the cipher type is GCMP

enumerator **WIFI_CIPHER_TYPE_GCMP256**

the cipher type is GCMP-256

enumerator **WIFI_CIPHER_TYPE_AES_GMAC128**

the cipher type is AES-GMAC-128

enumerator **WIFI_CIPHER_TYPE_AES_GMAC256**

the cipher type is AES-GMAC-256

enumerator **WIFI_CIPHER_TYPE_UNKNOWN**

the cipher type is unknown

enum **wifi_ant_t**

WiFi antenna.

Values:

enumerator **WIFI_ANT_ANT0**

WiFi antenna 0

enumerator **WIFI_ANT_ANT1**

WiFi antenna 1

enumerator **WIFI_ANT_MAX**

Invalid WiFi antenna

enum **wifi_scan_method_t**

Values:

enumerator **WIFI_FAST_SCAN**

Do fast scan, scan will end after find SSID match AP

enumerator **WIFI_ALL_CHANNEL_SCAN**

All channel scan, scan will end after scan all the channel

enum **wifi_sort_method_t**

Values:

enumerator **WIFI_CONNECT_AP_BY_SIGNAL**

Sort match AP in scan list by RSSI

enumerator **WIFI_CONNECT_AP_BY_SECURITY**

Sort match AP in scan list by security mode

enum **wifi_ps_type_t**

Values:

enumerator **WIFI_PS_NONE**

No power save

enumerator **WIFI_PS_MIN_MODEM**

Minimum modem power saving. In this mode, station wakes up to receive beacon every DTIM period

enumerator **WIFI_PS_MAX_MODEM**

Maximum modem power saving. In this mode, interval to receive beacons is determined by the listen_interval parameter in *wifi_sta_config_t*

enum **wifi_bandwidth_t**

Values:

enumerator **WIFI_BW_HT20**

enumerator **WIFI_BW_HT40**

enum **wifi_sae_pwe_method_t**

Configuration for SAE PWE derivation

Values:

enumerator **WPA3_SAE_PWE_UNSPECIFIED**

enumerator **WPA3_SAE_PWE_HUNT_AND_PECK**

enumerator **WPA3_SAE_PWE_HASH_TO_ELEMENT**

enumerator **WPA3_SAE_PWE_BOTH**

enum **wifi_sae_pk_mode_t**

Configuration for SAE-PK

Values:

enumerator **WPA3_SAE_PK_MODE_AUTOMATIC**

enumerator **WPA3_SAE_PK_MODE_ONLY**

enumerator **WPA3_SAE_PK_MODE_DISABLED**

enum **wifi_storage_t**

Values:

enumerator **WIFI_STORAGE_FLASH**

all configuration will store in both memory and flash

enumerator **WIFI_STORAGE_RAM**

all configuration will only store in the memory

enum **wifi_vendor_ie_type_t**

Vendor Information Element type.

Determines the frame type that the IE will be associated with.

Values:

enumerator **WIFI_VND_IE_TYPE_BEACON**

enumerator **WIFI_VND_IE_TYPE_PROBE_REQ**

enumerator **WIFI_VND_IE_TYPE_PROBE_RESP**

enumerator **WIFI_VND_IE_TYPE_ASSOC_REQ**

enumerator **WIFI_VND_IE_TYPE_ASSOC_RESP**

enum **wifi_vendor_ie_id_t**

Vendor Information Element index.

Each IE type can have up to two associated vendor ID elements.

Values:

enumerator **WIFI_VND_IE_ID_0**

enumerator **WIFI_VND_IE_ID_1**

enum **wifi_phy_mode_t**

Operation Phymode.

Values:

enumerator **WIFI_PHY_MODE_LR**

PHY mode for Low Rate

enumerator **WIFI_PHY_MODE_11B**

PHY mode for 11b

enumerator **WIFI_PHY_MODE_11G**

PHY mode for 11g

enumerator **WIFI_PHY_MODE_HT20**

PHY mode for Bandwidth HT20

enumerator **WIFI_PHY_MODE_HT40**

PHY mode for Bandwidth HT40

enumerator **WIFI_PHY_MODE_HE20**

PHY mode for Bandwidth HE20

enum **wifi_promiscuous_pkt_type_t**

Promiscuous frame type.

Passed to promiscuous mode RX callback to indicate the type of parameter in the buffer.

Values:

enumerator **WIFI_PKT_MGMT**

Management frame, indicates ‘buf’ argument is *wifi_promiscuous_pkt_t*

enumerator **WIFI_PKT_CTRL**

Control frame, indicates ‘buf’ argument is *wifi_promiscuous_pkt_t*

enumerator **WIFI_PKT_DATA**

Data frame, indiciates ‘buf’ argument is *wifi_promiscuous_pkt_t*

enumerator **WIFI_PKT_MISC**

Other type, such as MIMO etc. ‘buf’ argument is *wifi_promiscuous_pkt_t* but the payload is zero length.

enum **wifi_ant_mode_t**

WiFi antenna mode.

Values:

enumerator **WIFI_ANT_MODE_ANT0**

Enable WiFi antenna 0 only

enumerator **WIFI_ANT_MODE_ANT1**

Enable WiFi antenna 1 only

enumerator **WIFI_ANT_MODE_AUTO**

Enable WiFi antenna 0 and 1, automatically select an antenna

enumerator **WIFI_ANT_MODE_MAX**

Invalid WiFi enabled antenna

enum **wifi_nan_service_type_t**

NAN Services types.

Values:

enumerator **NAN_PUBLISH_SOLICITED**

Send unicast Publish frame to Subscribers that match the requirement

enumerator **NAN_PUBLISH_UNSOLICITED**

Send broadcast Publish frames in every Discovery Window(DW)

enumerator **NAN_SUBSCRIBE_ACTIVE**

Send broadcast Subscribe frames in every DW

enumerator **NAN_SUBSCRIBE_PASSIVE**

Passively listens to Publish frames

enum **wifi_phy_rate_t**

WiFi PHY rate encodings.

Values:

enumerator **WIFI_PHY_RATE_1M_L**

1 Mbps with long preamble

enumerator **WIFI_PHY_RATE_2M_L**

2 Mbps with long preamble

enumerator **WIFI_PHY_RATE_5M_L**

5.5 Mbps with long preamble

enumerator **WIFI_PHY_RATE_11M_L**

11 Mbps with long preamble

enumerator **WIFI_PHY_RATE_2M_S**

2 Mbps with short preamble

enumerator **WIFI_PHY_RATE_5M_S**

5.5 Mbps with short preamble

enumerator **WIFI_PHY_RATE_11M_S**

11 Mbps with short preamble

enumerator **WIFI_PHY_RATE_48M**

48 Mbps

enumerator **WIFI_PHY_RATE_24M**

24 Mbps

enumerator **WIFI_PHY_RATE_12M**

12 Mbps

enumerator **WIFI_PHY_RATE_6M**

6 Mbps

enumerator **WIFI_PHY_RATE_54M**

54 Mbps

enumerator **WIFI_PHY_RATE_36M**

36 Mbps

enumerator **WIFI_PHY_RATE_18M**

18 Mbps

enumerator **WIFI_PHY_RATE_9M**

9 Mbps rate table and guard interval information for each MCS rate

enumerator **WIFI_PHY_RATE_MCS0_LGI**

MCS0 with long GI

enumerator **WIFI_PHY_RATE_MCS1_LGI**

MCS1 with long GI

enumerator **WIFI_PHY_RATE_MCS2_LGI**

MCS2 with long GI

enumerator **WIFI_PHY_RATE_MCS3_LGI**

MCS3 with long GI

enumerator **WIFI_PHY_RATE_MCS4_LGI**

MCS4 with long GI

enumerator **WIFI_PHY_RATE_MCS5_LGI**

MCS5 with long GI

enumerator **WIFI_PHY_RATE_MCS6_LGI**

MCS6 with long GI

enumerator **WIFI_PHY_RATE_MCS7_LGI**

MCS7 with long GI

enumerator **WIFI_PHY_RATE_MCS0_SGI**

MCS0 with short GI

enumerator **WIFI_PHY_RATE_MCS1_SGI**

MCS1 with short GI

enumerator **WIFI_PHY_RATE_MCS2_SGI**

MCS2 with short GI

enumerator **WIFI_PHY_RATE_MCS3_SGI**

MCS3 with short GI

enumerator **WIFI_PHY_RATE_MCS4_SGI**

MCS4 with short GI

enumerator **WIFI_PHY_RATE_MCS5_SGI**

MCS5 with short GI

enumerator **WIFI_PHY_RATE_MCS6_SGI**

MCS6 with short GI

enumerator **WIFI_PHY_RATE_MCS7_SGI**

MCS7 with short GI

enumerator **WIFI_PHY_RATE_LORA_250K**

250 Kbps

enumerator **WIFI_PHY_RATE_LORA_500K**

500 Kbps

enumerator **WIFI_PHY_RATE_MAX**

enum **wifi_event_t**

WiFi event declarations

Values:

enumerator **WIFI_EVENT_WIFI_READY**
WiFi ready

enumerator **WIFI_EVENT_SCAN_DONE**
Finished scanning AP

enumerator **WIFI_EVENT_STA_START**
Station start

enumerator **WIFI_EVENT_STA_STOP**
Station stop

enumerator **WIFI_EVENT_STA_CONNECTED**
Station connected to AP

enumerator **WIFI_EVENT_STA_DISCONNECTED**
Station disconnected from AP

enumerator **WIFI_EVENT_STA_AUTHMODE_CHANGE**
the auth mode of AP connected by device’ s station changed

enumerator **WIFI_EVENT_STA_WPS_ER_SUCCESS**
Station wps succeeds in enrollee mode

enumerator **WIFI_EVENT_STA_WPS_ER_FAILED**
Station wps fails in enrollee mode

enumerator **WIFI_EVENT_STA_WPS_ER_TIMEOUT**
Station wps timeout in enrollee mode

enumerator **WIFI_EVENT_STA_WPS_ER_PIN**
Station wps pin code in enrollee mode

enumerator **WIFI_EVENT_STA_WPS_ER_PBC_OVERLAP**
Station wps overlap in enrollee mode

enumerator **WIFI_EVENT_AP_START**
Soft-AP start

enumerator **WIFI_EVENT_AP_STOP**
Soft-AP stop

enumerator **WIFI_EVENT_AP_STACONNECTED**
a station connected to Soft-AP

enumerator **WIFI_EVENT_AP_STADISCONNECTED**
a station disconnected from Soft-AP

enumerator **WIFI_EVENT_AP_PROBEREQRECVED**

Receive probe request packet in soft-AP interface

enumerator **WIFI_EVENT_FTM_REPORT**

Receive report of FTM procedure

enumerator **WIFI_EVENT_STA_BSS_RSSI_LOW**

AP’ s RSSI crossed configured threshold

enumerator **WIFI_EVENT_ACTION_TX_STATUS**

Status indication of Action Tx operation

enumerator **WIFI_EVENT_ROC_DONE**

Remain-on-Channel operation complete

enumerator **WIFI_EVENT_STA_BEACON_TIMEOUT**

Station beacon timeout

enumerator **WIFI_EVENT_CONNECTIONLESS_MODULE_WAKE_INTERVAL_START**

Connectionless module wake interval start

enumerator **WIFI_EVENT_AP_WPS_RG_SUCCESS**

Soft-AP wps succeeds in registrar mode

enumerator **WIFI_EVENT_AP_WPS_RG_FAILED**

Soft-AP wps fails in registrar mode

enumerator **WIFI_EVENT_AP_WPS_RG_TIMEOUT**

Soft-AP wps timeout in registrar mode

enumerator **WIFI_EVENT_AP_WPS_RG_PIN**

Soft-AP wps pin code in registrar mode

enumerator **WIFI_EVENT_AP_WPS_RG_PBC_OVERLAP**

Soft-AP wps overlap in registrar mode

enumerator **WIFI_EVENT_ITWT_SETUP**

iTWT setup

enumerator **WIFI_EVENT_ITWT_TEARDOWN**

iTWT teardown

enumerator **WIFI_EVENT_ITWT_PROBE**

iTWT probe

enumerator **WIFI_EVENT_ITWT_SUSPEND**

iTWT suspend

enumerator **WIFI_EVENT_NAN_STARTED**

NAN Discovery has started

enumerator **WIFI_EVENT_NAN_STOPPED**

NAN Discovery has stopped

enumerator **WIFI_EVENT_NAN_SVC_MATCH**

NAN Service Discovery match found

enumerator **WIFI_EVENT_NAN_REPLIED**

Replied to a NAN peer with Service Discovery match

enumerator **WIFI_EVENT_NAN_RECEIVE**

Received a Follow-up message

enumerator **WIFI_EVENT_NDP_INDICATION**

Received NDP Request from a NAN Peer

enumerator **WIFI_EVENT_NDP_CONFIRM**

NDP Confirm Indication

enumerator **WIFI_EVENT_NDP_TERMINATED**

NAN Datapath terminated indication

enumerator **WIFI_EVENT_MAX**

Invalid WiFi event ID

enum **wifi_event_sta_wps_fail_reason_t**

Argument structure for WIFI_EVENT_STA_WPS_ER_FAILED event

Values:

enumerator **WPS_FAIL_REASON_NORMAL**

WPS normal fail reason

enumerator **WPS_FAIL_REASON_RECV_M2D**

WPS receive M2D frame

enumerator **WPS_FAIL_REASON_MAX**

enum **wifi_ftm_status_t**

FTM operation status types.

Values:

enumerator **FTM_STATUS_SUCCESS**

FTM exchange is successful

enumerator **FTM_STATUS_UNSUPPORTED**

Peer does not support FTM

enumerator **FTM_STATUS_CONF_REJECTED**

Peer rejected FTM configuration in FTM Request

enumerator **FTM_STATUS_NO_RESPONSE**

Peer did not respond to FTM Requests

enumerator **FTM_STATUS_FAIL**

Unknown error during FTM exchange

enum **wps_fail_reason_t**

Values:

enumerator **WPS_AP_FAIL_REASON_NORMAL**

WPS normal fail reason

enumerator **WPS_AP_FAIL_REASON_CONFIG**

WPS failed due to incorrect config

enumerator **WPS_AP_FAIL_REASON_AUTH**

WPS failed during auth

enumerator **WPS_AP_FAIL_REASON_MAX**

Wi-Fi Easy Connect™ (DPP)

Wi-Fi Easy Connect™, also known as Device Provisioning Protocol (DPP) or Easy Connect, is a provisioning protocol certified by Wi-Fi Alliance. It is a secure and standardized provisioning protocol for configuration of Wi-Fi Devices. With Easy Connect adding a new device to a network is as simple as scanning a QR Code. This reduces complexity and enhances user experience while onboarding devices without UI like Smart Home and IoT products. Unlike old protocols like WiFi Protected Setup (WPS), Wi-Fi Easy Connect incorporates strong encryption through public key cryptography to ensure networks remain secure as new devices are added. Easy Connect brings many benefits in the User Experience:

- Simple and intuitive to use; no lengthy instructions to follow for new device setup
- No need to remember and enter passwords into the device being provisioned
- Works with electronic or printed QR codes, or human-readable strings
- Supports both WPA2 and WPA3 networks

Please refer to Wi-Fi Alliance ' s official page on Easy Connect for more information.

ESP32-C2 supports Enrollee mode of Easy Connect with QR Code as the provisioning method. A display is required to display this QR Code. Users can scan this QR Code using their capable device and provision the ESP32-C2 to their Wi-Fi network. The provisioning device needs to be connected to the AP which need not support Wi-Fi Easy Connect™. Easy Connect is still an evolving protocol. Of known platforms that support the QR Code method are some Android smartphones with Android 10 or higher. To use Easy Connect no additional App needs to be installed on the supported smartphone.

Application Example Example on how to provision ESP32-C2 using a supported smartphone: wifi/wifi_easy_connect/dpp-enrollee.

API Reference

Header File

- components/wpa_supplicant/esp_supplicant/include/esp_dpp.h

Functions

esp_err_t **esp_supp_dpp_init**(*esp_supp_dpp_event_cb_t* evt_cb)

Initialize DPP Supplicant.

```
Starts DPP Supplicant and initializes related Data Structures.
```

return

- ESP_OK: Success
- ESP_FAIL: Failure

参数 **evt_cb** –Callback function to receive DPP related events

void **esp_supp_dpp_deinit**(void)

De-initalize DPP Supplicant.

```
Frees memory from DPP Supplicant Data Structures.
```

esp_err_t **esp_supp_dpp_bootstrap_gen**(const char *chan_list, *esp_supp_dpp_bootstrap_t* type, const char *key, const char *info)

Generates Bootstrap Information as an Enrollee.

```
Generates Out Of Band Bootstrap information as an Enrollee which can be
used by a DPP Configurator to provision the Enrollee.
```

参数

- **chan_list** –List of channels device will be available on for listening
- **type** –Bootstrap method type, only QR Code method is supported for now.
- **key** –(Optional) 32 byte Raw Private Key for generating a Bootstrapping Public Key
- **info** –(Optional) Ancilliary Device Information like Serial Number

返回

- ESP_OK: Success
- ESP_FAIL: Failure

esp_err_t **esp_supp_dpp_start_listen**(void)

Start listening on Channels provided during esp_supp_dpp_bootstrap_gen.

```
Listens on every Channel from Channel List for a pre-defined wait time.
```

返回

- ESP_OK: Success
- ESP_FAIL: Generic Failure
- ESP_ERR_INVALID_STATE: ROC attempted before WiFi is started
- ESP_ERR_NO_MEM: Memory allocation failed while posting ROC request

void **esp_supp_dpp_stop_listen**(void)

Stop listening on Channels.

```
Stops listening on Channels and cancels ongoing listen operation.
```

Macros

ESP_ERR_DPP_FAILURE

Generic failure during DPP Operation

ESP_ERR_DPP_TX_FAILURE

DPP Frame Tx failed OR not Acked

ESP_ERR_DPP_INVALID_ATTR

Encountered invalid DPP Attribute

Type Definitions

typedef enum *dpp_bootstrap_type* **esp_supp_dpp_bootstrap_t**

Types of Bootstrap Methods for DPP.

typedef void (***esp_supp_dpp_event_cb_t**)(*esp_supp_dpp_event_t* evt, void *data)

Callback function for receiving DPP Events from Supplicant.

```
Callback function will be called with DPP related information.
```

Param evt DPP event ID
Param data Event data payload

Enumerations

enum **dpp_bootstrap_type**

Types of Bootstrap Methods for DPP.

Values:

enumerator **DPP_BOOTSTRAP_QR_CODE**

QR Code Method

enumerator **DPP_BOOTSTRAP_PKEX**

Proof of Knowledge Method

enumerator **DPP_BOOTSTRAP_NFC_URI**

NFC URI record Method

enum **esp_supp_dpp_event_t**

Types of Callback Events received from DPP Supplicant.

Values:

enumerator **ESP_SUPP_DPP_URI_READY**

URI is ready through Bootstrapping

enumerator **ESP_SUPP_DPP_CFG_RECVD**

Config received via DPP Authentication

enumerator **ESP_SUPP_DPP_FAIL**

DPP Authentication failure

Wi-Fi Aware™ (NAN)

Wi-Fi Aware™ or NAN (Neighbor Awareness Networking) is a protocol that allows Wi-Fi devices to discover services in their proximity. Typically, location-based services are based on querying servers for information about the environment and the location knowledge is based on GPS or other location reckoning techniques. However NAN does not require real-time connection to servers, GPS or other geo-location, but instead uses direct device-to-device Wi-Fi to discover and exchange information. NAN scales effectively in dense Wi-Fi environments and complements the connectivity of Wi-Fi by providing information about people and services in the proximity.

Multiple NAN devices which are in the vicinity will form a NAN cluster which allows them to communicate with each other. Devices within a NAN cluster can advertise (Publish method) or look for (Subscribe method) services using NAN Service Discovery protocols. Matching of services is done by service name, once a match is found a device can either send a message or establish an IPv6 datapath with the peer.

ESP32-C2 supports Wi-Fi Aware in standalone mode with support for both Service Discovery and Datapath. Wi-Fi Aware is still an evolving protocol. Please refer to Wi-Fi Alliance's official page on Wi-Fi Aware for more information. Many Android smartphones with Android 8 or higher support Wi-Fi Aware. Refer to Android's developer guide on Wi-Fi Aware Wi-Fi Aware for more information.

Application Example A pair of examples for a Publisher-Subscriber use case: wifi/wifi_aware/nan_publisher and wifi/wifi_aware/nan_subscriber. A user interactive console example to explore full functionality of Wi-Fi Aware: wifi/wifi_aware/nan_console. Please check the *README* for more details in respective example directories.

API Reference

Header File

- components/esp_wifi/wifi_apps/include/esp_nan.h

Functions

esp_err_t **esp_wifi_nan_start**(const *wifi_nan_config_t* *nan_cfg)

Start NAN Discovery with provided configuration.

Attention This API should be called after esp_wifi_init().

参数 **nan_cfg** –NAN related parameters to be configured.

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_nan_stop**(void)

Stop NAN Discovery, end NAN Services and Datapaths.

返回

- ESP_OK: succeed
- others: failed

uint8_t **esp_wifi_nan_publish_service**(const *wifi_nan_publish_cfg_t* *publish_cfg, bool ndp_resp_needed)

Start Publishing a service to the NAN Peers in vicinity.

Attention This API should be called after esp_wifi_nan_start().

参数

- **publish_cfg** –Configuration parameters for publishing a service.
- **ndp_resp_needed** –Setting this true will require user response for every NDP Req using esp_wifi_nan_datapath_resp API.

返回

- non-zero: Publish service identifier
- zero: failed

uint8_t **esp_wifi_nan_subscribe_service**(const *wifi_nan_subscribe_cfg_t* *subscribe_cfg)

Subscribe for a service within the NAN cluster.

Attention This API should be called after esp_wifi_nan_start().

参数 **subscribe_cfg** –Configuration parameters for subscribing for a service.

返回

- non-zero: Subscribe service identifier
- zero: failed

esp_err_t **esp_wifi_nan_send_message**(*wifi_nan_followup_params_t* *fup_params)

Send a follow-up message to the NAN Peer with matched service.

Attention This API should be called after a NAN service is discovered due to a match.

参数 **fup_params** –Configuration parameters for sending a Follow-up message.

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_nan_cancel_service**(uint8_t service_id)

Cancel a NAN service.

参数 **service_id** –Publish/Subscribe service id to be cancelled.

返回

- ESP_OK: succeed
- others: failed

uint8_t **esp_wifi_nan_datapath_req**(*wifi_nan_datapath_req_t* *req)

Send NAN Datapath Request to a NAN Publisher with matched service.

Attention This API should be called by the Subscriber after a match occurs with a Publisher.

参数 **req** –NAN Datapath Request parameters.

返回

- non-zero NAN Datapath identifier: If NAN datapath req was accepted by publisher
- zero: If NAN datapath req was rejected by publisher or a timeout occurs

esp_err_t **esp_wifi_nan_datapath_resp**(*wifi_nan_datapath_resp_t* *resp)

Respond to a NAN Datapath request with Accept or Reject.

Attention This API should be called if ndp_auto_accept is not set True by the Publisher and a WIFI_EVENT_NDP_INDICATION event is received due to an incoming NDP request.

参数 **resp** –NAN Datapath Response parameters.

返回

- ESP_OK: succeed
- others: failed

esp_err_t **esp_wifi_nan_datapath_end**(*wifi_nan_datapath_end_req_t* *req)

Terminate a NAN Datapath.

参数 **req** –NAN Datapath end request parameters.

返回

- ESP_OK: succeed
- others: failed

void **esp_wifi_nan_get_ipv6_linklocal_from_mac**(ip6_addr_t *ip6, uint8_t *mac_addr)

Get IPv6 Link Local address using MAC address.

参数

- **ip6** –**[out]** Derived IPv6 Link Local address.
- **mac_addr** –**[in]** Input MAC Address.

esp_err_t **esp_wifi_nan_get_own_svc_info**(uint8_t *own_svc_id, char *svc_name, int *num_peer_records)

brief Get own Service information from Service ID OR Name.

参数

- **own_svc_id** –**[inout]** As input, it indicates Service ID to search for. As output, it indicates Service ID of the service found using Service Name.
- **svc_name** –**[inout]** As input, it indicates Service Name to search for. As output, it indicates Service Name of the service found using Service ID.
- **num_peer_records** –**[out]** Number of peers discovered by corresponding service.

返回

- ESP_OK: succeed
- ESP_FAIL: failed

esp_err_t **esp_wifi_nan_get_peer_records**(int *num_peer_records, uint8_t own_svc_id, struct *nan_peer_record* *peer_record)

brief Get a list of Peers discovered by the given Service.

参数

- **num_peer_records** –**[inout]** As input param, it stores max peers peer_record can hold. As output param, it specifies the actual number of peers this API returns.
- **own_svc_id** –Service ID of own service.
- **peer_record** –**[out]** Pointer to first peer record.

返回

- ESP_OK: succeed
- ESP_FAIL: failed

esp_err_t **esp_wifi_nan_get_peer_info**(char *svc_name, uint8_t *peer_mac, struct *nan_peer_record* *peer_info)

brief Find Peer’s Service information using Peer MAC and optionally Service Name.

参数

- **svc_name** –Service Name of the published/subscribed service.
- **peer_mac** –Peer’s NAN Management Interface MAC address.

- **peer_info** –**[out]** Peer’s service information structure.

返回

- ESP_OK: succeed
- ESP_FAIL: failed

Structures

struct **nan_peer_record**

Parameters of a peer service record

Public Members

uint8_t **peer_svc_id**

Identifier of Peer’s service

uint8_t **own_svc_id**

Identifier of own service associated with Peer

uint8_t **peer_nmi**[6]

Peer’s NAN Management Interface address

uint8_t **peer_svc_type**

Peer’s service type (Publish/Subscribe)

uint8_t **ndp_id**

Specifies if the peer has any active datapath

uint8_t **peer_ndi**[6]

Peer’s NAN Data Interface address, only valid when ndp_id is non-zero

Macros

WIFI_NAN_CONFIG_DEFAULT()

NDP_STATUS_ACCEPTED

NDP_STATUS_REJECTED

NAN_MAX_PEERS_RECORD

ESP_NAN_PUBLISH

ESP_NAN_SUBSCRIBE

本部分的 Wi-Fi API 示例代码存放在 ESP-IDF 示例项目的 wifi 目录下。

2.5.2 以太网

以太网

概述 ESP-IDF 提供一系列功能强大且兼具一致性的 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率应用。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：

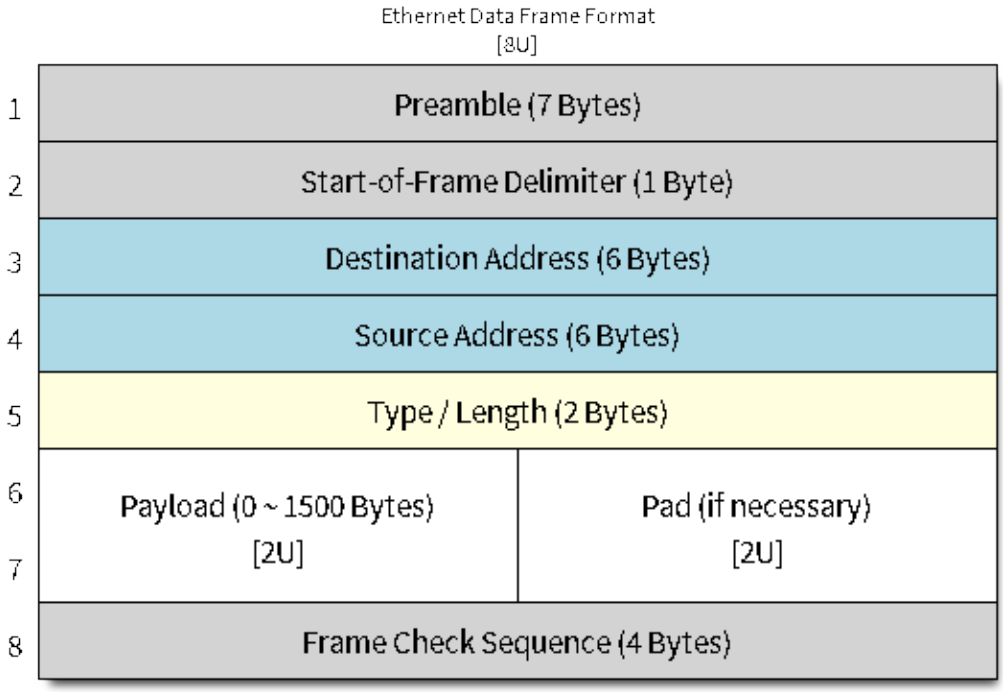


图 2: 以太网数据帧格式

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。

根据您的以太网板设计，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 MAC 驱动应在禁用缓存后开始工作，那么则需要用*ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE* 配置这个字段。

MAC 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。以太网驱动程序会自动与对等的以太网节点进行协商，以确定双工和速度模式。此值通常取决于您电路板上 PHY 设备的性能。
- *eth_phy_config_t::reset_gpio_num*：如果您的开发板同时将 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。

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);
```

（下页继续）

(续上页)

```
dm9051_config.int_gpio_num = CONFIG_EXAMPLE_ETH_SPI_INT_GPIO;
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。您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*ESP Event*.

```
/** 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:
```

(下页继续)

(续上页)

```
        ESP_LOGI(TAG, "Ethernet Stopped");
        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。只有当链路的两边都支持暂停帧时，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”：ethernet/eth2ap
- 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 目前市面上已有多家 PHY 制造商提供了大量的芯片组合。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/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，您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/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/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
   - 可选包含支持非 IEEE 802.3 或自定义功能所需的额外变量。示例请参见 esp_eth/src/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 IDF 组件管理中心 将驱动分享给其他用户。

API 参考

Header File

- components/esp_eth/include/esp_eth.h

Header File

- components/esp_eth/include/esp_eth_driver.h

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 successfull
- 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 Etherent 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_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

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_CUSTOM_MAC_CMDS**

enumerator **ETH_CMD_CUSTOM_PHY_CMDS**

Header File

- components/esp_eth/include/esp_eth_com.h

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

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

Unions

union **eth_mac_clock_config_t**

#include <esp_eth_mac.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

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, uint32_t 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.

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.

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 APLL Clock. EMAC Clock GPIO number needs to be configured when this option is selected.

enum **emac_rmii_clock_gpio_t**

RMII Clock GPIO number Options.

Values:

enumerator **EMAC_CLK_IN_GPIO**

MAC will get RMII clock from outside at this GPIO.

备注： ESP32 only supports GPIO0 to input the RMII clock.

---

enumerator **EMAC_APPL_CLK_OUT_GPIO**

Output RMII Clock from internal APLL Clock available at GPIO0.

备注： GPIO0 can be set to output a pre-divided PLL clock (test only!). Enabling this option will configure GPIO0 to output a 50MHz clock. In fact this clock doesn't have directly relationship with EMAC peripheral. Sometimes this clock won't work well with your PHY chip. You might need to add some extra devices after GPIO0 (e.g. inverter). Note that outputting RMII clock on GPIO0 is an experimental practice. If you want the Ethernet to work with WiFi, don't select GPIO0 output mode for stability.

---

enumerator **EMAC_CLK_OUT_GPIO**

Output RMII Clock from internal APLL Clock available at GPIO16.

enumerator **EMAC_CLK_OUT_180_GPIO**

Inverted Output RMII Clock from internal APLL Clock available at GPIO17.

Header File

- components/esp_eth/include/esp_eth_phy.h

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 (*`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, uint32_t 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 controll 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

Functions

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

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

Header File

- components/esp_eth/include/esp_eth_netif_glue.h

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。

2.5.3 Thread

Thread

Introduction Thread is a IP-based mesh networking protocol. It's based on the 802.15.4 physical and MAC layer.

Application Examples The openthread directory of ESP-IDF examples contains the following applications:

- The OpenThread interactive shell openthread/ot_cli.
- The Thread border router openthread/ot_br.
- The Thread radio co-processor openthread/ot_rcp.

API Reference For manipulating the Thread network, the OpenThread api shall be used. The OpenThread api docs can be found at the OpenThread official website.

ESP-IDF provides extra apis for launching and managing the OpenThread stack, binding to network interfaces and border routing features.

Header File

- components/openthread/include/esp_openthread.h

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_launch_mainloop**(void)

Launches the OpenThread main loop.

备注: Thie 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

Structures

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

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)

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_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 >

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)

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_RCP_UART**

RCP UART connection to the host

enumerator **HOST_CONNECTION_MODE_RCP_SPI**

RCP SPI connection to the host

Header File

- components/openthread/include/esp_openthread_lock.h

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 OT APIs that takes an otInstance argument MUST be protected with this API lock except that the call site is in OT callbacks.

参数 **block_ticks** –**[in]** The maxinum 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 maxinum 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.

Header File

- components/openthread/include/esp_openthread_netif_glue.h

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.

Macros

ESP_NETIF_INHERENT_DEFAULT_OPENTHREAD()

Default configuration reference of OT esp-netif.

ESP_NETIF_DEFAULT_OPENTHREAD()

Header File

- components/openthread/include/esp_openthread_border_router.h

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_rcp_deinit**(void)

Deinitializes the conneciton to 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。

2.5.4 IP 网络层协议

ESP-NETIF

The purpose of ESP-NETIF library is twofold:

- It provides an abstraction layer for the application on top of the TCP/IP stack. This will allow applications to choose between IP stacks in the future.
- The APIs it provides are thread safe, even if the underlying TCP/IP stack APIs are not.

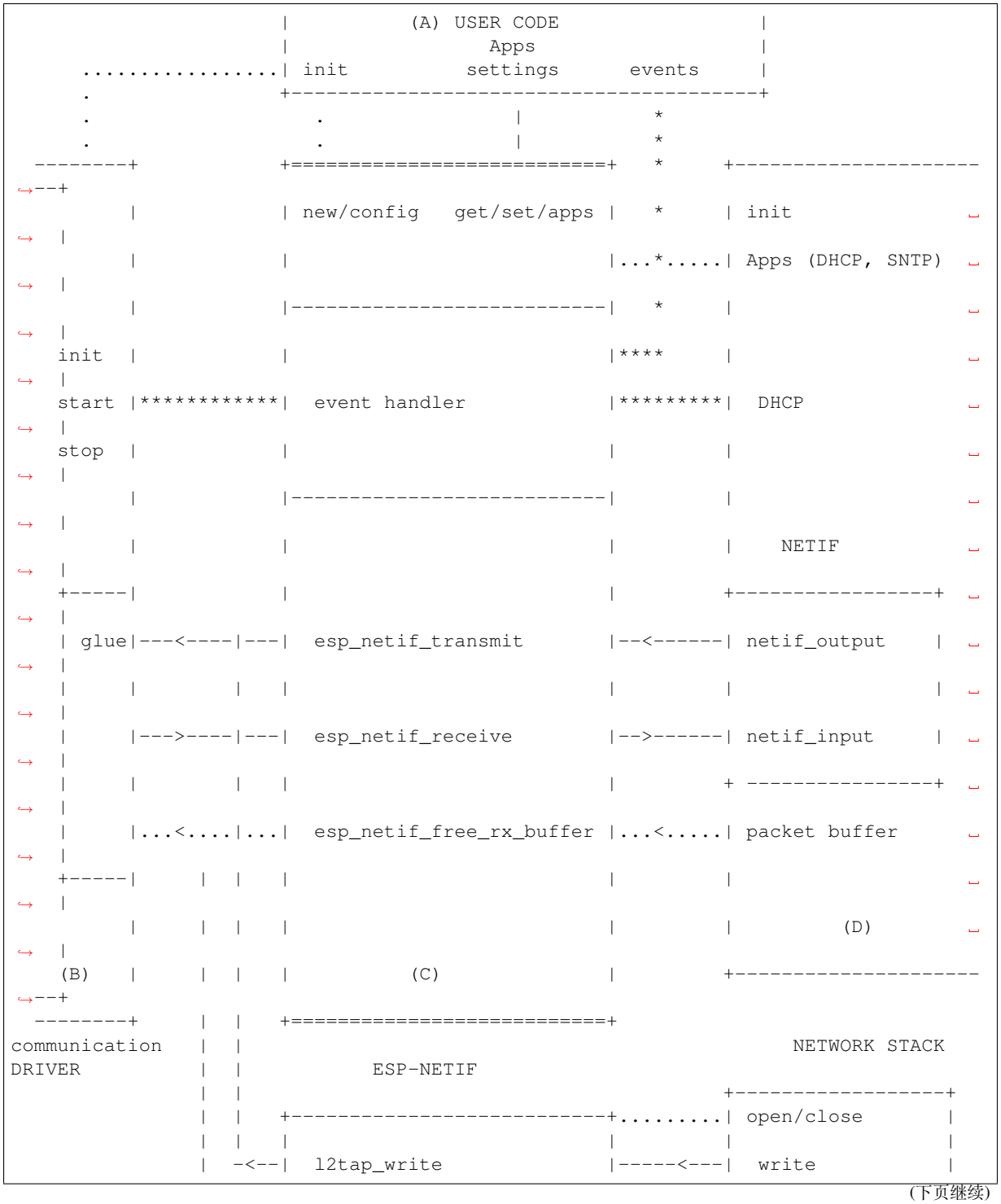
ESP-IDF currently implements ESP-NETIF for the lwIP TCP/IP stack only. However, the adapter itself is TCP/IP implementation agnostic and different implementations are possible.

It is also possible to use a custom TCP/IP stack with ESP-IDF, provided it implements BSD API. For more information on building ESP-IDF without lwIP, please refer to components/esp_netif_stack/README.md.

Some ESP-NETIF API functions are intended to be called by application code, for example to get/set interface IP addresses, configure DHCP. Other functions are intended for internal ESP-IDF use by the network driver layer.

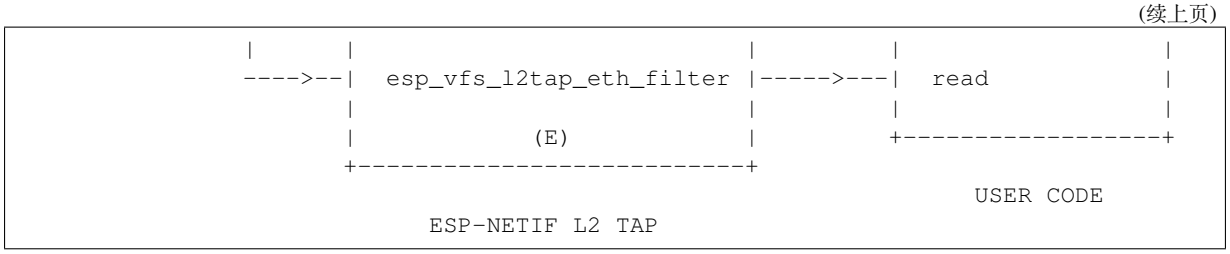
In many cases, applications do not need to call ESP-NETIF APIs directly as they are called from the default network event handlers.

ESP-NETIF architecture



(下页继续)

(续上页)



Data and event flow in the diagram

- `. . . . . .` Initialization line from user code to ESP-NETIF and communication driver
- `--<--->--` Data packets going from communication media to TCP/IP stack and back
- `********` Events aggregated in ESP-NETIF propagates to driver, user code and network stack
- `|` User settings and runtime configuration

ESP-NETIF interaction

A) User code, boiler plate Overall application interaction with a specific IO driver for communication media and configured TCP/IP network stack is abstracted using ESP-NETIF APIs and outlined as below:

A) Initialization code

1) Initializes IO driver
2) Creates a new instance of ESP-NETIF and configure with
   - ESP-NETIF specific options (flags, behaviour, name)
   - Network stack options (netif init and input functions, not publicly available)
   - IO driver specific options (transmit, free rx buffer functions, IO driver handle)
3) Attaches the IO driver handle to the ESP-NETIF instance created in the above steps
4) Configures event handlers
   - use default handlers for common interfaces defined in IO drivers; or define a specific handlers for customised behaviour/new interfaces
   - register handlers for app related events (such as IP lost/acquired)

B) Interaction with network interfaces using ESP-NETIF API

- Getting and setting TCP/IP related parameters (DHCP, IP, etc)
- Receiving IP events (connect/disconnect)
- Controlling application lifecycle (set interface up/down)

B) Communication driver, IO driver, media driver Communication driver plays these two important roles in relation with ESP-NETIF:

1) Event handlers: Define behaviour patterns of interaction with ESP-NETIF (for example: ethernet link-up -> turn netif on)
2) Glue IO layer: Adapts the input/output functions to use ESP-NETIF transmit, receive and free receive buffer
   - Installs driver_transmit to appropriate ESP-NETIF object, so that outgoing packets from network stack are passed to the IO driver
   - Calls *`esp_netif_receive()`* to pass incoming data to network stack

C) ESP-NETIF ESP-NETIF is an intermediary between an IO driver and a network stack, connecting packet data path between these two. As that it provides a set of interfaces for attaching a driver to ESP-NETIF object (runtime) and configuring a network stack (compile time). In addition to that a set of API is provided to control network interface lifecycle and its TCP/IP properties. As an overview, the ESP-NETIF public interface could be divided into these 6 groups:

1) Initialization APIs (to create and configure ESP-NETIF instance)
2) Input/Output API (for passing data between IO driver and network stack)
3) Event or Action API

- Used for network interface lifecycle management
- ESP-NETIF provides building blocks for designing event handlers

4) Setters and Getters for basic network interface properties
5) Network stack abstraction: enabling user interaction with TCP/IP stack

- Set interface up or down
- DHCP server and client API
- DNS API
- *SNTP API*

6) Driver conversion utilities

D) Network stack Network stack has no public interaction with application code with regard to public interfaces and shall be fully abstracted by ESP-NETIF API.

E) ESP-NETIF L2 TAP Interface The ESP-NETIF L2 TAP interface is ESP-IDF mechanism utilized to access Data Link Layer (L2 per OSI/ISO) for frame reception and transmission from user application. Its typical usage in embedded world might be implementation of non-IP related protocols such as PTP, Wake on LAN and others. Note that only Ethernet (IEEE 802.3) is currently supported.

From user perspective, the ESP-NETIF L2 TAP interface is accessed using file descriptors of VFS which provides a file-like interfacing (using functions like `open()`, `read()`, `write()`, etc). Refer to 虚拟文件系统组件 to learn more.

There is only one ESP-NETIF L2 TAP interface device (path name) available. However multiple file descriptors with different configuration can be opened at a time since the ESP-NETIF L2 TAP interface can be understood as generic entry point to Layer 2 infrastructure. Important is then specific configuration of particular file descriptor. It can be configured to give an access to specific Network Interface identified by `if_key` (e.g. *ETH_DEF*) and to filter only specific frames based on their type (e.g. Ethernet type in case of IEEE 802.3). Filtering only specific frames is crucial since the ESP-NETIF L2 TAP needs to exist along with IP stack and so the IP related traffic (IP, ARP, etc.) should not be passed directly to the user application. Even though such option is still configurable, it is not recommended in standard use cases. Filtering is also advantageous from a perspective the user's application gets access only to frame types it is interested in and the remaining traffic is either passed to other L2 TAP file descriptors or to IP stack.

ESP-NETIF L2 TAP Interface Usage Manual

Initialization To be able to use the ESP-NETIF L2 TAP interface, it needs to be enabled in Kconfig by *CONFIG_ESP_NETIF_L2_TAP* first and then registered by *`esp_vfs_l2tap_intf_register()`* prior usage of any VFS function.

open() Once the ESP-NETIF L2 TAP is registered, it can be opened at path name "/dev/net/tap". The same path name can be opened multiple times up to *CONFIG_ESP_NETIF_L2_TAP_MAX_FDS* and multiple file descriptors with with different configuration may access the Data Link Layer frames.

The ESP-NETIF L2 TAP can be opened with `O_NONBLOCK` file status flag to the `read()` does not block. Note that the `write()` may block in current implementation when accessing a Network interface since it is a shared resource among multiple ESP-NETIF L2 TAP file descriptors and IP stack, and there is currently no queuing mechanism deployed. The file status flag can be retrieved and modified using `fcntl()`.

On success, `open()` returns the new file descriptor (a nonnegative integer). On error, -1 is returned and `errno` is set to indicate the error.

ioctl() The newly opened ESP-NETIF L2 TAP file descriptor needs to be configured prior its usage since it is not bounded to any specific Network Interface and no frame type filter is configured. The following configuration options are available to do so:

- `L2TAP_S_INTF_DEVICE` - bounds the file descriptor to specific Network Interface which is identified by its `if_key`. ESP-NETIF Network Interface `if_key` is passed to `ioctl()` as the third parameter. Note that default Network Interfaces `if_key`'s used in ESP-IDF can be found in esp_netif/include/esp_netif_defaults.h.
- `L2TAP_S_DEVICE_DRV_HNDL` - is other way how to bound the file descriptor to specific Network Interface. In this case the Network interface is identified directly by IO Driver handle (e.g. *`esp_eth_handle_t`* in case of Ethernet). The IO Driver handle is passed to `ioctl()` as the third parameter.
- `L2TAP_S_RCV_FILTER` - sets the filter to frames with this type to be passed to the file descriptor. In case of Ethernet frames, the frames are to be filtered based on Length/Ethernet type field. In case the filter value is set less than or equal to 0x05DC, the Ethernet type field is considered to represent IEEE802.3 Length Field and all frames with values in interval <0, 0x05DC> at that field are to be passed to the file descriptor. The IEEE802.2 logical link control (LLC) resolution is then expected to be performed by user's application. In case the filter value is set greater than 0x05DC, the Ethernet type field is considered to represent protocol identification and only frames which are equal to the set value are to be passed to the file descriptor.

All above set configuration options have getter counterpart option to read the current settings.

警 告: The file descriptor needs to be firstly bounded to specific Network Interface by `L2TAP_S_INTF_DEVICE` or `L2TAP_S_DEVICE_DRV_HNDL` to be `L2TAP_S_RCV_FILTER` option available.

备注: VLAN tagged frames are currently not recognized. If user needs to process VLAN tagged frames, they need set filter to be equal to VLAN tag (i.e. 0x8100 or 0x88A8) and process the VLAN tagged frames in user application.

备注: `L2TAP_S_DEVICE_DRV_HNDL` is particularly useful when user's application does not require usage of IP stack and so ESP-NETIF is not required to be initialized too. As a result, Network Interface cannot be identified by its `if_key` and hence it needs to be identified directly by its IO Driver handle.

On success, `ioctl()` returns 0. On error, -1 is returned, and `errno` is set to indicate the error.

EBADF - not a valid file descriptor.

EACCES - option change is denied in this state (e.g. file descriptor has not be bounded to Network interface yet).

EINVAL - invalid configuration argument. Ethernet type filter is already used by other file descriptor on that same Network interface.

ENODEV - no such Network Interface which is tried to be assigned to the file descriptor exists.

ENOSYS - unsupported operation, passed configuration option does not exists.

fcntl() `fcntl()` is used to manipulate with properties of opened ESP-NETIF L2 TAP file descriptor.

The following commands manipulate the status flags associated with file descriptor:

- `F_GETFD` - the function returns the file descriptor flags, the third argument is ignored.
- `F_SETFD` - sets the file descriptor flags to the value specified by the third argument. Zero is returned.

On error, -1 is returned, and `errno` is set to indicate the error.

EBADF - not a valid file descriptor.

ENOSYS - unsupported command.

read() Opened and configured ESP-NETIF L2 TAP file descriptor can be accessed by `read()` to get inbound frames. The read operation can be either blocking or non-blocking based on actual state of `O_NONBLOCK` file status flag. When the file status flag is set blocking, the read operation waits until a frame is received and context is switched to other task. When the file status flag is set non-blocking, the read operation returns immediately. In such case, either a frame is returned if it was already queued or the function indicates the queue is empty. The number of queued frames associated with one file descriptor is limited by *CONFIG_ESP_NETIF_L2_TAP_RX_QUEUE_SIZE* Kconfig option. Once the number of queued frames reach configured threshold, the newly arriving frames are dropped until the queue has enough room to accept incoming traffic (Tail Drop queue management).

On success, `read()` returns the number of bytes read. Zero is returned when size of the destination buffer is 0. On error, -1 is returned, and `errno` is set to indicate the error.

EBADF - not a valid file descriptor.

EAGAIN - the file descriptor has been marked non-blocking (`O_NONBLOCK`), and the read would block.

write() A raw Data Link Layer frame can be sent to Network Interface via opened and configured ESP-NETIF L2 TAP file descriptor. User's application is responsible to construct the whole frame except for fields which are added automatically by the physical interface device. The following fields need to be constructed by the user's application in case of Ethernet link: source/destination MAC addresses, Ethernet type, actual protocol header and user data. See below for more information about Ethernet frame structure.

```
+-------------------+-------------------+-------------+----------------------------
↪---      --+
|  Destination MAC  |     Source MAC    | Type/Length | Payload (protocol header/
↪data) ...    |
+-------------------+-------------------+-------------+----------------------------
↪---      --+
        6B                  6B                2B                 0-1486B
```

In other words, there is no additional frame processing performed by the ESP-NETIF L2 TAP interface. It only checks the Ethernet type of the frame is the same as the filter configured in the file descriptor. If the Ethernet type is different, an error is returned and the frame is not sent. Note that the `write()` may block in current implementation when accessing a Network interface since it is a shared resource among multiple ESP-NETIF L2 TAP file descriptors and IP stack, and there is currently no queuing mechanism deployed.

On success, `write()` returns the number of bytes written. Zero is returned when size of the input buffer is 0. On error, -1 is returned, and `errno` is set to indicate the error.

EBADF - not a valid file descriptor.

EBADMSG - Ethernet type of the frame is different then file descriptor configured filter.

EIO - Network interface not available or busy.

close() Opened ESP-NETIF L2 TAP file descriptor can be closed by the `close()` to free its allocated resources. The ESP-NETIF L2 TAP implementation of `close()` may block. On the other hand, it is thread safe and can be called from different task than the file descriptor is actually used. If such situation occurs and one task is blocked in I/O operation and another task tries to close the file descriptor, the first task is unblocked. The first's task read operation then ends with error.

On success, `close()` returns zero. On error, -1 is returned, and `errno` is set to indicate the error.

EBADF - not a valid file descriptor.

select() Select is used in a standard way, just *CONFIG_VFS_SUPPORT_SELECT* needs to be enabled to be the `select()` function available.

SNTP API You can find a brief introduction to SNTP in general, its initialization code and basic modes in *SNTP 时间同步* section in the *System Time Document*.

This section provides more details about specific use cases of SNTP service, with statically configured servers, or using DHCP provided servers, or both. The workflow is usually very simple:

1) Initialize and configure the service using *esp_netif_sntp_init()*.
2) Start the service via *esp_netif_sntp_start()*. This step is not needed if we auto-started the service in the previous step (default). It's useful to start the service explicitly after connecting, if we want to use DHCP obtained NTP servers. (This option needs to be enabled before connecting, but SNTP service should be started after)
3) Wait for the system time to synchronize using *esp_netif_sntp_sync_wait()* (only if needed).
4) Stop and destroy the service using *esp_netif_sntp_deinit()*.

Basic mode with statically defined server(s) Initialize the module with the default configuration after connecting to network. Note that it's possible to provide multiple NTP servers in the configuration struct:

```
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: If we want to configure multiple SNTP servers, we have to update lwIP configuration *CONFIG_LWIP_SNTP_MAX_SERVERS*.

Use DHCP obtained SNTP server(s) First of all, we have to enable lwIP configuration option *CONFIG_LWIP_DHCP_GET_NTP_SRV*. Then we have to initialize the SNTP module with the DHCP option and no NTP server:

```
esp_sntp_config_t config = ESP_NETIF_SNTP_DEFAULT_CONFIG_MULTIPLE(0, {} );
config.start = false;                       // start SNTP service explicitly
config.server_from_dhcp = true;             // accept NTP offer from DHCP server
esp_netif_sntp_init(&config);
```

Then, once we're connected, we could start the service using:

```
esp_netif_sntp_start();
```

备注: It's also possible to start the service during initialization (default `config.start=true`). This would likely cause the initial SNTP request to fail (since we are not connected yet) and thus some backoff time for subsequent requests.

Use both static and dynamic servers Very similar to the scenario above (DHCP provided SNTP server), but in this configuration we need to make sure that the static server configuration is refreshed when obtaining NTP servers by DHCP. The underlying lwIP code cleans up the rest of the list of NTP servers when DHCP provided information gets accepted. Thus the ESP-NETIF SNTP module saves the statically configured server(s) and reconfigures them after obtaining DHCP lease.

The typical configuration now looks as per below, providing the specific `IP_EVENT` to update the config and index of the first server to reconfigure (for example setting `config.index_of_first_server=1` would keep DHCP provided server at index 0, and the statically configured server at index 1).

```
esp_sntp_config_t config = ESP_NETIF_SNTP_DEFAULT_CONFIG("pool.ntp.org");
config.start = false;                       // start SNTP service explicitly␣
↪(after connecting)
```

(下页继续)

(续上页)

```
config.server_from_dhcp = true;              // accept NTP offers from DHCP server
config.renew_servers_after_new_IP = true;    // let esp-netif update configured␣
↪SNTP server(s) after receiving DHCP lease
config.index_of_first_server = 1;            // updates from server num 1, leaving␣
↪server 0 (from DHCP) intact
config.ip_event_to_renew = IP_EVENT_STA_GOT_IP;  // IP event on which we refresh␣
↪the configuration
```

Then we start the service normally with *esp_netif_sntp_start()*.

ESP-NETIF programmer's manual Please refer to the example section for basic initialization of default interfaces:

- WiFi Station: wifi/getting_started/station/main/station_example_main.c
- Ethernet: ethernet/basic/main/ethernet_example_main.c
- L2 TAP: protocols/l2tap/main/l2tap_main.c
- WiFi Access Point: wifi/getting_started/softAP/main/softap_example_main.c

For more specific cases please consult this guide: *ESP-NETIF Custom I/O Driver*.

WiFi default initialization The initialization code as well as registering event handlers for default interfaces, such as softAP and station, are provided in separate APIs to facilitate simple startup code for most applications:

- *esp_netif_create_default_wifi_sta()*
- *esp_netif_create_default_wifi_ap()*

Please note that these functions return the `esp_netif` handle, i.e. a pointer to a network interface object allocated and configured with default settings, which as a consequence, means that:

- The created object has to be destroyed if a network de-initialization is provided by an application using *esp_netif_destroy_default_wifi()*.
- These *default* interfaces must not be created multiple times, unless the created handle is deleted using *esp_netif_destroy()*.
- When using Wifi in `AP+STA` mode, both these interfaces has to be created.

API Reference

Header File

- components/esp_netif/include/esp_netif.h

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** –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** –
- **event_id** –
- **data** –

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** –
- **event_id** –
- **data** –

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** –
- **event_id** –
- **data** –

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** –
- **event_id** –
- **data** –

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** –
- **event_id** –
- **data** –

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** –
- **event_id** –
- **data** –

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** –
- **event_id** –
- **data** –

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** –
- **event_id** –
- **data** –

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** –
- **event_id** –
- **data** –

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.

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.

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

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 ommited: “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 paramters

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.

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

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 *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

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

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

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

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

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**

WiFi default API reference

Header File

- components/esp_wifi/include/esp_wifi_default.h

Functions

esp_err_t **esp_netif_attach_wifi_station**(*esp_netif_t* *esp_netif)

Attaches wifi station interface to supplied netif.

参数 **esp_netif** – instance to attach the wifi station to

返回

- ESP_OK on success
- ESP_FAIL if attach failed

esp_err_t **esp_netif_attach_wifi_ap**(*esp_netif_t* *esp_netif)

Attaches wifi soft AP interface to supplied netif.

参数 **esp_netif** – instance to attach the wifi AP to

返回

- ESP_OK on success
- ESP_FAIL if attach failed

esp_err_t **esp_wifi_set_default_wifi_sta_handlers**(void)

Sets default wifi event handlers for STA interface.

返回

- ESP_OK on success, error returned from esp_event_handler_register if failed

esp_err_t **esp_wifi_set_default_wifi_ap_handlers**(void)

Sets default wifi event handlers for AP interface.

返回

- ESP_OK on success, error returned from esp_event_handler_register if failed

esp_err_t **esp_wifi_set_default_wifi_nan_handlers**(void)

Sets default wifi event handlers for NAN interface.

返回

- ESP_OK on success, error returned from esp_event_handler_register if failed

esp_err_t **esp_wifi_clear_default_wifi_driver_and_handlers**(void *esp_netif)

Clears default wifi event handlers for supplied network interface.

参数 **esp_netif** – instance of corresponding if object

返回

- ESP_OK on success, error returned from esp_event_handler_register if failed

esp_netif_t ***esp_netif_create_default_wifi_ap**(void)

Creates default WIFI AP. In case of any init error this API aborts.

备注: The API creates esp_netif object with default WiFi access point config, attaches the netif to wifi and registers default wifi handlers.

返回 pointer to esp-netif instance

esp_netif_t ***esp_netif_create_default_wifi_sta**(void)

Creates default WIFI STA. In case of any init error this API aborts.

备注: The API creates esp_netif object with default WiFi station config, attaches the netif to wifi and registers default wifi handlers.

返回 pointer to esp-netif instance

esp_netif_t ***esp_netif_create_default_wifi_nan**(void)

Creates default WIFI NAN. In case of any init error this API aborts.

备注: The API creates esp_netif object with default WiFi station config, attaches the netif to wifi and registers default wifi handlers.

返回 pointer to esp-netif instance

void **esp_netif_destroy_default_wifi**(void *esp_netif)

Destroys default WIFI netif created with esp_netif_create_default_wifi_…() API.

备注: This API unregisters wifi handlers and detaches the created object from the wifi. (this function is a no-operation if esp_netif is NULL)

参数 **esp_netif** –**[in]** object to detach from WiFi and destroy

esp_netif_t ***esp_netif_create_wifi**(*wifi_interface_t* wifi_if, const *esp_netif_inherent_config_t* *esp_netif_config)

Creates esp_netif WiFi object based on the custom configuration.

Attention This API DOES NOT register default handlers!

参数

- **wifi_if** –**[in]** type of wifi interface
- **esp_netif_config** –inherent esp-netif configuration pointer

返回 pointer to esp-netif instance

esp_err_t **esp_netif_create_default_wifi_mesh_netifs**(*esp_netif_t* **p_netif_sta, *esp_netif_t* **p_netif_ap)

Creates default STA and AP network interfaces for esp-mesh.

Both netifs are almost identical to the default station and softAP, but with DHCP client and server disabled. Please note that the DHCP client is typically enabled only if the device is promoted to a root node.

Returns created interfaces which could be ignored setting parameters to NULL if an application code does not need to save the interface instances for further processing.

参数

- **p_netif_sta** –**[out]** pointer where the resultant STA interface is saved (if non NULL)
- **p_netif_ap** –**[out]** pointer where the resultant AP interface is saved (if non NULL)

返回 ESP_OK on success

2.5.5 IP 网络层协议

ESP-NETIF Custom I/O Driver

This section outlines implementing a new I/O driver with esp-netif connection capabilities. By convention the I/O driver has to register itself as an esp-netif driver and thus holds a dependency on esp-netif component and is responsible for providing data path functions, post-attach callback and in most cases also default event handlers to define network interface actions based on driver's lifecycle transitions.

Packet input/output As shown in the diagram, the following three API functions for the packet data path must be defined for connecting with esp-netif:

- *esp_netif_transmit()*
- *esp_netif_free_rx_buffer()*
- *esp_netif_receive()*

The first two functions for transmitting and freeing the rx buffer are provided as callbacks, i.e. they get called from esp-netif (and its underlying TCP/IP stack) and I/O driver provides their implementation.

The receiving function on the other hand gets called from the I/O driver, so that the driver's code simply calls *esp_netif_receive()* on a new data received event.

Post attach callback A final part of the network interface initialization consists of attaching the esp-netif instance to the I/O driver, by means of calling the following API:

```
esp_err_t esp_netif_attach(esp_netif_t *esp_netif, esp_netif_iodriver_handle␣
↪driver_handle);
```

It is assumed that the `esp_netif_iodriver_handle` is a pointer to driver's object, a struct derived from `struct esp_netif_driver_base_s`, so that the first member of I/O driver structure must be this base structure with pointers to

- post-attach function callback
- related esp-netif instance

As a consequence the I/O driver has to create an instance of the struct per below:

```
typedef struct my_netif_driver_s {
        esp_netif_driver_base_t base;           /*!< base structure reserved as␣
↪esp-netif driver */
        driver_impl             *h;             /*!< handle of driver␣
↪implementation */
    } my_netif_driver_t;
```

with actual values of `my_netif_driver_t::base.post_attach` and the actual drivers handle `my_netif_driver_t::h`. So when the *esp_netif_attach()* gets called from the initialization code, the post-attach callback from I/O driver's code gets executed to mutually register callbacks between esp-netif and I/O driver instances. Typically the driver is started as well in the post-attach callback. An example of a simple post-attach callback is outlined below:

```
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));
```

(下页继续)

(续上页)

```
    my_driver_start(driver->driver_impl);
    return ESP_OK;
}
```

Default handlers I/O drivers also typically provide default definitions of lifecycle behaviour of related network interfaces based on state transitions of I/O drivers. For example *driver start -> network start*, etc. An example of such a default handler is provided below:

```
esp_err_t my_driver_netif_set_default_handlers(my_netif_driver_t *driver, esp_
↪netif_t * esp_netif)
{
    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;
}
```

Network stack connection The packet data path functions for transmitting and freeing the rx buffer (defined in the I/O driver) are called from the esp-netif, specifically from its TCP/IP stack connecting layer.

Note, that IDF provides several network stack configurations for the most common network interfaces, such as for the WiFi station or Ethernet. These configurations are defined in esp_netif/include/esp_netif_defaults.h and should be sufficient for most network drivers. (In rare cases, expert users might want to define custom lwIP based interface layers; it is possible, but an explicit dependency to lwIP needs to be set)

The following API reference outlines these network stack interaction with the esp-netif:

Header File

- components/esp_netif/include/esp_netif_net_stack.h

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 (if supported)

Note that it is not supported to acquire PPP netif impl pointer and this function will return NULL for esp_netif instances configured to PPP mode

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。

2.5.6 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。

2.6 外设 API

2.6.1 Analog to Digital Converter (ADC) Oneshot Mode Driver

Introduction

The Analog to Digital Converter is an on-chip sensor which is able to measure analog signals from dedicated analog IO pads.

ESP32-C2 has one ADC unit(s), which can be used in scenario(s) like:

- Generate one-shot ADC conversion result

This guide will introduce ADC oneshot mode conversion.

Functional Overview

The following sections of this document cover the typical steps to install and operate an ADC:

- *Resource Allocation* - covers which parameters should be set up to get an ADC handle and how to recycle the resources when ADC finishes working.
- *Unit Configuration* - covers the parameters that should be set up to configure the ADC unit, so as to get ADC conversion raw result.
- *Read Conversion Result* - covers how to get ADC conversion raw result.
- *Hardware Limitations* - describes the ADC related hardware limitations.
- *Power Management* - covers power management related.
- *IRAM Safe* - describes tips on how to read ADC conversion raw result when 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.

Resource Allocation The ADC oneshot mode driver is implemented based on ESP32-C2 SAR ADC module. Different ESP chips might have different number of independent ADCs. From oneshot mode driver's point of view, an ADC instance is represented by `adc_oneshot_unit_handle_t`.

To install an ADC instance, set up the required initial configuration structure `adc_oneshot_unit_init_cfg_t`:

- `adc_oneshot_unit_init_cfg_t::unit_id` selects the ADC. Please refer to the datasheet to know dedicated analog IOs for this ADC.
- `adc_oneshot_unit_init_cfg_t::clk_src` selects the source clock of the ADC. If it's set to 0, driver will fallback to use a default clock source, see `adc_oneshot_clk_src_t` to know the details.
- `adc_oneshot_unit_init_cfg_t::ulp_mode` sets if the ADC will be working under ULP mode.

After setting up the initial configurations for the ADC, call `adc_oneshot_new_unit()` with the prepared `adc_oneshot_unit_init_cfg_t`. This function will return an ADC unit handle, if the allocation is successful.

This function may fail due to various errors such as invalid argumemts, insufficient memory, etc. Specifically, when the to-be-allocated ADC instance is registered already, this function will return `ESP_ERR_NOT_FOUND` error. Number of available ADC(s) is recorded by `SOC_ADC_PERIPH_NUM`.

If a previously created ADC instance is no loger required, you should recycle the ADC instance by calling `adc_oneshot_del_unit()`, related hardware and software resources will be recycled as well.

Create an ADC Unit Handle under Normal Oneshot Mode

```
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));
```

Recycle the ADC Unit

```
ESP_ERROR_CHECK(adc_oneshot_del_unit(adc1_handle));
```

Unit Configuration After an ADC instance is created, set up the `adc_oneshot_chan_cfg_t` to configure ADC IOs to measure analog signal:

- `adc_oneshot_chan_cfg_t::atten`, ADC attenuation. Refer to the On-Chip Sensor chapter in TRM.
- `adc_oneshot_chan_cfg_t::bitwidth`, the bitwidth of the raw conversion result.

备注: For the IO corresponding ADC channel number. Check datasheet to know the ADC IOs. On the other hand, `adc_continuous_io_to_channel()` and `adc_continuous_channel_to_io()` can be used to know the ADC channels and ADC IOs.

To make these settings take effect, call *`adc_oneshot_config_channel()`* with above configuration structure. You should specify an ADC channel to be configured as well. This function (*`adc_oneshot_config_channel()`*) can be called multiple times to configure different ADC channels. The Driver will save each of these channel configurations internally.

Configure Two ADC Channels

```
adc_oneshot_chan_cfg_t config = {
    .bitwidth = ADC_BITWIDTH_DEFAULT,
    .atten = ADC_ATTEN_DB_11,
};
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));
```

Read Conversion Result After above configurations, the ADC is ready to measure the analog signal(s) from the configured ADC channel(s). Call *`adc_oneshot_read()`* to get the conversion raw result of an ADC channel.

- *`adc_oneshot_read()`* is safe to use. ADC(s) are shared by some other drivers / peripherals, see *Hardware Limitations*. This function uses mutexes to avoid concurrent hardware usage. Therefore, this function should not be used in an ISR context. This function may fail when the ADC is in use by other drivers / peripherals, and return *`ESP_ERR_TIMEOUT`*. Under this condition, the ADC raw result is invalid.

These two functions will both fail due to invalid arguments.

The ADC conversion results read from these two functions are raw data. To calculate the voltage based on the ADC raw results, this formula can be used:

```
Vout = Dout * Vmax / Dmax        (1)
```

where:

Vout	Digital output result, standing for the voltage.
Dout	ADC raw digital reading result.
Vmax	Maximum measurable input analog voltage, this is related to the ADC attenuation, please refer to the On-Chip Sensor chapter in TRM.
Dmax	Maximum of the output ADC raw digital reading result, which is 2^bitwidth, where bitwidth is the :cpp:member::*adc_oneshot_chan_cfg_t:bitwidth* configured before.

To do further calbration to convert the ADC raw result to voltage in mV, please refer to calibration doc *Analog to Digital Converter (ADC) Calibration Driver*.

Read Raw Result

```
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]);
```

Hardware Limitations

- Random Number Generator uses ADC as a input source. When ADC `adc_oneshot_read()` works, the random number generated from RNG will be less random.

Power Management When power management is enabled (i.e. *CONFIG_PM_ENABLE* is on), the system clock frequency may be adjusted when the system is in an idle state. However, the ADC oneshot mode driver works in a polling routine, the `adc_oneshot_read()` will poll the CPU until the function returns. During this period of time, the task in which ADC oneshot mode driver resides won't be blocked. Therefore the clock frequency is stable when reading.

IRAM Safe By default, all the ADC oneshot mode driver APIs are not supposed to be run when the Cache is disabled (Cache may be disabled due to many reasons, such as Flash writing/erasing, OTA, etc.). If these APIs executes when the Cache is disabled, you will probably see errors like Illegal Instruction or Load/Store Prohibited.

Thread Safety

- `adc_oneshot_new_unit()`
- `adc_oneshot_config_channel()`
- `adc_oneshot_read()`

Above functions are guaranteed to be thread safe. Therefore, you can call them from different RTOS tasks without protection by extra locks.

- `adc_oneshot_del_unit()` is not thread safe. Besides, concurrently calling this function may result in failures of above thread-safe APIs.

Kconfig Options

- *CONFIG_ADC_ONESHOT_CTRL_FUNC_IN_IRAM* controls where to place the ADC fast read function (IRAM or Flash), see *IRAM Safe* for more details.

Application Examples

- ADC oneshot mode example: peripherals/adc/oneshot_read.

API Reference

Header File

- components/hal/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 (1.33 x)

enumerator **ADC_ATTEN_DB_6**

The input voltage of ADC will be attenuated extending the range of measurement by about 6 dB (2 x)

enumerator **ADC_ATTEN_DB_11**

The input voltage of ADC will be attenuated extending the range of measurement by about 11 dB (3.55 x)

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.

Header File

- components/esp_adc/include/esp_adc/adc_oneshot.h

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* *unit_id, *adc_channel_t* *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 *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.

2.6.2 Analog to Digital Converter (ADC) Calibration Driver

Introduction

Based on series of comparisons with the reference voltage, ESP32-C2 ADC determines each bit of the output digital result. Per design the ESP32-C2 ADC reference voltage is 1100 mV, however the true reference voltage can range from 1000 mV to 1200 mV among different chips. This guide will introduce an ADC calibration driver to minimize this difference.

Functional Overview

The following sections of this document cover the typical steps to install and use the ADC calibration driver:

- *Calibration Scheme Creation* - covers how to create a calibration scheme handle and delete the calibration scheme handle.
- *Result Conversion* - convers how to convert ADC raw result to calibrated result.
- *Thread Safety* - lists which APIs are guaranteed to be thread safe by the driver.
- *Minimize Noise* - describes a general way to minimize the noise.

Calibration Scheme Creation The ADC calibration driver provides ADC calibration scheme(s). From calibration driver's point of view, an ADC calibration scheme is created to an ADC calibration handle *adc_cali_handle_t*.

adc_cali_check_scheme() can be used to know which calibration scheme is supported on the chip. For those users who are already aware of the supported scheme, this step can be skipped. Just call the corresponding function to create the scheme handle.

For those users who use their custom ADC calibration schemes, you could either modify this function *adc_cali_check_scheme()*, or just skip this step and call your custom creation function.

ADC Calibration Line Fitting Scheme ESP32-C2 supports ADC_CALI_SCHEME_VER_LINE_FITTING scheme. To create this scheme, set up adc_cali_line_fitting_config_t first.

- adc_cali_line_fitting_config_t::unit_id, the ADC that your ADC raw results are from.
- adc_cali_line_fitting_config_t::atten, ADC attenuation that your ADC raw results use.
- adc_cali_line_fitting_config_t::bitwidth, the ADC raw result bitwidth.

After setting up the configuration structure, call adc_cali_create_scheme_line_fitting() to create a Line Fitting calibration scheme handle.

```
ESP_LOGI(TAG, "calibration scheme version is %s", "Line Fitting");
adc_cali_line_fitting_config_t cali_config = {
    .unit_id = unit,
    .atten = atten,
    .bitwidth = ADC_BITWIDTH_DEFAULT,
};
ESP_ERROR_CHECK(adc_cali_create_scheme_line_fitting(&cali_config, &handle));
```

When the ADC calibration is no longer used, please delete the calibration scheme handle by calling adc_cali_delete_scheme_line_fitting().

Delete Line Fitting Scheme

```
ESP_LOGI(TAG, "delete %s calibration scheme", "Line Fitting");
ESP_ERROR_CHECK(adc_cali_delete_scheme_line_fitting(handle));
```

备注: For users who want to use their custom calibration schemes, you could provide a creation function to create your calibration scheme handle. Check the function table *adc_cali_scheme_t* in *components/esp_adc/interface/adc_cali_interface.h* to know the ESP ADC calibration interface.

Result Conversion After setting up the calibration characteristics, you can call *adc_cali_raw_to_voltage()* to convert the ADC raw result into calibrated result. The calibrated result is in the unit of mV. This function may fail due to invalid argument. Especailly, if this function returns *ESP_ERR_INVALID_STATE*, this means the calibration scheme isn't created. You need to create a calibration scheme handle, use *adc_cali_check_scheme()* to know the supported calibration scheme. On the other hand, you could also provide a custom calibration scheme and create the handle.

备注: ADC calibration is only supported under ADC_ATTEN_DB_0 and ADC_ATTEN_DB_11. Under ADC_ATTEN_DB_0, input voltage higher than 950 mV is not supported. Under ADC_ATTEN_DB_11, input voltage higher than 2800 mV is not supported.

Get Voltage

```
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]);
```

(下页继续)

(续上页)

Thread Safety The factory function `esp_adc_cali_new_scheme()` is guaranteed to be thread safe by the driver. Therefore, you can call them from different RTOS tasks without protection by extra locks.

Other functions that take the *adc_cali_handle_t* as the first positional parameter are not thread safe, you should avoid calling them from multiple tasks.

Minimize Noise The ESP32-C2 ADC can be sensitive to noise leading to large discrepancies in ADC readings. Depending on the usage scenario, you may need to connect a bypass capacitor (e.g. a 100 nF ceramic capacitor) to the ADC input pad in use, to minimize noise. Besides, multisampling may also be used to further mitigate the effects of noise.

API Reference

Header File

- components/esp_adc/include/esp_adc/adc_cali.h

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

2.6.3 Clock Tree

The clock subsystem of ESP32-C2 is used to source and distribute system/module clocks from a range of root clocks. The clock tree driver maintains the basic functionality of the system clock and the intricate relationship among module clocks.

This document starts with the introduction to root and module clocks. Then it covers the clock tree APIs that users can call to monitor the status of the module clocks at runtime.

Introduction

This section lists definitions of the ESP32-C2's supported root clocks and module clocks. These definitions are commonly used in the driver configuration, to help user select a proper source clock for the peripheral.

Root Clocks Root clocks generate reliable clock signals. These clock signals then pass through various gates, muxes, dividers, or multipliers to become the clock sources for every functional module: the CPU core(s), WIFI, BT, the RTC, and the peripherals.

ESP32-C2's root clocks are listed in *soc_root_clk_t*:

- Internal 17.5MHz RC Oscillator (RC_FAST)
  This RC oscillator generates a ~17.5MHz clock signal output as the RC_FAST_CLK.
  The ~17.5MHz signal output is also passed into a configurable divider, which by default divides the input clock frequency by 256, to generate a RC_FAST_D256_CLK.
  The exact frequency of RC_FAST_CLK can be computed in runtime through calibration on the RC_FAST_D256_CLK.
- External 40/26MHz Crystal (XTAL)
- Internal 136kHz RC Oscillator (RC_SLOW)
  This RC oscillator generates a ~136kHz clock signal output as the RC_SLOW_CLK.
  The exact frequency of this clock can be computed in runtime through calibration.
- External Slow Clock - optional (OSC_SLOW)
  A clock signal generated by an external circuit can be connected to pin0 to be the clock source for the RTC_SLOW_CLK. This clock can also be calibrated to get its exact frequency.

Typically, the frequency of the signal generated from a RC oscillator circuit is less accurate and more sensitive to environment comparing to the signal generated from a crystal. ESP32-C2 provides several clock source options for the RTC_SLOW_CLK, and users can make the choice based on the requirements for system time accuracy and power consumption (refer to *RTC 定时器时钟源* for more details).

Module Clocks ESP32-C2's available module clocks are listed in *soc_module_clk_t*. Each module clock has a unique ID. You can get more information on each clock by checking the documented enum value.

API Usage

The clock tree driver provides an all-in-one API to get the frequency of the module clocks, *esp_clk_tree_src_get_freq_hz()*. Users can call this function at any moment, with specifying the clock name (*soc_module_clk_t*) and the desired degree of precision of the returned frequency value (*esp_clk_tree_src_freq_precision_t*).

API Reference

Header File

- components/soc/esp32c2/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_RC_FAST_D256_FREQ_APPROX

Approximate RC_FAST_D256_CLK frequency in Hz

SOC_CLK_OSC_SLOW_FREQ_APPROX

Approximate OSC_SLOW_CLK (external slow clock) 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_TEMP_SENSOR_CLKS

Array initializer for all supported clock sources of Temperature Sensor.

SOC_SPI_CLKS

Array initializer for all supported clock sources of SPI.

SOC_I2C_CLKS

Array initializer for all supported clock sources of I2C.

SOC_ADC_DIGI_CLKS

Array initializer for all supported clock sources of ADC digital controller.

SOC_GLITCH_FILTER_CLKS

Array initializer for all supported clock sources of Glitch Filter.

SOC_MWDT_CLKS

Array initializer for all supported clock sources of MWDT.

SOC_LEDC_CLKS

Array initializer for all supported clock sources of LEDC.

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 26/40MHz crystal

enumerator **SOC_ROOT_CLK_EXT_OSC_SLOW**

External slow clock signal at pin0, only support 32.768 KHz currently

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_PLL**

Select PLL_CLK as CPU_CLK source (PLL_CLK is the output of 26/40MHz crystal oscillator frequency multiplier, 480MHz)

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_OSC_SLOW**

Select OSC_SLOW_CLK (external slow clock) as RTC_SLOW_CLK source

enumerator **SOC_RTC_SLOW_CLK_SRC_RC_FAST_D256**

Select RC_FAST_D256_CLK (referred as FOSC_DIV or 8m_d256/8md256 in TRM and reg. description) 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_XTAL_D2**

Select XTAL_D2_CLK (may referred as XTAL_CLK_DIV_2) as RTC_FAST_CLK source

enumerator **SOC_RTC_FAST_CLK_SRC_XTAL_DIV**

Alias name for `SOC_RTC_FAST_CLK_SRC_XTAL_D2`

enumerator **SOC_RTC_FAST_CLK_SRC_RC_FAST**

Select RC_FAST_CLK as RTC_FAST_CLK source

enumerator **SOC_RTC_FAST_CLK_SRC_INVALID**

Invalid RTC_FAST_CLK source

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, PLL, or RC_FAST by configuring soc_cpu_clk_src_t

enumerator **SOC_MOD_CLK_RTC_FAST**

RTC_FAST_CLK can be sourced from XTAL_D2 or RC_FAST 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 RC_FAST_D256 by configuring soc_rtc_slow_clk_src_t

enumerator **SOC_MOD_CLK_APB**

APB_CLK is always 40MHz no matter it derives from XTAL or PLL

enumerator **SOC_MOD_CLK_PLL_F40M**

PLL_F40M_CLK is derived from PLL, and has a fixed frequency of 40MHz

enumerator **SOC_MOD_CLK_PLL_F60M**

PLL_F60M_CLK is derived from PLL, and has a fixed frequency of 60MHz

enumerator **SOC_MOD_CLK_PLL_F80M**

PLL_F80M_CLK is derived from PLL, and has a fixed frequency of 80MHz

enumerator **SOC_MOD_CLK_OSC_SLOW**

OSC_SLOW_CLK comes from an external slow clock signal, 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_RC_FAST_D256**

RC_FAST_D256_CLK comes from the internal 20MHz rc oscillator, divided by 256, and passing a clock gating to the peripherals

enumerator **SOC_MOD_CLK_XTAL**

XTAL_CLK comes from the external 26/40MHz crystal

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_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_F40M**

Select PLL_F40M as the source clock

enumerator **GPTIMER_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **GPTIMER_CLK_SRC_DEFAULT**

Select PLL_F40M 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_F40M**

Timer group clock source is PLL_F40M

enumerator **TIMER_SRC_CLK_XTAL**

Timer group clock source is XTAL

enumerator **TIMER_SRC_CLK_DEFAULT**

Timer group clock source default choice is PLL_F40M

enum **soc_periph_temperature_sensor_clk_src_t**

Type of Temp Sensor clock source.

Values:

enumerator **TEMPERATURE_SENSOR_CLK_SRC_XTAL**

Select XTAL as the source clock

enumerator **TEMPERATURE_SENSOR_CLK_SRC_RC_FAST**

Select RC_FAST as the source clock

enumerator **TEMPERATURE_SENSOR_CLK_SRC_DEFAULT**

Select XTAL as the default choice

enum **soc_periph_uart_clk_src_legacy_t**

Type of UART clock source, reserved for the legacy UART driver.

Values:

enumerator **UART_SCLK_PLL_F40M**

UART source clock is PLL_F40M CLK

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_F40M

enum **soc_periph_spi_clk_src_t**

Type of SPI clock source.

Values:

enumerator **SPI_CLK_SRC_DEFAULT**
Select PLL_40M as SPI source clock

enumerator **SPI_CLK_SRC_PLL_F40M**
Select PLL_40M as SPI source clock

enumerator **SPI_CLK_SRC_XTAL**
Select XTAL as SPI source 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 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_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_glitch_filter_clk_src_t**
Glitch filter clock source.
Values:

enumerator **GLITCH_FILTER_CLK_SRC_APB**
Select APB clock as the source clock

enumerator **GLITCH_FILTER_CLK_SRC_DEFAULT**
Select APB clock 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_F40M**

Select PLL 40 Mhz as the source clock

enumerator **MWDT_CLK_SRC_DEFAULT**

Select PLL 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_PLL_DIV_CLK**

Select PLL_F60M as the source clock

enumerator **LEDC_USE_RC_FAST_CLK**

Select RC_FAST as the source clock

enumerator **LEDC_USE_XTAL_CLK**

Select XTAL as the source clock

enumerator **LEDC_USE_RTC8M_CLK**

Alias of ‘LEDC_USE_RC_FAST_CLK’

Header File

- components/esp_hw_support/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**

2.6.4 GPIO & RTC GPIO

GPIO 汇总

ESP32-C2 芯片具有 21 个物理 GPIO 管脚（GPIO0 ~ GPIO20）。对于内置 SiP flash 的芯片型号，GPIO11 ~ GPIO17 专门用于连接 SiP flash。因此，对于这类芯片只有 14 个 GPIO 管脚可用。

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-C2 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	注释
GPIO0	ADC1_CH0	RTC
GPIO1	ADC1_CH1	RTC
GPIO2	ADC1_CH2	RTC
GPIO3	ADC1_CH3	RTC
GPIO4	ADC1_CH4	RTC
GPIO5		RTC
GPIO6		
GPIO7		
GPIO8		Strapping 管脚
GPIO9		Strapping 管脚
GPIO10		
GPIO11		
GPIO12		SPI0/1
GPIO13		SPI0/1
GPIO14		SPI0/1
GPIO15		SPI0/1
GPIO16		SPI0/1
GPIO17		SPI0/1
GPIO18		
GPIO19		
GPIO20		

备注:

- Strapping 管脚：GPIO8 和 GPIO9 是 Strapping 管脚。更多信息请参考 ESP8684 技术规格书。
- SPI0/1：GPIO12-17 通常用于 SPI flash，不推荐用于其他用途。
- RTC：GPIO0-5 可以在 Deep-sleep 模式时使用。

GPIO 毛刺过滤器

ESP32-C2 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。

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()*。

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/driver/gpio/include/driver/gpio.h

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 chip or system is reset, for example, when watchdog time-out or Deep-sleep events are triggered.

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.

However, 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 Deep-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 hight 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_deep_sleep_hold_en**(void)

Enable all digital gpio pads hold function during Deep-sleep.

Enabling this feature makes all digital gpio pads be at the holding state during Deep-sleep. The state of each pad holds is its active configuration (not pad's sleep configuration!).

Note that this pad hold feature only works when the chip is in Deep-sleep mode. When the chip is in active mode, the digital gpio state can be changed freely even you have called this function.

After this API is being called, the digital gpio Deep-sleep hold feature will work during every sleep process. You should call `gpio_deep_sleep_hold_dis` to disable this feature.

void **gpio_deep_sleep_hold_dis**(void)

Disable all digital gpio pads hold function during Deep-sleep.

void **gpio_iomux_in**(uint32_t gpio_num, uint32_t signal_idx)

SOC_GPIO_SUPPORT_HOLD_SINGLE_IO_IN_DSLP.

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 oen_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`.

- **oen_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:** 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.

esp_err_t **gpio_force_unhold_all**(void)

Force unhold all digital and rtc gpio pads.

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

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

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

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

Enumerations

enum **gpio_port_t**

Values:

enumerator **GPIO_PORT_0**

enumerator **GPIO_PORT_MAX**

enum **gpio_num_t**

Values:

enumerator **GPIO_NUM_NC**

Use to signal not connected to S/W

enumerator **GPIO_NUM_0**

GPIO0, input and output

enumerator **GPIO_NUM_1**

GPIO1, input and output

enumerator **GPIO_NUM_2**

GPIO2, input and output

enumerator **GPIO_NUM_3**
GPIO3, input and output

enumerator **GPIO_NUM_4**
GPIO4, input and output

enumerator **GPIO_NUM_5**
GPIO5, input and output

enumerator **GPIO_NUM_6**
GPIO6, input and output

enumerator **GPIO_NUM_7**
GPIO7, input and output

enumerator **GPIO_NUM_8**
GPIO8, input and output

enumerator **GPIO_NUM_9**
GPIO9, input and output

enumerator **GPIO_NUM_10**
GPIO10, input and output

enumerator **GPIO_NUM_11**
GPIO11, input and output

enumerator **GPIO_NUM_12**
GPIO12, input and output

enumerator **GPIO_NUM_13**
GPIO13, input and output

enumerator **GPIO_NUM_14**
GPIO14, input and output

enumerator **GPIO_NUM_15**
GPIO15, input and output

enumerator **GPIO_NUM_16**
GPIO16, input and output

enumerator **GPIO_NUM_17**
GPIO17, input and output

enumerator **GPIO_NUM_18**
GPIO18, input and output

enumerator **GPIO_NUM_19**

GPIO19, input and output

enumerator **GPIO_NUM_20**

GPIO20, input and output

enumerator **GPIO_NUM_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_CTRL_EFUSE**

Pad input hysteresis ctrl by efuse

enumerator **GPIO_HYS_SOFT_ENABLE**

Pad input hysteresis enable by software

enumerator **GPIO_HYS_SOFT_DISABLE**

Pad input hysteresis disable by software

API 参考 - GPIO 毛刺过滤器

Header File

- components/driver/gpio/include/driver/gpio_filter.h

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.

2.6.5 通用定时器

简介

通用定时器是 ESP32-C2 定时器组外设的驱动程序。ESP32-C2 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。
- 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。后台驱动会在资源池中管理所有可用的硬件资源，这样您便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。
- 选用 *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,
    .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。

[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-C2 技术参考手册》> 章节定时器组 (TIMG) [PDF]。驱动程序不会禁止您申请更多的定时器，但是当所有可用的硬件资源用完时将会返回错误。在分配资源时，请务必检查返回值（例如 *gptimer_new_timer()*）。

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。

您也可以通过参数 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。您需要确保 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)
{
    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
```

(下页继续)

(续上页)

```
    };
    // 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 = {
    .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, &alarm_config));
```

电源管理 有些电源管理的策略会在某些时刻关闭时钟源，或者改变时钟源的频率，以求降低功耗。比如在启用 DFS 后，APB 时钟源会降低频率。如果浅睡眠（light sleep）模式也被开启，PLL 和 XTAL 时钟都会被默认关闭，从而导致 GPTimer 的计时不准确。

驱动程序会根据具体的时钟源选择，通过创建不同的电源锁来避免上述情况的发生。驱动会在 *gptimer_enable()* 函数中增加电源锁的引用计数，并在 *gptimer_disable()* 函数中减少电源锁的引用计数，从而保证了在 *gptimer_enable()* 和 *gptimer_disable()* 之间，GPTimer 的时钟源始处于稳定工作的状态。

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，这意味着您可以从不同的 RTOS 任务中调用这些函数，而无需额外的互斥锁去保护。以下这些函数还被允许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()*

Kconfig 选项

- *CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM* 控制放置通用定时器控制函数（IRAM 或 flash）的位置。了解更多信息，请参考章节 *IRAM 安全*。
- *CONFIG_GPTIMER_ISR_IRAM_SAFE* 控制默认 ISR 程序在 cache 禁用时是否可以运行。了解更多信息，请参考章节 *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。

[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。

API 参考

Header File

- components/driver/gptimer/include/driver/gptimer.h

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 that user can set new alarm action immediately in the 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.

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

uint32_t **intr_shared**

Set true, the timer interrupt number can be shared with other peripherals

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/driver/gptimer/include/driver/gptimer_etm.h

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/driver/gptimer/include/driver/gptimer_types.h

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

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

2.6.6 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，您必须使用 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：
   - 设置 GPIO 位：`csrrsi rd, csr, imm[4:0]`
   - 清除 GPIO 位：`csrrci rd, csr, imm[4:0]`
   - 注意：只能控制最低位的 4 个 GPIO 通道

通过汇编操作专用 GPIO 的示例代码存放在 ESP-IDF 示例项目的 peripherals/dedicated_gpio 目录下。示例演示了如何通过汇编操作专用 GPIO 来模拟 UART、I2C 和 SPI 总线。

有关支持的专用 GPIO 指令的详细信息，请参考 *ESP32-C2 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。

API 参考

Header File

- components/driver/gpio/include/driver/dedic_gpio.h

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.

2.6.7 I2C 驱动程序

概述

I2C 是一种串行同步半双工通信协议，总线上可以同时挂载多个主机和从机。I2C 总线由串行数据线 (SDA) 和串行时钟线 (SCL) 线构成。这些线都需要上拉电阻。

I2C 具有简单且制造成本低廉等优点，主要用于低速外围设备的短距离通信（一英尺以内）。

ESP32-C2 有 1 个 I2C 控制器（也称为端口），负责处理在 I2C 总线上的通信。每个控制器都可以设置为主机或从机。

驱动程序的功能

I2C 驱动程序管理在 I2C 总线上设备的通信，该驱动程序具备以下功能：

- 在主机模式下读写字节
- 读取并写入寄存器，然后由主机读取/写入

使用驱动程序

以下部分将指导您完成 I2C 驱动程序配置和工作的基本步骤：

1. 配置驱动程序 - 设置初始化参数（如主机模式或从机模式，SDA 和 SCL 使用的 GPIO 管脚，时钟速度等）
2. 安装驱动程序- 激活一个 I2C 控制器的驱动，该控制器可为主机也可为从机
3. 根据是为主机还是从机配置驱动程序，选择合适的项目
   a) 主机模式下通信 - 发起通信（主机模式）
4. 中断处理 - 配置 I2C 中断服务
5. 用户自定义配置 - 调整默认的 I2C 通信参数（如时序、位序等）
6. 错误处理 - 如何识别和处理驱动程序配置和通信错误
7. 删除驱动程序- 在通信结束时释放 I2C 驱动程序所使用的资源

配置驱动程序 建立 I2C 通信第一步是配置驱动程序，这需要设置*i2c_config_t* 结构中的几个参数：

- 设置 I2C **工作模式** - 从*i2c_mode_t* 中选择主机模式或从机模式
- 设置 **通信管脚**
  - 指定 SDA 和 SCL 信号使用的 GPIO 管脚
  - 是否启用 ESP32-C2 的内部上拉电阻
- （仅限主机模式）设置 I2C **时钟速度**

然后，初始化给定 I2C 端口的配置，请使用端口号和*i2c_config_t* 作为函数调用参数来调用*i2c_param_config()* 函数。

配置示例（主机）：

```
int i2c_master_port = 0;
i2c_config_t conf = {
    .mode = I2C_MODE_MASTER,
    .sda_io_num = I2C_MASTER_SDA_IO,          // 配置 SDA 的 GPIO
```

(下页继续)

(续上页)

```
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_io_num = I2C_MASTER_SCL_IO,          // 配置 SCL 的 GPIO
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .master.clk_speed = I2C_MASTER_FREQ_HZ,  // 为项目选择频率
    .clk_flags = 0,          // 可选项，可以使用 I2C_SCLK_SRC_FLAG_* 标志来选择␣
↪I2C 源时钟
};
```

在此阶段，*i2c_param_config()* 还将其他 I2C 配置参数设置为 I2C 总线协议规范中定义的默认值。有关默认值及修改默认值的详细信息，请参考用户自定义配置。

源时钟配置 增加了 **时钟源分配器**，用于支持不同的时钟源。时钟分配器将选择一个满足所有频率和能力要求的时钟源（如*i2c_config_t::clk_flags* 中的要求）。

当*i2c_config_t::clk_flags* 为 0 时，时钟分配器将仅根据所需频率进行选择。如果不需要诸如 APB 之类的特殊功能，则可以将时钟分配器配置为仅根据所需频率选择源时钟。为此，请将*i2c_config_t::clk_flags* 设置为 0。有关时钟特性，请参见下表。

备注： 如果时钟不满足请求的功能，则该时钟不是有效的选项，即，请求的功能中的任何位（clk_flags）在时钟的功能中均为 0。

对*i2c_config_t::clk_flags* 的解释如下：

1. *I2C_SCLK_SRC_FLAG_AWARE_DFS*：当 APB 时钟改变时，时钟的波特率不会改变。
2. *I2C_SCLK_SRC_FLAG_LIGHT_SLEEP*：支持轻度睡眠模式，APB 时钟则不支持。
3. ESP32-C2 可能不支持某些标志，请在使用前阅读技术参考手册。

备注： 在主机模式下，SCL 的时钟频率不应大于上表中提到的 SCL 的最大频率。

备注： SCL 的时钟频率会被上拉电阻和线上电容（或是从机电容）一起影响。因此，用户需要自己选择合适的上拉电阻去保证 SCL 时钟频率是准确的。尽管 I2C 协议推荐上拉电阻值为 1 K 欧姆到 10 K 欧姆，但是需要根据不同的频率需要选择不同的上拉电阻。

通常来说，所选择的频率越高，需要的上拉电阻越小（但是不要小于 1 K 欧姆）。这是因为高电阻会减小电流，这会延长上升时间从而使频率变慢。通常我们推荐的上拉阻值范围为 2 K 欧姆到 5 K 欧姆，但是用户可能也需要根据他们的实际情况做出一些调整。

安装驱动程序 配置好 I2C 驱动程序后，使用以下参数调用函数*i2c_driver_install()* 安装驱动程序：

- 端口号，从*i2c_port_t* 中二选一
- 主机或从机模式，从*i2c_mode_t* 中选择
- 用于分配中断的标志（请参考 esp_hw_support/include/esp_intr_alloc.h 中 ESP_INTR_FLAG_* 值）

主机模式下通信 安装 I2C 驱动程序后，ESP32-C2 即可与其他 I2C 设备通信。

ESP32-C2 的 I2C 控制器在主机模式下负责与 I2C 从机设备建立通信，并发送命令让从机响应，如进行测量并将结果发给主机。

为优化通信流程，驱动程序提供一个名为“命令链接”的容器，该容器应填充一系列命令，然后传递给 I2C 控制器执行。

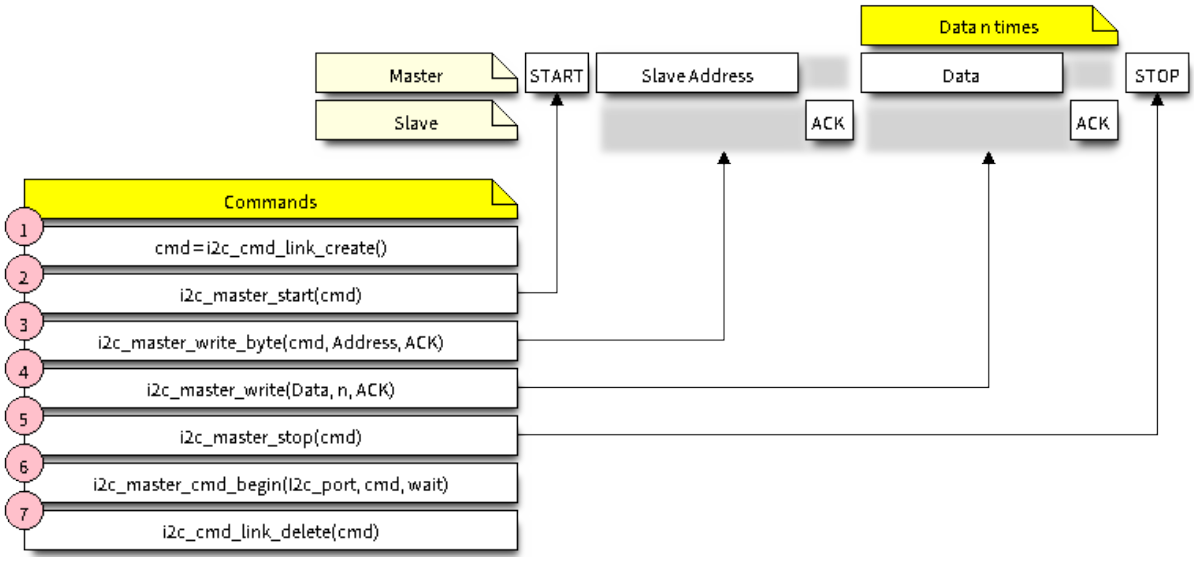


图 3: I2C command link - master write example

主机写入数据 下面的示例展示如何为 I2C 主机构建命令链接，从而向从机发送 *n* 个字节。

下面介绍如何为“主机写入数据”设置命令链接及其内部内容：

1. 使用 `i2c_cmd_link_create()` 创建一个命令链接。
   然后，将一系列待发送给从机的数据填充命令链接：
   a) **启动位** - `i2c_master_start()`
   b) **从机地址** - `i2c_master_write_byte()`。提供单字节地址作为调用此函数的实参。
   c) **数据** - 一个或多个字节的数据作为 `i2c_master_write()` 的实参。
   d) **停止位** - `i2c_master_stop()`
   函数 `i2c_master_write_byte()` 和 `i2c_master_write()` 都有额外的实参，规定主机是否应确认其有无接受到 ACK 位。
2. 通过调用 `i2c_master_cmd_begin()` 来触发 I2C 控制器执行命令链接。一旦开始执行，就不能再修改命令链接。
3. 命令发送后，通过调用 `i2c_cmd_link_delete()` 释放命令链接使用的资源。

主机读取数据 下面的示例展示如何为 I2C 主机构建命令链接，以便从从机读取 *n* 个字节。

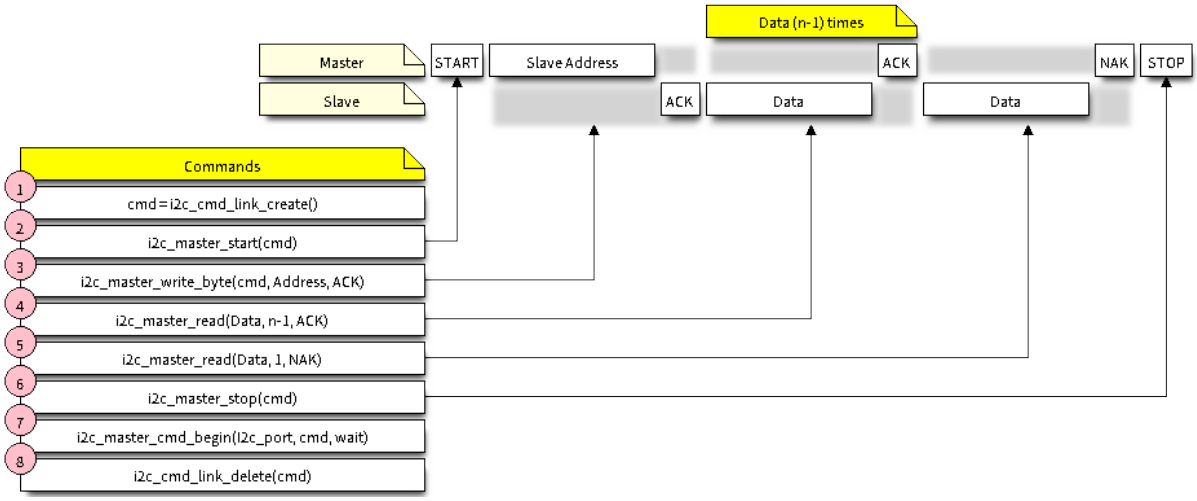


图 4: I2C command link - master read example

在读取数据时，在上图的步骤 4 中，不是用 `i2c_master_write...`，而是

用*i2c_master_read_byte()* 和/或*i2c_master_read()* 填充命令链接。同样，在步骤 5 中配置最后一次的读取，以便主机不提供 ACK 位。

指示写入或读取数据 发送从机地址后（请参考上图中第 3 步），主机可以写入或从从机读取数据。

主机实际执行的操作信息存储在从机地址的最低有效位中。

因此，为了将数据写入从机，主机发送的命令链接应包含地址 `(ESP_SLAVE_ADDR << 1) | I2C_MASTER_WRITE`，如下所示：

```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | I2C_MASTER_WRITE, ACK_EN);
```

同理，指示从从机读取数据的命令链接如下所示：

```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | I2C_MASTER_READ, ACK_EN);
```

中断处理 安装驱动程序时，默认情况下会安装中断处理程序。

用户自定义配置 如本节末尾所述配置驱动程序，函数*i2c_param_config()* 在初始化 I2C 端口的驱动程序配置时，也会将几个 I2C 通信参数设置为 I2C 总线协议规范规定的默认值。其他一些相关参数已在 I2C 控制器的寄存器中预先配置。

通过调用下表中提供的专用函数，可以将所有这些参数更改为用户自定义值。请注意，时序值是在 APB 时钟周期中定义。

表 2: 其他可配置的 I2C 通信参数

要更改的参数	函数
SCL 脉冲周期的高电平和低电平	*i2c_set_period()*
在产生 **启动**信号期间使用的 SCL 和 SDA 信号时序	*i2c_set_start_timing()*
在产生 **停止**信号期间使用的 SCL 和 SDA 信号时序	*i2c_set_stop_timing()*
从机采样以及主机切换时，SCL 和 SDA 信号之间的时序关系	*i2c_set_data_timing()*
I2C 超时	*i2c_set_timeout()*
优先发送/接收最高有效位 (LSB) 或最低有效位 (MSB），可在*i2c_trans_mode_t* 定义的模式中选择	*i2c_set_data_mode()*

上述每个函数都有一个 *_get_* 对应项来检查当前设置的值。例如，调用*i2c_get_timeout()* 来检查 I2C 超时值。

要检查在驱动程序配置过程中设置的参数默认值，请参考文件 driver/i2c/i2c.c 并查找带有后缀 `_DEFAULT` 的定义。

通过函数*i2c_set_pin()* 可以为 SDA 和 SCL 信号选择不同的管脚并改变上拉配置。如果要修改已经输入的值，请使用函数*i2c_param_config()*。

备注： ESP32-C2 的内部上拉电阻范围为几万欧姆，因此在大多数情况下，它们本身不足以用作 I2C 上拉电阻。建议用户使用阻值在 I2C 总线协议规范规定范围内的上拉电阻。计算阻值的具体方法，可参考 TI 应用说明

错误处理 大多数 I2C 驱动程序的函数在成功完成时会返回 `ESP_OK`，或在失败时会返回特定的错误代码。实时检查返回的值并进行错误处理是一种好习惯。驱动程序也会打印日志消息，其中包含错误说明，例如检查输入配置的正确性。有关详细信息，请参考文件 driver/i2c/i2c.c 并用后缀 `_ERR_STR` 查找定义。

使用专用中断来捕获通信故障。例如，如果从机将数据发送回主机耗费太长时间，会触发 `I2C_TIME_OUT_INT` 中断。详细信息请参考中断处理。

如果出现通信失败，可以分别为发送和接收缓存区调用*i2c_reset_tx_fifo()* 和*i2c_reset_rx_fifo()* 来重置内部硬件缓存区。

删除驱动程序 当使用*i2c_driver_install()* 建立 I2C 通信，一段时间后不再需要 I2C 通信时，可以通过调用*i2c_driver_delete()* 来移除驱动程序以释放分配的资源。

由于函数*i2c_driver_delete()* 无法保证线程安全性，请在调用该函数移除驱动程序前务必确保所有的线程都已停止使用驱动程序。

应用示例

I2C 主机和从机示例：peripherals/i2c。

API 参考

Header File

- components/driver/i2c/include/driver/i2c.h

Functions

esp_err_t **i2c_driver_install**(*i2c_port_t* i2c_num, *i2c_mode_t* mode, size_t slv_rx_buf_len, size_t slv_tx_buf_len, int intr_alloc_flags)

Install an I2C driver.

备注: Not all Espressif chips can support slave mode (e.g. ESP32C2)

备注: In master mode, if the cache is likely to be disabled(such as write flash) and the slave is time-sensitive, ESP_INTR_FLAG_IRAM is suggested to be used. In this case, please use the memory allocated from internal RAM in i2c read and write function, because we can not access the psram(if psram is enabled) in interrupt handle function when cache is disabled.

参数

- **i2c_num** –I2C port number
- **mode** –I2C mode (either master or slave).
- **slv_rx_buf_len** –Receiving buffer size. Only slave mode will use this value, it is ignored in master mode.
- **slv_tx_buf_len** –Sending buffer size. Only slave mode will use this value, it is ignored in master mode.
- **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 Parameter error
- ESP_FAIL Driver installation error

esp_err_t **i2c_driver_delete**(*i2c_port_t* i2c_num)

Delete I2C driver.

备注: This function does not guarantee thread safety. Please make sure that no thread will continuously hold semaphores before calling the delete function.

参数 **i2c_num** –I2C port to delete

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_param_config**(*i2c_port_t* i2c_num, const *i2c_config_t* *i2c_conf)

Configure an I2C bus with the given configuration.

参数

- **i2c_num** –I2C port to configure
- **i2c_conf** –Pointer to the I2C configuration

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_reset_tx_fifo**(*i2c_port_t* i2c_num)

reset I2C tx hardware fifo

参数 **i2c_num** –I2C port number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_reset_rx_fifo**(*i2c_port_t* i2c_num)

reset I2C rx fifo

参数 **i2c_num** –I2C port number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_set_pin**(*i2c_port_t* i2c_num, int sda_io_num, int scl_io_num, bool sda_pullup_en, bool scl_pullup_en, *i2c_mode_t* mode)

Configure GPIO pins for I2C SCK and SDA signals.

参数

- **i2c_num** –I2C port number
- **sda_io_num** –GPIO number for I2C SDA signal
- **scl_io_num** –GPIO number for I2C SCL signal
- **sda_pullup_en** –Enable the internal pullup for SDA pin
- **scl_pullup_en** –Enable the internal pullup for SCL pin
- **mode** –I2C mode

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_master_write_to_device**(*i2c_port_t* i2c_num, uint8_t device_address, const uint8_t *write_buffer, size_t write_size, TickType_t ticks_to_wait)

Perform a write to a device connected to a particular I2C port. This function is a wrapper to `i2c_master_start()`, `i2c_master_write()`, `i2c_master_read()`, etc…It shall only be called in I2C master mode.

参数

- **i2c_num** –I2C port number to perform the transfer on
- **device_address** –I2C device’s 7-bit address
- **write_buffer** –Bytes to send on the bus
- **write_size** –Size, in bytes, of the write buffer
- **ticks_to_wait** –Maximum ticks to wait before issuing a timeout.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Sending command error, slave hasn’t ACK the transfer.
- ESP_ERR_INVALID_STATE I2C driver not installed or not in master mode.
- ESP_ERR_TIMEOUT Operation timeout because the bus is busy.

esp_err_t **i2c_master_read_from_device**(*i2c_port_t* i2c_num, uint8_t device_address, uint8_t *read_buffer, size_t read_size, TickType_t ticks_to_wait)

Perform a read to a device connected to a particular I2C port. This function is a wrapper to `i2c_master_start()`, `i2c_master_write()`, `i2c_master_read()`, etc…It shall only be called in I2C master mode.

参数

- **i2c_num** –I2C port number to perform the transfer on
- **device_address** –I2C device’ s 7-bit address
- **read_buffer** –Buffer to store the bytes received on the bus
- **read_size** –Size, in bytes, of the read buffer
- **ticks_to_wait** –Maximum ticks to wait before issuing a timeout.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Sending command error, slave hasn’ t ACK the transfer.
- ESP_ERR_INVALID_STATE I2C driver not installed or not in master mode.
- ESP_ERR_TIMEOUT Operation timeout because the bus is busy.

esp_err_t **i2c_master_write_read_device**(*i2c_port_t* i2c_num, uint8_t device_address, const uint8_t *write_buffer, size_t write_size, uint8_t *read_buffer, size_t read_size, TickType_t ticks_to_wait)

Perform a write followed by a read to a device on the I2C bus. A repeated start signal is used between the `write` and `read`, thus, the bus is not released until the two transactions are finished. This function is a wrapper to `i2c_master_start()`, `i2c_master_write()`, `i2c_master_read()`, etc…It shall only be called in I2C master mode.

参数

- **i2c_num** –I2C port number to perform the transfer on
- **device_address** –I2C device’ s 7-bit address
- **write_buffer** –Bytes to send on the bus
- **write_size** –Size, in bytes, of the write buffer
- **read_buffer** –Buffer to store the bytes received on the bus
- **read_size** –Size, in bytes, of the read buffer
- **ticks_to_wait** –Maximum ticks to wait before issuing a timeout.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Sending command error, slave hasn’ t ACK the transfer.
- ESP_ERR_INVALID_STATE I2C driver not installed or not in master mode.
- ESP_ERR_TIMEOUT Operation timeout because the bus is busy.

i2c_cmd_handle_t **i2c_cmd_link_create_static**(uint8_t *buffer, uint32_t size)

Create and initialize an I2C commands list with a given buffer. All the allocations for data or signals (START, STOP, ACK, …) will be performed within this buffer. This buffer must be valid during the whole transaction. After finishing the I2C transactions, it is required to call `i2c_cmd_link_delete_static()`.

备注: It is **highly** advised to not allocate this buffer on the stack. The size of the data used underneath may increase in the future, resulting in a possible stack overflow as the macro `I2C_LINK_RECOMMENDED_SIZE` would also return a bigger value. A better option is to use a buffer allocated statically or dynamically (with `malloc`).

参数

- **buffer** –Buffer to use for commands allocations
- **size** –Size in bytes of the buffer

返回 Handle to the I2C command link or NULL if the buffer provided is too small, please use `I2C_LINK_RECOMMENDED_SIZE` macro to get the recommended size for the buffer.

i2c_cmd_handle_t **i2c_cmd_link_create**(void)

Create and initialize an I2C commands list with a given buffer. After finishing the I2C transactions, it is required to call `i2c_cmd_link_delete()` to release and return the resources. The required bytes will be dynamically allocated.

返回 Handle to the I2C command link or NULL in case of insufficient dynamic memory.

void **i2c_cmd_link_delete_static**(*i2c_cmd_handle_t* cmd_handle)

Free the I2C commands list allocated statically with `i2c_cmd_link_create_static`.

参数 **cmd_handle** –I2C commands list allocated statically. This handle should be created thanks to `i2c_cmd_link_create_static()` function

void **i2c_cmd_link_delete**(*i2c_cmd_handle_t* cmd_handle)

Free the I2C commands list.

参数 **cmd_handle** –I2C commands list. This handle should be created thanks to `i2c_cmd_link_create()` function

esp_err_t **i2c_master_start**(*i2c_cmd_handle_t* cmd_handle)

Queue a "START signal" to the given commands list. This function shall only be called in I2C master mode. Call `i2c_master_cmd_begin()` to send all the queued commands.

参数 **cmd_handle** –I2C commands list

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create `cmd_handler` is too small
- ESP_FAIL No more memory left on the heap

esp_err_t **i2c_master_write_byte**(*i2c_cmd_handle_t* cmd_handle, uint8_t data, bool ack_en)

Queue a "write byte" command to the commands list. A single byte will be sent on the I2C port. This function shall only be called in I2C master mode. Call `i2c_master_cmd_begin()` to send all queued commands.

参数

- **cmd_handle** –I2C commands list
- **data** –Byte to send on the port
- **ack_en** –Enable ACK signal

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create `cmd_handler` is too small
- ESP_FAIL No more memory left on the heap

esp_err_t **i2c_master_write**(*i2c_cmd_handle_t* cmd_handle, const uint8_t *data, size_t data_len, bool ack_en)

Queue a "write (multiple) bytes" command to the commands list. This function shall only be called in I2C master mode. Call `i2c_master_cmd_begin()` to send all queued commands.

参数

- **cmd_handle** –I2C commands list
- **data** –Bytes to send. This buffer shall remain **valid** until the transaction is finished. If the PSRAM is enabled and `intr_flag` is set to `ESP_INTR_FLAG_IRAM`, `data` should be allocated from internal RAM.
- **data_len** –Length, in bytes, of the data buffer
- **ack_en** –Enable ACK signal

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create `cmd_handler` is too small
- ESP_FAIL No more memory left on the heap

esp_err_t **i2c_master_read_byte**(*i2c_cmd_handle_t* cmd_handle, uint8_t *data, *i2c_ack_type_t* ack)

Queue a "read byte" command to the commands list. A single byte will be read on the I2C bus. This function shall only be called in I2C master mode. Call `i2c_master_cmd_begin()` to send all queued commands.

参数

- **cmd_handle** –I2C commands list
- **data** –Pointer where the received byte will the stored. This buffer shall remain **valid** until the transaction is finished.
- **ack** –ACK signal

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create `cmd_handler` is too small
- ESP_FAIL No more memory left on the heap

esp_err_t **i2c_master_read**(*i2c_cmd_handle_t* cmd_handle, uint8_t *data, size_t data_len, *i2c_ack_type_t* ack)

Queue a "read (multiple) bytes" command to the commands list. Multiple bytes will be read on the I2C bus. This function shall only be called in I2C master mode. Call `i2c_master_cmd_begin()` to send all queued commands.

参数

- **cmd_handle** –I2C commands list
- **data** –Pointer where the received bytes will the stored. This buffer shall remain **valid** until the transaction is finished.
- **data_len** –Size, in bytes, of the `data` buffer
- **ack** –ACK signal

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create `cmd_handler` is too small
- ESP_FAIL No more memory left on the heap

esp_err_t **i2c_master_stop**(*i2c_cmd_handle_t* cmd_handle)

Queue a "STOP signal" to the given commands list. This function shall only be called in I2C master mode. Call `i2c_master_cmd_begin()` to send all the queued commands.

参数 **cmd_handle** –I2C commands list

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create `cmd_handler` is too small
- ESP_FAIL No more memory left on the heap

esp_err_t **i2c_master_cmd_begin**(*i2c_port_t* i2c_num, *i2c_cmd_handle_t* cmd_handle, TickType_t ticks_to_wait)

Send all the queued commands on the I2C bus, in master mode. The task will be blocked until all the commands have been sent out. The I2C port is protected by mutex, so this function is thread-safe. This function shall only be called in I2C master mode.

参数

- **i2c_num** –I2C port number
- **cmd_handle** –I2C commands list
- **ticks_to_wait** –Maximum ticks to wait before issuing a timeout.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Sending command error, slave hasn't ACK the transfer.
- ESP_ERR_INVALID_STATE I2C driver not installed or not in master mode.
- ESP_ERR_TIMEOUT Operation timeout because the bus is busy.

esp_err_t **i2c_set_period**(*i2c_port_t* i2c_num, int high_period, int low_period)

Set I2C master clock period.

参数

- **i2c_num** –I2C port number
- **high_period** –Clock cycle number during SCL is high level, high_period is a 14 bit value
- **low_period** –Clock cycle number during SCL is low level, low_period is a 14 bit value

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_get_period**(*i2c_port_t* i2c_num, int *high_period, int *low_period)

Get I2C master clock period.

参数

- **i2c_num** –I2C port number
- **high_period** –pointer to get clock cycle number during SCL is high level, will get a 14 bit value
- **low_period** –pointer to get clock cycle number during SCL is low level, will get a 14 bit value

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_filter_enable**(*i2c_port_t* i2c_num, uint8_t cyc_num)

Enable hardware filter on I2C bus Sometimes the I2C bus is disturbed by high frequency noise(about 20ns), or the rising edge of the SCL clock is very slow, these may cause the master state machine to break. Enable hardware filter can filter out high frequency interference and make the master more stable.

备注: Enable filter will slow down the SCL clock.

参数

- **i2c_num** –I2C port number to filter
- **cyc_num** –the APB cycles need to be filtered (0<= cyc_num <=7). When the period of a pulse is less than cyc_num * APB_cycle, the I2C controller will ignore this pulse.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_filter_disable**(*i2c_port_t* i2c_num)

Disable filter on I2C bus.

参数 **i2c_num** –I2C port number

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_set_start_timing**(*i2c_port_t* i2c_num, int setup_time, int hold_time)

set I2C master start signal timing

参数

- **i2c_num** –I2C port number
- **setup_time** –clock number between the falling-edge of SDA and rising-edge of SCL for start mark, it’ s a 10-bit value.
- **hold_time** –clock num between the falling-edge of SDA and falling-edge of SCL for start mark, it’ s a 10-bit value.

返回

- ESP_OK Success

- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_get_start_timing**(*i2c_port_t* i2c_num, int *setup_time, int *hold_time)

get I2C master start signal timing

参数

- **i2c_num** –I2C port number
- **setup_time** –pointer to get setup time
- **hold_time** –pointer to get hold time

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_set_stop_timing**(*i2c_port_t* i2c_num, int setup_time, int hold_time)

set I2C master stop signal timing

参数

- **i2c_num** –I2C port number
- **setup_time** –clock num between the rising-edge of SCL and the rising-edge of SDA, it’s a 10-bit value.
- **hold_time** –clock number after the STOP bit’s rising-edge, it’s a 14-bit value.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_get_stop_timing**(*i2c_port_t* i2c_num, int *setup_time, int *hold_time)

get I2C master stop signal timing

参数

- **i2c_num** –I2C port number
- **setup_time** –pointer to get setup time.
- **hold_time** –pointer to get hold time.

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_set_data_timing**(*i2c_port_t* i2c_num, int sample_time, int hold_time)

set I2C data signal timing

参数

- **i2c_num** –I2C port number
- **sample_time** –clock number I2C used to sample data on SDA after the rising-edge of SCL, it’s a 10-bit value
- **hold_time** –clock number I2C used to hold the data after the falling-edge of SCL, it’s a 10-bit value

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_get_data_timing**(*i2c_port_t* i2c_num, int *sample_time, int *hold_time)

get I2C data signal timing

参数

- **i2c_num** –I2C port number
- **sample_time** –pointer to get sample time
- **hold_time** –pointer to get hold time

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_set_timeout**(*i2c_port_t* i2c_num, int timeout)

set I2C timeout value

参数

- **i2c_num** –I2C port number
- **timeout** –timeout value for I2C bus (unit: APB 80Mhz clock cycle)

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_get_timeout**(*i2c_port_t* i2c_num, int *timeout)

get I2C timeout value

参数

- **i2c_num** –I2C port number
- **timeout** –pointer to get timeout value

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_set_data_mode**(*i2c_port_t* i2c_num, *i2c_trans_mode_t* tx_trans_mode, *i2c_trans_mode_t* rx_trans_mode)

set I2C data transfer mode

参数

- **i2c_num** –I2C port number
- **tx_trans_mode** –I2C sending data mode
- **rx_trans_mode** –I2C receving data mode

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2c_get_data_mode**(*i2c_port_t* i2c_num, *i2c_trans_mode_t* *tx_trans_mode, *i2c_trans_mode_t* *rx_trans_mode)

get I2C data transfer mode

参数

- **i2c_num** –I2C port number
- **tx_trans_mode** –pointer to get I2C sending data mode
- **rx_trans_mode** –pointer to get I2C receiving data mode

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Structures

struct **i2c_config_t**

I2C initialization parameters.

Public Members

i2c_mode_t **mode**

I2C mode

int **sda_io_num**

GPIO number for I2C sda signal

int **scl_io_num**

GPIO number for I2C scl signal

bool **sda_pullup_en**

Internal GPIO pull mode for I2C sda signal

bool **scl_pullup_en**

Internal GPIO pull mode for I2C scl signal

uint32_t **clk_speed**

I2C clock frequency for master mode, (no higher than 1MHz for now)

struct *i2c_config_t*::[anonymous]::[anonymous] **master**

I2C master config

uint32_t **clk_flags**

Bitwise of `I2C_SCLK_SRC_FLAG_**FOR_DFS**` for clk source choice

Macros

I2C_SCLK_SRC_FLAG_FOR_NOMAL

Any one clock source that is available for the specified frequency may be choosen

I2C_SCLK_SRC_FLAG_AWARE_DFS

For REF tick clock, it won't change with APB.

I2C_SCLK_SRC_FLAG_LIGHT_SLEEP

For light sleep mode.

I2C_INTERNAL_STRUCT_SIZE

Minimum size, in bytes, of the internal private structure used to describe I2C commands link.

I2C_LINK_RECOMMENDED_SIZE(TRANSACTIONS)

The following macro is used to determine the recommended size of the buffer to pass to `i2c_cmd_link_create_static()` function. It requires one parameter, `TRANSACTIONS`, describing the number of transactions intended to be performed on the I2C port. For example, if one wants to perform a read on an I2C device register, `TRANSACTIONS` must be at least 2, because the commands required are the following:

- write device register
- read register content

Signals such as "(repeated) start", "stop", "nack", "ack" shall not be counted.

Type Definitions

typedef void ***i2c_cmd_handle_t**

I2C command handle

Header File

- components/hal/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 devider

uint16_t **scl_low**

I2C scl low period

uint16_t **scl_high**

I2C scl hight 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

struct **i2c_hal_timing_config_t**

Timing configuration structure. Used for I2C reset internally.

Public Members

int **high_period**

high_period time

int **low_period**

low_period time

int **wait_high_period**

wait_high_period time

int **rstart_setup**
: restart setup

int **start_hold**
: start hold time

int **stop_setup**
: stop setup

int **stop_hold**
: stop hold time

int **sda_sample**
: high_period time

int **sda_hold**
: sda hold time

int **timeout**
: timeout value

Type Definitions

typedef *soc_periph_i2c_clk_src_t* **i2c_clock_source_t**
: I2C group clock source.

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_MAX**
: I2C port max

enum **i2c_mode_t**

Values:

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**

2.6.8 LCD

Introduction

ESP chips can generate various kinds of timings that needed by common LCDs on the market, like SPI LCD, I80 LCD (a.k.a Intel 8080 parallel LCD), RGB/SRGB LCD, I2C LCD, etc. The `esp_lcd` component is officially to support those LCDs with a group of universal APIs across chips.

Functional Overview

In esp_lcd, an LCD panel is represented by *esp_lcd_panel_handle_t*, which plays the role of an **abstract frame buffer**, regardless of the frame memory is allocated inside ESP chip or in external LCD controller. Based on the location of the frame buffer and the hardware connection interface, the LCD panel drivers are mainly grouped into the following categories:

- Controller based LCD driver involves multiple steps to get a panel handle, like bus allocation, IO device registration and controller driver install. The frame buffer is located in the controller's internal GRAM (Graphical RAM). ESP-IDF provides only a limited number of LCD controller drivers out of the box (e.g. ST7789, SSD1306), *More Controller Based LCD Drivers* are maintained in the *Espressif Component Registry <https://components.espressif.com/>_*.
- *SPI Interfaced LCD* describes the steps to install the SPI LCD IO driver and then get the panel handle.
- *I2C Interfaced LCD* describes the steps to install the I2C LCD IO driver and then get the panel handle.
- *LCD Panel IO Operations* - provides a set of APIs to operate the LCD panel, like turning on/off the display, setting the orientation, etc. These operations are common for either controller-based LCD panel driver or RGB LCD panel driver.

SPI Interfaced LCD

1. Create an SPI bus. Please refer to *SPI Master API doc* for more details.

```
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, // Quad SPI LCD driver is not yet supported
    .quadhd_io_num = -1, // Quad SPI LCD driver is not yet supported
    .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 will use 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 will use 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 will use 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 = {
```

(下页继续)

(续上页)

```
    .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_endian* sets the endian of the RGB color data.
    - *esp_lcd_panel_dev_config_t::bits_per_pixel* sets the bit width of the pixel color data. The LCD driver will use 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 = {
    .reset_gpio_num = EXAMPLE_PIN_NUM_RST,
    .rgb_endian = LCD_RGB_ENDIAN_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));
```

I2C Interfaced LCD

1. Create I2C bus. Please refer to *I2C API doc* for more details.

```
i2c_config_t i2c_conf = {
    .mode = I2C_MODE_MASTER, // I2C LCD is a master node
    .sda_io_num = EXAMPLE_PIN_NUM_SDA,
    .scl_io_num = EXAMPLE_PIN_NUM_SCL,
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .master.clk_speed = EXAMPLE_LCD_PIXEL_CLOCK_HZ,
};
ESP_ERROR_CHECK(i2c_param_config(I2C_HOST, &i2c_conf));
ESP_ERROR_CHECK(i2c_driver_install(I2C_HOST, I2C_MODE_MASTER, 0, 0,␣
↪0));
```

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 will use this address to communicate with the LCD controller chip.
    - *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,
    .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((esp_lcd_i2c_bus_handle_t)I2C_
↪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 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 will use 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));
```

More Controller Based LCD Drivers

More LCD panel drivers and touch drivers are available in IDF Component Registry. The list of available and planned drivers with links is in this table.

LCD Panel IO Operations

- *esp_lcd_panel_reset()* can reset the LCD panel.
- Use *esp_lcd_panel_swap_xy()* and *esp_lcd_panel_mirror()*, you can rotate the LCD screen.
- *esp_lcd_panel_disp_on_off()* can turn on or off the LCD screen (different from LCD backlight).
- *esp_lcd_panel_draw_bitmap()* is the most significant function, that will do the magic to draw the user provided color buffer to the LCD screen, where the draw window is also configurable.

Application Example

LCD examples are located under: peripherals/lcd:

- Universal SPI LCD example with SPI touch - peripherals/lcd/spi_lcd_touch
- Jpeg decoding and LCD display - peripherals/lcd/tjpgd
- I2C interfaced OLED display scrolling text - peripherals/lcd/i2c_oled

API Reference

Header File

- components/hal/include/hal/lcd_types.h

Enumerations

enum **lcd_color_rgb_endian_t**

RGB color endian.

Values:

enumerator **LCD_RGB_ENDIAN_RGB**

RGB data endian: RGB

enumerator **LCD_RGB_ENDIAN_BGR**

RGB data endian: BGR

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_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

Header File

- components/esp_lcd/include/esp_lcd_types.h

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

Header File

- components/esp_lcd/include/esp_lcd_panel_io.h

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

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

esp_err_t **esp_lcd_new_panel_io_i2c**(*esp_lcd_i2c_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.

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

Structures

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_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_low_on_data**

If this flag is enabled, DC line = 0 means transfer data, DC line = 1 means transfer command; vice versa

unsigned int **octal_mode**

transmit with octal mode (8 data lines), this mode is used to simulate Intel 8080 timing

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

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

Type Definitions

typedef void ***esp_lcd_spi_bus_handle_t**

Type of LCD SPI bus handle

typedef void ***esp_lcd_i2c_bus_handle_t**

Type of LCD I2C bus handle

typedef struct esp_lcd_i80_bus_t ***esp_lcd_i80_bus_handle_t**

Type of LCD intel 8080 bus 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

Header File

- components/esp_lcd/include/esp_lcd_panel_ops.h

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 index on x-axis (x_start included)
- **y_start** –**[in]** Start index on y-axis (y_start included)
- **x_end** –**[in]** End index on x-axis (x_end not included)
- **y_end** –**[in]** End index on y-axis (y_end 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

Header File

- components/esp_lcd/include/esp_lcd_panel_rgb.h

Header File

- components/esp_lcd/include/esp_lcd_panel_vendor.h

Functions

esp_err_t **esp_lcd_new_panel_st7789**(const *esp_lcd_panel_io_handle_t* io, const *esp_lcd_panel_dev_config_t* *panel_dev_config, *esp_lcd_panel_handle_t* *ret_panel)

Create LCD panel for model ST7789.

参数

- **io** –**[in]** LCD panel IO handle
- **panel_dev_config** –**[in]** general panel device configuration
- **ret_panel** –**[out]** Returned LCD panel 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_nt35510**(const *esp_lcd_panel_io_handle_t* io, const *esp_lcd_panel_dev_config_t* *panel_dev_config, *esp_lcd_panel_handle_t* *ret_panel)

Create LCD panel for model NT35510.

参数

- **io** –**[in]** LCD panel IO handle
- **panel_dev_config** –**[in]** general panel device configuration
- **ret_panel** –**[out]** Returned LCD panel 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_ssd1306**(const *esp_lcd_panel_io_handle_t* io, const *esp_lcd_panel_dev_config_t* *panel_dev_config, *esp_lcd_panel_handle_t* *ret_panel)

Create LCD panel for model SSD1306.

参数

- **io** –**[in]** LCD panel IO handle
- **panel_dev_config** –**[in]** general panel device configuration
- **ret_panel** –**[out]** Returned LCD panel 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_dev_config_t**

Configuration structure for panel device.

Public Members

int **reset_gpio_num**

GPIO used to reset the LCD panel, set to -1 if it’ s not used

lcd_color_rgb_endian_t **color_space**

Deprecated:

Set RGB color space, please use rgb_endian instead

lcd_color_rgb_endian_t **rgb_endian**

Set RGB data endian: RGB or BGR

unsigned int **bits_per_pixel**

Color depth, in bpp

unsigned int **reset_active_high**

Setting this if the panel reset is high level active

struct *esp_lcd_panel_dev_config_t*::[anonymous] **flags**

LCD panel config flags

void ***vendor_config**

vendor specific configuration, optional, left as NULL if not used

2.6.9 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 6 路通道，可以产生独立的波形来驱动 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-C2 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。

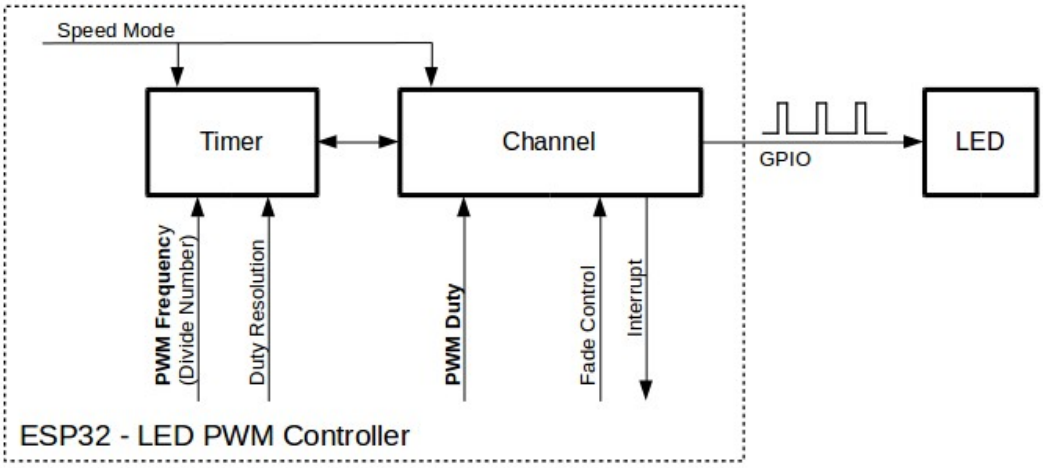


图 5: LED PWM 控制器 API 的关键配置

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
- 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 3: ESP32-C2 LEDC 时钟源特性

时钟名称	时钟频率	时钟功能
PLL_60M_CLK	60 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-C2 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-C2 的所有定时器共用一个时钟源。因此 ESP32-C2 不支持给不同的定时器配置不同的时钟源。

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-C2 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) - 1`。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 1023。此时分辨率为 ~0.1%。

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()*

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-C2 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。

API 参考

Header File

- components/driver/ledc/include/driver/ledc.h

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 resolution.

参数 **ledc_conf** –Pointer of LEDC channel configure struct

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

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_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

---

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.

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) - 1]
- **hpoint** –Set the LEDC hpoint value(max: 0xfffff)

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) - 1]

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) - 1]
- **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 Function pointer error.

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 of duty values is [0, (2**duty_resolution)]
- **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) - 1]
- **scale** –Controls the increase or decrease step scale.
- **cycle_num** –increase or decrease the duty every cycle_num cycles

返回

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

- ESP_ERR_INVALID_STATE Fade function not installed.
- 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) - 1]
- **max_fade_time_ms** –The maximum time of the fading (ms).

返回

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- 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_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 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 garanteed 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 reachs 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 Fade function not installed.
- ESP_ERR_INVALID_ARG Parameter 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) - 1]

- **hpoint** –Set the LEDC hpoint value(max: 0xfffff)

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) - 1]
- **max_fade_time_ms** –The maximum time of the fading (ms).
- **fade_mode** –choose blocking or non-blocking mode

返回

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- 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) - 1]
- **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_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- 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_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

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 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 max value is 0xfffff

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 timer for ledc_timer_config function.

Public Members

ledc_mode_t **speed_mode**

LEDC speed speed_mode, high-speed mode 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.

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) - 1]

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

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

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_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_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**

2.6.10 SD SPI Host Driver

Overview

The SD SPI host driver allows communicating with one or more SD cards by the SPI Master driver which makes use of the SPI host. Each card is accessed through an SD SPI device represented by an *sdspi_dev_handle_t* spi_handle returned when attaching the device to an SPI bus by calling *sdspi_host_init_device*. The bus should be already initialized before (by *spi_bus_initialize*).

With the help of *SPI Master driver* based on, the SPI bus can be shared among SD cards and other SPI devices. The SPI Master driver will handle exclusive access from different tasks.

The SD SPI driver uses software-controlled CS signal.

How to Use

Firstly, use the macro `SDSPI_DEVICE_CONFIG_DEFAULT` to initialize a structure `sdspi_device_config_t`, which is used to initialize an SD SPI device. This macro will also fill in the default pin mappings, which is same as the pin mappings of SDMMC host driver. Modify the host and pins of the structure to desired value. Then call *sdspi_host_init_device* to initialize the SD SPI device and attach to its bus.

Then use `SDSPI_HOST_DEFAULT` macro to initialize a `sdmmc_host_t` structure, which is used to store the state and configurations of upper layer (SD/SDIO/MMC driver). Modify the *slot* parameter of the structure to the SD SPI device spi_handle just returned from *sdspi_host_init_device*. Call *sdmmc_card_init* with the `sdmmc_host_t` to probe and initialize the SD card.

Now you can use SD/SDIO/MMC driver functions to access your card!

Other Details

Only the following driver's API functions are normally used by most applications:

- `sdspi_host_init()`
- `sdspi_host_init_device()`
- `sdspi_host_remove_device()`
- `sdspi_host_deinit()`

Other functions are mostly used by the protocol level SD/SDIO/MMC driver via function pointers in the `sdmmc_host_t` structure. For more details, see *the SD/SDIO/MMC Driver*.

备注: SD over SPI does not support speeds above `SDMMC_FREQ_DEFAULT` due to the limitations of the SPI driver.

警告: If you want to share the SPI bus among SD card and other SPI devices, there are some restrictions, see *Sharing the SPI bus among SD card and other SPI devices*.

Related Docs

Sharing the SPI bus among SD card and other SPI devices The SD card has a SPI mode, which allows it to be communicated to as a SPI device. But there are some restrictions that we need to pay attention to.

Pin loading of other devices When adding more devices onto the same bus, the overall pin loading increases. The loading consists of AC loading (pin capacitor) and DC loading (pull-ups).

AC loading SD cards, which are designed for high-speed communications, have small pin capacitors (AC loading) to work until 50MHz. However, the other attached devices will increase the pin’ s AC loading.

Heavy AC loading of a pin may prevent the pin from being toggled quickly. By using an oscilloscope, you will see the edges of the pin become smoother and not ideal any more (the gradient of the edge is smaller). The setup timing requirements of an SD card may be violoated when the card is connected to such bus. Even worse, the clock from the host may not be recognized by the SD card and other SPI devices on the same bus.

This issue may be more obvious if other attached devices are not designed to work at the same frequency as the SD card, because they may have larger pin capacitors.

To see if your pin AC loading is too heavy, you can try the following tests:

(Terminology: **launch edge**: at which clock edge the data start to toggle; **latch edge**: at which clock edge the data is supposed to be sampled by the receiver, for SD cad, it’ s the rising edge.)

1. Use an oscilloscope to see the clock and compare the data line to the clock. - If you see the clock is not fast enough (for example, the rising/falling edge is longer than 1/4 of the clock cycle), it means the clock is skewed too much. - If you see the data line unstable before the latch edge of the clock, it means the load of the data line is too large.
   You may also observed the corresponding phenomenon (data delayed largely from launching edge of clock) with logic analyzers. But it’ s not as obvious as with an oscilloscope.
2. Try to use slower clock frequency.
   If the lower frequency can work while the higher frequency can’ t, it’ s an indication of the AC loading on the pins is too large.

If the AC loading of the pins is too large, you can either use other faster devices (with lower pin load) or slow down the clock speed.

DC loading The pull-ups required by SD cards are usually around 10 kOhm to 50 kOhm, which may be too strong for some other SPI devices.

Check the specification of your device about its DC output current , it should be larger than 700uA, otherwise the device output may not be read correctly.

Initialization sequence

备注: If you see any problem in the following steps, please make sure the timing is correct first. You can try to slow down the clock speed (SDMMC_FREQ_PROBING = 400 KHz for SD card) to avoid the influence of pin AC loading (see above section).

When using ab SD card with other SPI devices on the same SPI bus, due to the restrictions of the SD card startup flow, the following initialization sequence should be followed: (See also storage/sd_card)

1. Initialize the SPI bus properly by *spi_bus_initialize*.
2. Tie the CS lines of all other devices than the SD card to high. This is to avoid conflicts to the SD card in the following step.
   You can do this by either:
   1. Attach devices to the SPI bus by calling *spi_bus_add_device*. This function will initialize the GPIO that is used as CS to the idle level: high.
   2. Initialize GPIO on the CS pin that needs to be tied up before actually adding a new device.
   3. Rely on the internal/external pull-up (not recommended) to pull-up all the CS pins when the GPIOs of ESP are not initialized yet. You need to check carefull the pull-up is strong enough and there are no other pull-downs that will influence the pull-up (For example, internal pull-down should be enabled).
3. Mount the card to the filesystem by calling *esp_vfs_fat_sdspi_mount*.
   This step will put the SD card into the SPI mode, which SHOULD be done before all other SPI communications on the same bus. Otherwise the card will stay in the SD mode, in which mode it may randomly respond to any SPI communications on the bus, even when its CS line is not addressed.

If you want to test this behavior, please also note that, once the card is put into SPI mode, it will not return to SD mode before next power cycle, i.e. powered down and powered up again.

4. Now you can talk to other SPI devices freely!

API Reference

Header File

- components/driver/spi/include/driver/sdspi_host.h

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

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.

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_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.

2.6.11 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，您仍可以使用 `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）。使用不同的芯片指针，您可以访问连接到 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-C2 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*
- *OPI flash support*
- *32-bit Address Flash Chips*

备注：

- The features listed above needs to be supported by both esp chips and flash chips.
- 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 IDF code has supported the features of those flash chips. It's 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 ESP Chips List:

1. ESP32C3

Flash Chips List:

1. XM25QxxC series.

Flash unique ID 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.

ESP Chips Lists:

ALL

Flash Chips List:

1. ISSI
2. GD
3. TH
4. FM
5. Winbond
6. XMC
7. BOYA

High performance mode

备注： This section is provided for Dual mode (DOUT/DIO) and Quad mode (QIO/QOUT) flash chips. Octal flash used on ESP-chips support High performance mode by default so far, you can refer to the octal flash support list below.

High performance mode (HPM) means that the SPI1 and flash chip works under high frequency. Usually, when the operating frequency of the flash is greater than 80MHz, it is considered that the flash works under HPM. As far as we acknowledged, flash chips have more than two different coping strategies when flash work under HPM. For some flash chips, HPM is controlled by high performance flag (HPF) in status register and for some flash chips, HPM is controlled by dummy cycle bit.

For following conditions，IDF start code deals with HPM internally.

ESP Chips List:

1. ESP32S3

Flash Chips (name & ID) List:

1. GD25Q64C (ID: 0xC84017)
2. GD25Q32C (ID: 0xC84016)

注意： It is hard to create several strategies to cover all situations, so all flash chips using HPM need to be supported explicitly. Therefore, if you try to use a flash not listed as supported under high performance mode, it might cause some error. So, when you try to use the flash chip beyond supported list, please test properly.

OPI flash support 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.

ESP Chips List:

1. ESP32S3

Flash Chips List:

1. MX25UM25645G

32-bit Address 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 16 MBytes), flash uses a 32-bits address to address larger memory. Regretfully, 32-bits address chips have vendor-specific commands, so we need to support the chips one by one.

ESP Chips List:

ALL ESP Chips support this.

Flash Chips List:

1. W25Q256
2. GD25Q256

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 。

如果您想使用这些功能，则需保证 ESP32-C2 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。

您也可以自定义 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 will iterate 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 IDF, thus are updated in together with each IDF version. However 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 IDF functions, which have slight possibility to change between 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 IDF bug fixes to other chip drivers will not be 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 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 IDF Default Driver List

1. Enable the *CONFIG_SPI_FLASH_OVERRIDE_CHIP_DRIVER_LIST* config option. This will prevent compilation and linking of the Default Chip Driver List (*default_registered_chips*) provided by IDF. Instead, the linker will search 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 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，您需要在 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。您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-C2 上，flash 读取/写入/擦除时，cache 必须被禁用。

当 cache 被禁用时 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。如果您使用本文档中 API 函数，上述限制将自动生效且透明（无需您额外关注），但这些限制可能会影响系统中的其他任务的性能。

为避免意外读取 flash cache，在 flash 操作完成前，所有 CPU 上，所有的非 IRAM 安全的中断都会被禁用。

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 如果您需要在 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*。

在函数或符号未被正确放入 IRAM/DRAM 的情况下，中断处理程序在 flash 操作期间从 flash cache 中读取数据时，会导致程序崩溃。这可能是因为代码未被正确放入 IRAM 而产生非法指令异常，也可能是因为常数未被正确放入 DRAM 而读取到垃圾数据。

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 cache 被禁用时，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 cache 被禁用，将不会发生相应的硬件事件。

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

您可以对 SPI flash 内容进行加密，并在硬件层对其进行透明解密。

请参阅*flash 加密*，查看详细信息。

内存映射 API

ESP32-C2 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-C2 memory-spi 来实现的。但是，由于 ESP32-C2 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。

您甚至可以仅通过 GPIO 来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 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。因此，您需要格外注意这一点。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。
   - 您也可以在 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。
2. 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恐慌处理程序。不过，启用该选项也可以帮助处理应用程序中的意外异常，您可以根据实际情况决定是否需要启用这个选项。
3. 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*），您需要禁用上述两个 cache 以防发生 CPU 间通信。

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 will free 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. See *32-bit Address 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 will print the bad writing.
- *CONFIG_SPI_FLASH_WARN_SETTING_ZERO_TO_ONE*, enabling this option will check if you’ re writing zero to one.
- *CONFIG_SPI_FLASH_DANGEROUS_WRITE*, enabling this option will check 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.

Bugfixes Introduced in ESP-IDF but not in Chip-ROM

- Detected Flash physical size correctly, for larger than 256MBit Flash chips. (Commit ID: b4964279d44f73cce7cfd5cf684567fbdfd6fd9e)
- Fixed issue that only at most 128KB virtual address ranges can be mapped to instructions on Flash.

ESP-IDF 和 Chip-ROM 版本 SPI Flash 驱动对比

请参考*SPI Flash API ESP-IDF version vs Chip-ROM version*.

SPI Flash API 参考

Header File

- components/spi_flash/include/esp_flash_spi_init.h

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

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.

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.
- 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.
- 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.
- 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_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

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

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_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

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.

Flash 加密 API 参考

Header File

- components/bootloader_support/include/esp_flash_encrypt.h

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.

---

esp_err_t **esp_flash_encryption_enable_secure_features**(void)

Set all secure eFuse features related to flash encryption.

返回

- ESP_OK - Successfully

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**

2.6.12 SPI Master Driver

SPI Master driver is a program that controls ESP32-C2's SPI peripherals while they function as masters.

Overview of ESP32-C2's SPI peripherals

ESP32-C2 integrates 2 SPI peripherals.

- SPI0 and SPI1 are used internally to access the ESP32-C2's attached flash memory. Both controllers share the same SPI bus signals, and there is an arbiter to determine which can access the bus.
  Currently, SPI Master driver does not support SPI1 bus.
- SPI2 is a general purpose SPI controller. It has an independent signal bus with the same name. The bus has 6 CS lines to drive up to 6 SPI slaves.

Terminology

The terms used in relation to the SPI master driver are given in the table below.

Term	Definition
Host	The SPI controller peripheral inside ESP32-C2 that initiates SPI transmissions over the bus, and acts as an SPI Master.
Device	SPI slave device. An SPI bus may be connected to one or more Devices. Each Device shares the MOSI, MISO and SCLK signals but is only active on the bus when the Host asserts the Device's individual CS line.
Bus	A signal bus, common to all Devices connected to one Host. In general, a bus includes the following lines: MISO, MOSI, SCLK, one or more CS lines, and, optionally, QUADWP and QUADHD. So Devices are connected to the same lines, with the exception that each Device has its own CS line. Several Devices can also share one CS line if connected in the daisy-chain manner.
MOSI	Master Out, Slave In, a.k.a. D. Data transmission from a Host to Device. Also data0 signal in Octal/OPI mode.
MISO	Master In, Slave Out, a.k.a. Q. Data transmission from a Device to Host. Also data1 signal in Octal/OPI mode.
SCLK	Serial Clock. Oscillating signal generated by a Host that keeps the transmission of data bits in sync.
CS	Chip Select. Allows a Host to select individual Device(s) connected to the bus in order to send or receive data.
QUADWP	Write Protect signal. Used for 4-bit (qio/qout) transactions. Also for data2 signal in Octal/OPI mode.
QUADHD	Hold signal. Used for 4-bit (qio/qout) transactions. Also for data3 signal in Octal/OPI mode.
DATA4	Data4 signal in Octal/OPI mode.
DATA5	Data5 signal in Octal/OPI mode.
DATA6	Data6 signal in Octal/OPI mode.
DATA7	Data7 signal in Octal/OPI mode.
Assertion	The action of activating a line.
Deassertion	The action of returning the line back to inactive (back to idle) status.
Transaction	One instance of a Host asserting a CS line, transferring data to and from a Device, and de-asserting the CS line. Transactions are atomic, which means they can never be interrupted by another transaction.
Launch edge	Edge of the clock at which the source register *launches* the signal onto the line.
Latch edge	Edge of the clock at which the destination register *latches in* the signal.

Driver Features

The SPI master driver governs communications of Hosts with Devices. The driver supports the following features:

- Multi-threaded environments
- Transparent handling of DMA transfers while reading and writing data
- Automatic time-division multiplexing of data coming from different Devices on the same signal bus, see *SPI 总线锁*.

警告： The SPI master driver has the concept of multiple Devices connected to a single bus (sharing a single ESP32-C2 SPI peripheral). As long as each Device is accessed by only one task, the driver is thread safe. However, if multiple tasks try to access the same SPI Device, the driver is **not thread-safe**. In this case, it is recommended to either:

- Refactor your application so that each SPI peripheral is only accessed by a single task at a time. You can use `spi_bus_config_t::isr_cpu_id` to register the SPI ISR to the same core as SPI peripheral related tasks to ensure thread safety.
- Add a mutex lock around the shared Device using `xSemaphoreCreateMutex`.

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。

SPI Transactions

An SPI bus transaction consists of five phases which can be found in the table below. Any of these phases can be skipped.

Phase	Description
Command	In this phase, a command (0-16 bit) is written to the bus by the Host.
Address	In this phase, an address (0-32 bit) is transmitted over the bus by the Host.
Dummy	This phase is configurable and is used to meet the timing requirements.
Write	Host sends data to a Device. This data follows the optional command and address phases and is indistinguishable from them at the electrical level.
Read	Device sends data to its Host.

The attributes of a transaction are determined by the bus configuration structure *spi_bus_config_t*, device configuration structure *spi_device_interface_config_t*, and transaction configuration structure *spi_transaction_t*.

An SPI Host can send full-duplex transactions, during which the read and write phases occur simultaneously. The total transaction length is determined by the sum of the following members:

- *spi_device_interface_config_t::command_bits*
- *spi_device_interface_config_t::address_bits*
- *spi_transaction_t::length*

While the member *spi_transaction_t::rxlength* only determines the length of data received into the buffer.

In half-duplex transactions, the read and write phases are not simultaneous (one direction at a time). The lengths of the write and read phases are determined by *spi_transaction_t::length* and *spi_transaction_t::rxlength* respectively.

The command and address phases are optional, as not every SPI device requires a command and/or address. This is reflected in the Device’s configuration: if *spi_device_interface_config_t::command_bits* and/or *spi_device_interface_config_t::address_bits* are set to zero, no command or address phase will occur.

The read and write phases can also be optional, as not every transaction requires both writing and reading data. If *spi_transaction_t::rx_buffer* is NULL and *SPI_TRANS_USE_RXDATA* is not set, the read phase is skipped. If *spi_transaction_t::tx_buffer* is NULL and *SPI_TRANS_USE_TXDATA* is not set, the write phase is skipped.

The driver supports two types of transactions: the interrupt transactions and polling transactions. The programmer can choose to use a different transaction type per Device. If your Device requires both transaction types, see *Notes on Sending Mixed Transactions to the Same Device*.

Interrupt Transactions Interrupt transactions will block the transaction routine until the transaction completes, thus allowing the CPU to run other tasks.

An application task can queue multiple transactions, and the driver will automatically handle them one-by-one in the interrupt service routine (ISR). It allows the task to switch to other procedures until all the transactions complete.

Polling Transactions Polling transactions do not use interrupts. The routine keeps polling the SPI Host' s status bit until the transaction is finished.

All the tasks that use interrupt transactions can be blocked by the queue. At this point, they will need to wait for the ISR to run twice before the transaction is finished. Polling transactions save time otherwise spent on queue handling and context switching, which results in smaller transaction duration. The disadvantage is that the CPU is busy while these transactions are in progress.

The `spi_device_polling_end()` routine needs an overhead of at least 1 us to unblock other tasks when the transaction is finished. It is strongly recommended to wrap a series of polling transactions using the functions `spi_device_acquire_bus()` and `spi_device_release_bus()` to avoid the overhead. For more information, see *Bus Acquiring*.

Transaction Line Mode Supported line modes for ESP32-C2 are listed as follows, to make use of these modes, set the member *flags* in the struct `spi_transaction_t` as shown in the *Transaction Flag* column. If you want to check if corresponding IO pins are set or not, set the member *flags* in the `spi_bus_config_t` as shown in the *Bus IO setting Flag* column.

Mode name	Command Line Width	Address Line Width	Data Line Width	Transaction Flag	Bus IO setting Flag
Normal SPI	1	1	1	0	0
Dual Output	1	1	2	SPI_TRANS_MODE_DIO	SPICOMMON_BUSFLAG_DUAL
Dual I/O	1	2	2	SPI_TRANS_MODE_DIO | SPI_TRANS_MULTILINE_ADDR	
Quad Output	1	1	4	SPI_TRANS_MODE_QIO	SPICOMMON_BUSFLAG_QUAD
Quad I/O	1	4	4	SPI_TRANS_MODE_QIO | SPI_TRANS_MULTILINE_ADDR	

Command and Address Phases During the command and address phases, the members `spi_transaction_t::cmd` and `spi_transaction_t::addr` are sent to the bus, nothing is read at this time. The default lengths of the command and address phases are set in `spi_device_interface_config_t` by calling `spi_bus_add_device()`. If the flags `SPI_TRANS_VARIABLE_CMD` and `SPI_TRANS_VARIABLE_ADDR` in the member `spi_transaction_t::flags` are not set, the driver automatically sets the length of these phases to default values during Device initialization.

If the lengths of the command and address phases need to be variable, declare the struct `spi_transaction_ext_t`, set the flags `SPI_TRANS_VARIABLE_CMD` and/or `SPI_TRANS_VARIABLE_ADDR` in the member `spi_transaction_ext_t::base` and configure the rest of base as usual. Then the length of each phase will be equal to `spi_transaction_ext_t::command_bits` and `spi_transaction_ext_t::address_bits` set in the struct `spi_transaction_ext_t`.

If the command and address phase need to be as the same number of lines as data phase, you need to set *SPI_TRANS_MULTILINE_CMD* and/or *SPI_TRANS_MULTILINE_ADDR* to the *flags* member in the struct `spi_transaction_t`. Also see *Transaction Line Mode*.

Write and Read Phases Normally, the data that needs to be transferred to or from a Device will be read from or written to a chunk of memory indicated by the members `spi_transaction_t::rx_buffer` and `spi_transaction_t::tx_buffer`. If DMA is enabled for transfers, the buffers are required to be:

1. Allocated in DMA-capable internal memory. If *external PSRAM is enabled*, this means using `pvPortMallocCaps(size, MALLOC_CAP_DMA)`.
2. 32-bit aligned (staring from a 32-bit boundary and having a length of multiples of 4 bytes).

If these requirements are not satisfied, the transaction efficiency will be affected due to the allocation and copying of temporary buffers.

If using more than one data lines to transmit, please set *SPI_DEVICE_HALFDUPLEX* flag for the member *flags* in the struct `spi_device_interface_config_t`. And the member *flags* in the struct `spi_transaction_t` should be set as described in *Transaction Line Mode*.

备注： Half-duplex transactions with both read and write phases are not supported. Please use full duplex mode.

Bus Acquiring Sometimes you might want to send SPI transactions exclusively and continuously so that it takes as little time as possible. For this, you can use bus acquiring, which helps to suspend transactions (both polling or interrupt) to other devices until the bus is released. To acquire and release a bus, use the functions `spi_device_acquire_bus()` and `spi_device_release_bus()`.

Driver Usage

- Initialize an SPI bus by calling the function `spi_bus_initialize()`. Make sure to set the correct I/O pins in the struct `spi_bus_config_t`. Set the signals that are not needed to `-1`.
- Register a Device connected to the bus with the driver by calling the function `spi_bus_add_device()`. Make sure to configure any timing requirements the device might need with the parameter `dev_config`. You should now have obtained the Device's handle which will be used when sending a transaction to it.
- To interact with the Device, fill one or more `spi_transaction_t` structs with any transaction parameters required. Then send the structs either using a polling transaction or an interrupt transaction:
  - ***Interrupt*** Either queue all transactions by calling the function `spi_device_queue_trans()` and, at a later time, query the result using the function `spi_device_get_trans_result()`, or handle all requests synchronously by feeding them into `spi_device_transmit()`.
  - ***Polling*** Call the function `spi_device_polling_transmit()` to send polling transactions. Alternatively, if you want to insert something in between, send the transactions by using `spi_device_polling_start()` and `spi_device_polling_end()`.
- (Optional) To perform back-to-back transactions with a Device, call the function `spi_device_acquire_bus()` before sending transactions and `spi_device_release_bus()` after the transactions have been sent.
- (Optional) To unload the driver for a certain Device, call `spi_bus_remove_device()` with the Device handle as an argument.
- (Optional) To remove the driver for a bus, make sure no more drivers are attached and call `spi_bus_free()`.

The example code for the SPI master driver can be found in the peripherals/spi_master directory of ESP-IDF examples.

Transactions with Data Not Exceeding 32 Bits When the transaction data size is equal to or less than 32 bits, it will be sub-optimal to allocate a buffer for the data. The data can be directly stored in the transaction struct instead. For transmitted data, it can be achieved by using the

spi_transaction_t::tx_data member and setting the *SPI_TRANS_USE_TXDATA* flag on the transmission. For received data, use *spi_transaction_t::rx_data* and set *SPI_TRANS_USE_RXDATA*. In both cases, do not touch the *spi_transaction_t::tx_buffer* or *spi_transaction_t::rx_buffer* members, because they use the same memory locations as *spi_transaction_t::tx_data* and *spi_transaction_t::rx_data*.

Transactions with Integers Other Than `uint8_t` An SPI Host reads and writes data into memory byte by byte. By default, data is sent with the most significant bit (MSB) first, as LSB first used in rare cases. If a value less than 8 bits needs to be sent, the bits should be written into memory in the MSB first manner.

For example, if `0b00010` needs to be sent, it should be written into a `uint8_t` variable, and the length for reading should be set to 5 bits. The Device will still receive 8 bits with 3 additional "random" bits, so the reading must be performed correctly.

On top of that, ESP32-C2 is a little-endian chip, which means that the least significant byte of `uint16_t` and `uint32_t` variables is stored at the smallest address. Hence, if `uint16_t` is stored in memory, bits [7:0] are sent first, followed by bits [15:8].

For cases when the data to be transmitted has the size differing from `uint8_t` arrays, the following macros can be used to transform data to the format that can be sent by the SPI driver directly:

- *SPI_SWAP_DATA_TX* for data to be transmitted
- *SPI_SWAP_DATA_RX* for data received

Notes on Sending Mixed Transactions to the Same Device To reduce coding complexity, send only one type of transactions (interrupt or polling) to one Device. However, you still can send both interrupt and polling transactions alternately. The notes below explain how to do this.

The polling transactions should be initiated only after all the polling and interrupt transactions are finished.

Since an unfinished polling transaction blocks other transactions, please do not forget to call the function *spi_device_polling_end()* after *spi_device_polling_start()* to allow other transactions or to allow other Devices to use the bus. Remember that if there is no need to switch to other tasks during your polling transaction, you can initiate a transaction with *spi_device_polling_transmit()* so that it will be ended automatically.

In-flight polling transactions are disturbed by the ISR operation to accommodate interrupt transactions. Always make sure that all the interrupt transactions sent to the ISR are finished before you call *spi_device_polling_start()*. To do that, you can keep calling *spi_device_get_trans_result()* until all the transactions are returned.

To have better control of the calling sequence of functions, send mixed transactions to the same Device only within a single task.

GPIO Matrix and IO_MUX Most of chip's peripheral signals have direct connection to their dedicated IO_MUX pins. However, the signals can also be routed to any other available pins using the less direct GPIO matrix. If at least one signal is routed through the GPIO matrix, then all signals will be routed through it.

When an SPI Host is set to 80MHz or lower frequencies, routing SPI pins via GPIO matrix will behave the same comparing to routing them via IOMUX.

The IO_MUX pins for SPI buses are given below.

Pin Name	GPIO Number (SPI2)
CS0[1]	10
SCLK	6
MISO	2
MOSI	7
QUADWP	5
QUADHD	4

Transfer Speed Considerations

There are three factors limiting the transfer speed:

- Transaction interval
- SPI clock frequency
- Cache miss of SPI functions, including callbacks

The main parameter that determines the transfer speed for large transactions is clock frequency. For multiple small transactions, the transfer speed is mostly determined by the length of transaction intervals.

Transaction Duration Transaction duration includes setting up SPI peripheral registers, copying data to FIFOs or setting up DMA links, and the time for SPI transaction.

Interrupt transactions allow appending extra overhead to accommodate the cost of FreeRTOS queues and the time needed for switching between tasks and the ISR.

For **interrupt transactions**, the CPU can switch to other tasks when a transaction is in progress. This saves the CPU time but increases the transaction duration. See *Interrupt Transactions*. For **polling transactions**, it does not block the task but allows to do polling when the transaction is in progress. For more information, see *Polling Transactions*.

If DMA is enabled, setting up the linked list requires about 2 us per transaction. When a master is transferring data, it automatically reads the data from the linked list. If DMA is not enabled, the CPU has to write and read each byte from the FIFO by itself. Usually, this is faster than 2 us, but the transaction length is limited to 64 bytes for both write and read.

Typical transaction duration for one byte of data are given below.

- Interrupt Transaction via DMA: 42 µs.
- Interrupt Transaction via CPU: 40 µs.
- Polling Transaction via DMA: 17 µs.
- Polling Transaction via CPU: 15 µs.

Note that these data are tested with *CONFIG_SPI_MASTER_ISR_IN_IRAM* enabled. SPI transaction related code are placed in the internal memory. If this option is turned off (for example, for internal memory optimization), the transaction duration may be affected.

SPI Clock Frequency Clock source of the GPSPI peripherals can be selected by setting `spi_device_handle_t::cfg::clock_source`. You can refer to *`spi_clock_source_t`* to know the supported clock sources. By default driver will set `spi_device_handle_t::cfg::clock_source` to *SPI_CLK_SRC_DEFAULT*. This usually stands for the highest frequency among GPSPI clock sources. Its value will be different among chips.

Actual clock frequency of a device may not be exactly equal to the number you set, it will be re-calculated by the driver to the nearest hardware compatible number, and not larger than the clock frequency of the clock source. You can call *`spi_device_get_actual_freq()`* to know the actual frequency computed by the driver.

Theoretical maximum transfer speed of Write or Read phase can be calculated according to the table below:

Line Width of Write/Read phase	Speed (Bps)
1-Line	*SPI Frequency / 8*
2-Line	*SPI Frequency / 4*
4-Line	*SPI Frequency / 2*

The transfer speed calculation of other phases(command, address, dummy) are similar.

[1] Only the first Device attached to the bus can use the CS0 pin.

Cache Miss The default config puts only the ISR into the IRAM. Other SPI related functions, including the driver itself and the callback, might suffer from cache misses and will need to wait until the code is read from flash. Select *CONFIG_SPI_MASTER_IN_IRAM* to put the whole SPI driver into IRAM and put the entire callback(s) and its callee functions into IRAM to prevent cache misses.

备注： SPI driver implementation is based on FreeRTOS APIs, to use *CONFIG_SPI_MASTER_IN_IRAM*, you should not enable *CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH*.

For an interrupt transaction, the overall cost is *20+8n/Fspi[MHz]* [us] for n bytes transferred in one transaction. Hence, the transferring speed is: *n/(20+8n/Fspi)*. An example of transferring speed at 8 MHz clock speed is given in the following table.

Frequency (MHz)	Transaction Interval (us)	Transaction Length (bytes)	Total Time (us)	Total Speed (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

When a transaction length is short, the cost of transaction interval is high. If possible, try to squash several short transactions into one transaction to achieve a higher transfer speed.

Please note that the ISR is disabled during flash operation by default. To keep sending transactions during flash operations, enable *CONFIG_SPI_MASTER_ISR_IN_IRAM* and set `ESP_INTR_FLAG_IRAM` in the member `spi_bus_config_t::intr_flags`. In this case, all the transactions queued before starting flash operations will be handled by the ISR in parallel. Also note that the callback of each Device and their callee functions should be in IRAM, or your callback will crash due to cache miss. For more details, see *IRAM 安全中断处理程序*.

Application Example

The code example for using the SPI master half duplex mode to read/write a AT93C46D EEPROM (8-bit mode) can be found in the peripherals/spi_master/hd_eeprom directory of ESP-IDF examples.

API Reference - SPI Common

Header File

- components/hal/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 **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/driver/spi/include/driver/spi_common.h

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, allowing 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* refering 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.

intr_cpu_id_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 Reference - SPI Master

Header File

- components/driver/spi/include/driver/spi_master.h

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.

备注： While in general, speeds up to 80MHz on the dedicated SPI pins and 40MHz on GPIO-matrix-routed pins are supported, full-duplex transfers routed over the GPIO matrix only support speeds up to 26MHz.

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)
- 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 succesfully completed. It will then return the description of the completed transaction so software can inspect the result and e.g. free the memory or re-use 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)
- 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 succesfully 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_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 aplied 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**

Clock speed, divisors of the SPI `clock_source`, in Hz.

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)

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.

2.6.13 SPI 从机驱动程序

SPI 从机驱动程序控制在 ESP32-C2 中作为从机的 SPI 外设。

ESP32-C2 中 SPI 外设概述

ESP32-C2 集成了 1 个通用的 SPI 控制器。该控制器具有与之同名的独立总线信号。

术语

下表为 SPI 主机驱动的相关术语。

术语	定义
主机 (Host)	ESP32-C2 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，建议您重构应用程序，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，您可以选择配置 `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 cpp:func:*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-C2 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	10
SCLK	6
MISO	2
MOSI	7
QUADWP	5
QUADHD	4

速度与时钟

传输事务间隔 ESP32-C2 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。

您也可以配置一个 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/driver/spi/include/driver/spi_slave.h

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.

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_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 succesfully completed. It will then return the description of the completed transaction so software can inspect the result and e.g. free the memory or re-use 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

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 anabled, 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)

Type Definitions

typedef void (***slave_transaction_cb_t**)(*spi_slave_transaction_t* *trans)

2.6.14 SPI Slave Half Duplex

Introduction

The half duplex (HD) mode is a special mode provided by ESP SPI Slave peripheral. Under this mode, the hardware provides more services than the full duplex (FD) mode (the mode for general purpose SPI transactions, see *SPI 从机驱动程序*). These services reduce the CPU load and the response time of SPI Slave, but the communication format is determined by the hardware. The communication format is always half duplex, so comes the name of Half Duplex Mode.

There are several different types of transactions, determined by the *command* phase of the transaction. Each transaction may consist of the following phases: command, address, dummy, data. The command phase is mandatory, while the other fields may be determined by the command field. During the command, address, dummy phases, the bus is always controlled by the master, while the direction of the data phase depends on the command. The data phase can be either an in phase, for the master to write data to the slave; or an out phase, for the master to read data from the slave.

About the details of how master should communicate with the SPI Slave, see *ESP SPI Slave HD (Half Duplex) Mode Protocol*.

By these different transactions, the slave provide these services to the master:

- A DMA channel for the master to write a great amount of data to the slave.
- A DMA channel for the master to read a great amount of data from the slave.
- Several general purpose registers, shard between the master and the slave.
- Several general purpose interrupts, for the master to interrupt the SW of slave.

Terminology

- Transaction
- Channel
- Sending
- Receiving
- Data Descriptor

Driver Feature

- Transaction read/write by master in segments
- Queues for data to send and received

Driver usage

Slave initialization Call `spi_slave_hd_init()` to initialize the SPI bus as well as the peripheral and the driver. The SPI slave will exclusively use the SPI peripheral, pins of the bus before it's deinitialized. Most configurations of the slave should be done as soon as the slave is being initialized.

The `spi_bus_config_t` specifies how the bus should be initialized, while `spi_slave_hd_slot_config_t` specifies how the SPI Slave driver should work.

Deinitialization (optional) Call `spi_slave_hd_deinit()` to uninstall the driver. The resources, including the pins, SPI peripheral, internal memory used by the driver, interrupt sources, will be released by the deinit function.

Send/Receive Data by DMA Channels To send data to the master through the sending DMA channel, the application should properly wrap the data to send by a `spi_slave_hd_data_t` descriptor structure before calling `spi_slave_hd_queue_trans()` with the data descriptor, and the channel argument of `SPI_SLAVE_CHAN_TX`. The pointers to descriptors are stored in the queue, and the data will be send to the master upon master's RDDMA command in the same order they are put into the queue by `spi_slave_hd_queue_trans()`.

The application should check the result of data sending by calling `spi_slave_hd_get_trans_res()` with the channel set as `SPI_SLAVE_CHAN_TX`. This function will block until the transaction with command RDDMA from master successfully completes (or timeout). The `out_trans` argument of the function will output the pointer of the data descriptor which is just finished.

Receiving data from the master through the receiving DMA channel is quite similar. The application calls `spi_slave_hd_queue_trans()` with proper data descriptor and the channel argument of `SPI_SLAVE_CHAN_RX`. And the application calls the `spi_slave_hd_get_trans_res()` later to get the descriptor to the receiving buffer, before it handles the data in the receiving buffer.

备注： This driver itself doesn't have internal buffer for the data to send, or just received. The application should provide data descriptors for the data buffer to send to master, or to receive data from the master.

The application will have to properly keep the data descriptor as well as the buffer it points to, after the descriptor is successfully sent into the driver internal queue by `spi_slave_hd_queue_trans()`, and before returned by

spi_slave_hd_get_trans_res(). During this period, the hardware as well as the driver may read or write to the buffer and the descriptor when required at any time.

Please note that the buffer doesn't have to be fully sent or filled before it's terminated. For example, in the segment transaction mode, the master has to send CMD7 to terminate a WRDMA transaction, or send CMD8 to terminate a RDDMA transaction (in segments), no matter the send (receive) buffer is used up (full) or not.

Using Data Arguments Sometimes you may have initiator (sending data descriptor) and closure (handling returned descriptors) functions in different places. When you get the returned data descriptor in the closure, you may need some extra information when handle the finished data descriptor. For example, you may want to know which round it is for the returned descriptor, when you send the same piece of data for several times.

Set the `arg` member in the data descriptor to an variable indicating the transaction (by force casting), or point it to a a structure which wraps all the information you may need when handling the sending/receiving data. Then you can get what you need in your closure.

Using callbacks

备注： These callbacks are called in the ISR, so that they are fast enough. However, you may need to be very careful to write the code in the ISR. The callback should return as soon as possible. No delay or blocking operations are allowed.

The `spi_slave_hd_intr_config_t` member in the *spi_slave_hd_slot_config_t* configuration structure passed when initialize the SPI Slave HD driver, allows you having callbacks for each events you may concern.

The corresponding interrupt for each callbacks that is not *NULL* will enabled, so that the callbacks can be called immediately when the events happen. You don't need to provide callbacks for the unconcerned events.

The `arg` member in the configuration structure can help you pass some context to the callback, or indicate which SPI Slave instance when you are using the same callbacks for several SPI Slave peripherals. Set the `arg` member to an variable indicating the SPI Slave instance (by force casting), or point it to a context structure. All the callbacks will be called with this `arg` argument you set when the callbacks are initialized.

There are two other arguments: the `event` and the `awoken`. The `event` passes the information of the current event to the callback. The *spi_slave_hd_event_t* type contains the information of the event, for example, event type, the data descriptor just finished (The *data argument* will be very useful in this case!). The `awoken` argument is an output one, telling the ISR there are tasks are awoken after this callback, and the ISR should call *portYIELD_FROM_ISR()* to do task scheduling. Just pass the `awoken` argument to all FreeRTOS APIs which may unblock tasks, and the awoken will be returned to the ISR.

Writing/Reading Shared Registers Call *spi_slave_hd_write_buffer()* to write the shared buffer, and *spi_slave_hd_read_buffer()* to read the shared buffer.

备注： On ESP32-C2, the shared registers are read/written in words by the application, but read/written in bytes by the master. There's no guarantee four continuous bytes read from the master are from the same word written by the slave's application. It's also possible that if the slave reads a word while the master is writing bytes of the word, the slave may get one word with half of them just written by the master, and the other half hasn't been written into.

The master can confirm that the word is not in transition by reading the word twice and comparing the values.

For the slave, it will be more difficult to ensure the word is not in transition because the process of master writing four bytes can be very long (32 SPI clocks). You can put some CRC in the last (largest address) byte of a word so that when the byte is written, the word is sure to be all written.

Due to the conflicts there may be among read/write from SW (worse if there are multi cores) and master, it is suggested that a word is only used in one direction (only written by master or only written by the slave).

Receiving General Purpose Interrupts From the Master When the master sends CMD 0x08, 0x09 or 0x0A, the slave corresponding will be triggered. Currently the CMD8 is permanently used to indicate the termination of RDDMA segments. To receiving general purpose interrupts, register callbacks for CMD 0x09 and 0x0A when the slave is initialized, see *Using callbacks*.

Application Example

The code example for Device/Host communication can be found in the peripherals/spi_slave_hd directory of ESP-IDF examples.

API reference

Header File

- components/driver/spi/include/driver/spi_slave_hd.h

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.

void ***arg**

Extra argument indiciating 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_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)

2.6.15 Temperature Sensor

Introduction

The ESP32-C2 has a built-in sensor used to measure the chip ' s internal temperature. The temperature sensor module contains an 8-bit Sigma-Delta ADC and a DAC to compensate for the temperature offset.

Due to restrictions of hardware, the sensor has predefined measurement ranges with specific measurement errors. See the table below for details.

predefined range (°C)	error (°C)
50 ~ 125	< 3
20 ~ 100	< 2
-10 ~ 80	< 1
-30 ~ 50	< 2
-40 ~ 20	< 3

备注: The temperature sensor is designed primarily to measure the temperature changes inside the chip. The temperature value depends on factors like microcontroller clock frequency or I/O load. Generally, the chip ' s internal temperature might be higher than the ambient temperature.

Functional Overview

- *Resource Allocation* - covers which parameters should be set up to get a temperature sensor handle and how to recycle the resources when temperature sensor finishes working.
- *Enable and Disable Temperature Sensor* - covers how to enable and disable the temperature sensor.
- *Get Temperature Value* - covers how to get the real-time temperature value.
- *Power Management* - covers how temperature sensor is affected when changing power mode (i.e. light sleep).
- *Thread Safety* - covers how to make the driver to be thread safe.

Resource Allocation The ESP32-C2 has just one built-in temperature sensor hardware. The temperature sensor instance is represented by `temperature_sensor_handle_t`, which is also the bond of the context. It would always be the parameter of the temperature APIs with the information of hardware and configurations, so user can just create a pointer of type `temperature_sensor_handle_t` and passing to APIs as needed.

In order to install a built-in temperature sensor instance, the first thing is to evaluate the temperature range in your detection environment (For example: if the testing environment is in a room, the range you evaluate might be 10 °C ~ 30 °C; if the testing in a lamp bulb, the range you evaluate might be 60 °C ~ 110 °C). Based on that, the following configuration structure should be defined in advance: `temperature_sensor_config_t`:

- `range_min`. The minimum value of testing range you have evaluated.
- `range_max`. The maximum value of testing range you have evaluated.

After the ranges are set, the structure could be passed to *temperature_sensor_install()*, which will instantiate the temperature sensor instance and return a handle.

As mentioned above, different measure ranges have different measurement errors. The user doesn't need to care about the measurement error because we have an internal mechanism to choose the minimum error according to the given range.

If the temperature sensor is no longer needed, you need to call *temperature_sensor_uninstall()* to free the temperature sensor resource.

Creating a Temperature Sensor Handle

- Step1: Evaluate the testing range. In this example, the range is 20 °C ~ 50 °C.
- Step2: Configure the range and obtain a handle

```
temperature_sensor_handle_t temp_handle = NULL;
temperature_sensor_config_t temp_sensor = {
    .range_min = 20,
    .range_max = 50,
};
ESP_ERROR_CHECK(temperature_sensor_install(&temp_sensor, &temp_handle));
```

Enable and Disable Temperature Sensor

1. Enable the temperature sensor by calling *temperature_sensor_enable()*. The internal temperature sensor circuit will start to work. The driver state will transit from init to enable.
2. To Disable the temperature sensor, please call *temperature_sensor_disable()*.

Get Temperature Value After the temperature sensor is enabled by *temperature_sensor_enable()*, user can get the current temperature by calling *temperature_sensor_get_celsius()*.

```
// Enable temperature sensor
ESP_ERROR_CHECK(temperature_sensor_enable(temp_handle));
// Get converted sensor data
float tsens_out;
ESP_ERROR_CHECK(temperature_sensor_get_celsius(temp_handle, &tsens_out));
printf("Temperature in %f °C\n", tsens_out);
// Disable the temperature sensor if it's not needed and save the power
ESP_ERROR_CHECK(temperature_sensor_disable(temp_handle));
```

Power Management When power management is enabled (i.e. `CONFIG_PM_ENABLE` is on), temperature sensor will still keep working because it uses XTAL clock (on ESP32-C3) or RTC clock (on ESP32-S2/S3).

Thread Safety In temperature sensor we don't add any protection to keep the thread safe. Because from the common usage, temperature sensor should only be called in one task. If you must use this driver in different tasks, please add extra locks to protect it.

Unexpected Behaviors

1. The value user gets from the chip is usually different from the ambient temperature. It is because the temperature sensor is built inside the chip. To some extent, it measures the temperature of the chip.
2. When installing the temperature sensor, the driver gives a 'the boundary you gave cannot meet the range of internal temperature sensor' error feedback. It is because the built-in temperature sensor has testing limit. The error due to setting *temperature_sensor_config_t*:

(1) Totally out of range, like 200 °C ~ 300 °C.
(2) Cross the boundary of each predefined measurement. like 40 °C ~ 110 °C.

Application Example

- Temperature sensor reading example: peripherals/temperature_sensor/temp_sensor.

API Reference

Header File

- components/driver/temperature_sensor/include/driver/temperature_sensor.h

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).

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.

Macros

TEMPERATURE_SENSOR_CONFIG_DEFAULT(min, max)

temperature_sensor_config_t default constructure

Type Definitions

typedef struct temperature_sensor_obj_t ***temperature_sensor_handle_t**

Type of temperature sensor driver handle.

2.6.16 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-C2 芯片有 2 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。所有控制器都与不同制造商的 UART 设备兼容，并且支持红外数据协会 (IrDA) 定义的标准协议。

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-C2 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-C2 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。

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。

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 4: 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：

- Tx 环形缓冲区的大小
- Rx 环形缓冲区的大小
- 事件队列句柄和大小
- 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，您可连接外部 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 一旦数据被 UART 接收并保存在 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];
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 如果硬件流控被禁用，您可使用函数*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-C2 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。

调用*uart_driver_install()* 函数可以安装驱动程序的内部中断处理程序，用以管理 Tx 和 Rx 环形缓冲区，并提供事件等高级 API 函数（见下文）。

API 提供了一种便利的方法来处理本文所讨论的特定中断，即用专用函数包装中断：

- **事件检测**：*uart_event_type_t* 定义了多个事件，使用 FreeRTOS 队列功能能够将其报告给用户应用程序。您可调用*安装驱动程序* 中的*uart_driver_install()* 函数启用此功能，请参考 peripherals/uart/uart_events 中使用事件检测的示例。
- **达到 FIFO 空间阈值或传输超时**：Tx 和 Rx FIFO 缓冲区在填充特定数量的字符和在发送或接收数据超时的情况下将会触发中断。如要使用此类中断，请执行以下操作：
  - 配置缓冲区长度和超时阈值：在结构体*uart_intr_config_t* 中输入相应阈值并调用*uart_intr_config()*
  - 启用中断：调用函数*uart_enable_tx_intr()* 和*uart_enable_rx_intr()*
  - 禁用中断：调用函数*uart_disable_tx_intr()* 或*uart_disable_rx_intr()*
- **模式检测**：在检测到重复接收/发送同一字符的“模式”时触发中断，请参考示例 peripherals/uart/uart_events。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()*

宏指令 API 还定义了一些宏指令。例如，*UART_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。

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。

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-C2 技术参考手册* > UART 控制器 (UART) > 寄存器摘要 [PDF]。您可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-C2 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-C2 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。

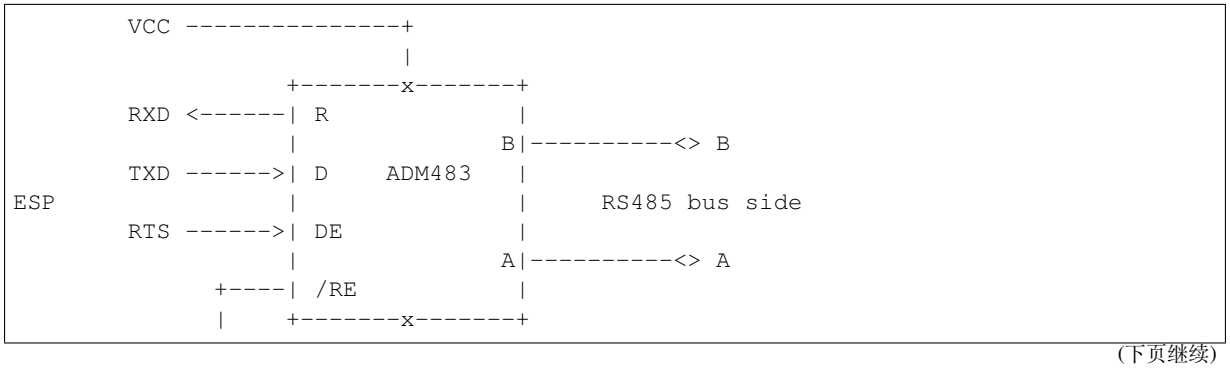
主机开始向 Tx FIFO 缓冲区写入数据时，UART 驱动程序会自动置位 RTS 管脚（逻辑 1）；最后一位数据传输完成后，驱动程序就会取消置位 RTS 管脚（逻辑 0）。要使用此模式，软件必须禁用硬件流控功能。此模式适用于下文所有已用电路。

接口连接选项 本节提供了示例原理图来介绍 ESP32-C2 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



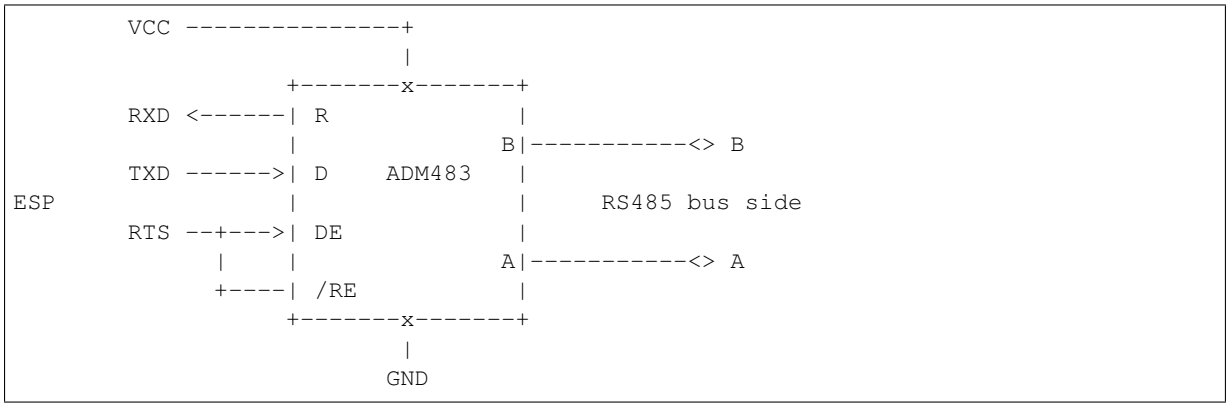
(下页继续)

(续上页)

```
                |             |
               GND           GND
```

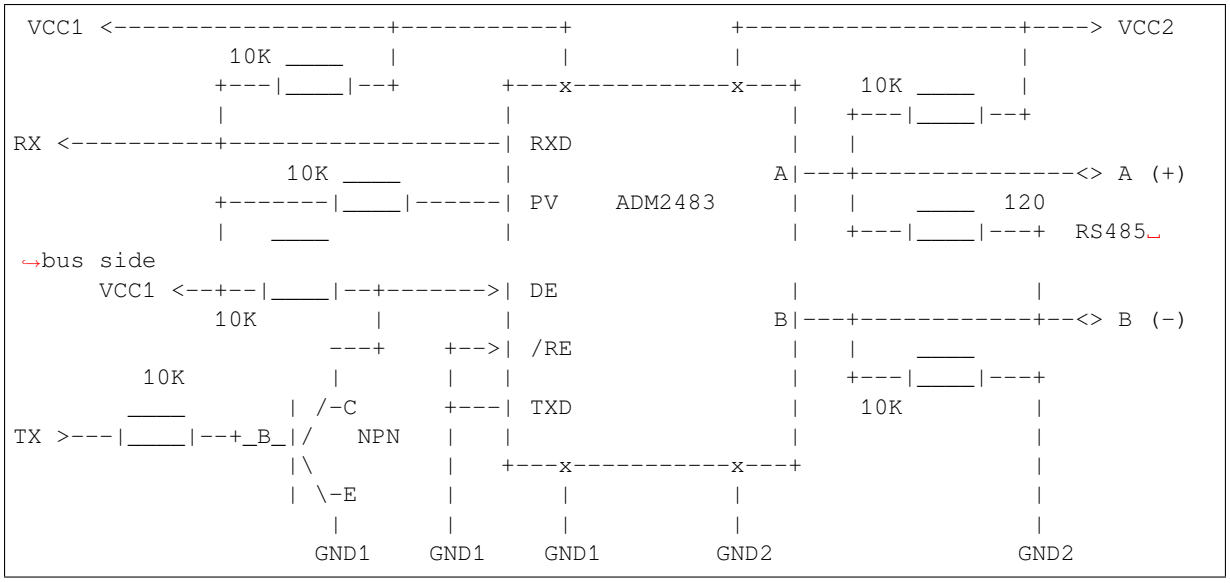
推荐这一电路，因为该电路较为简单，同时能够检测冲突。持续启用线路驱动中的接收器时，UART 将会监控 RS485 总线。启用 UART_RS485_CONF_REG.UART_RS485TX_RX_EN 位时，UART 外设会执行回波抑制。

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



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-C2 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/driver/uart/include/driver/uart.h

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_FIFO_LEN. Tx_buffer_size should be either zero or greater than UART_FIFO_LEN.

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 UART.

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_FIFO_LEN). 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_NUM_0, UART_NUM_1 or UART_NUM_2
- **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_FIFO_LEN

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_NUM_0, UART_NUM_1 or UART_NUM_2
- **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_NUM_0, UART_NUM_1 or UART_NUM_2
- **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 ture 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_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_NUM_0

UART port 0

UART_NUM_1

UART port 1

UART_NUM_MAX

UART port max

UART_PIN_NO_CHANGE

UART_FIFO_LEN

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

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

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

Type Definitions

typedef int **uart_port_t**

UART port number, can be UART_NUM_0 ~ (UART_NUM_MAX -1).

typedef *soc_periph_uart_clk_src_legacy_t* **uart_sclk_t**

UART source clock.

Enumerations

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，您可根据 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/esp32c2/include/soc/uart_channel.h

Macros

UART_GPIO20_DIRECT_CHANNEL

UART_NUM_0_TXD_DIRECT_GPIO_NUM

UART_GPIO19_DIRECT_CHANNEL

UART_NUM_0_RXD_DIRECT_GPIO_NUM

UART_TXD_GPIO20_DIRECT_CHANNEL

UART_RXD_GPIO19_DIRECT_CHANNEL

本部分的 API 示例代码存放在 ESP-IDF 示例项目的 peripherals 目录下。

2.7 项目配置

2.7.1 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。

2.7.2 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。

2.7.3 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。在构建系统中创建 `sdkconfig.defaults` 文件可以避免上述情况发生。该文件可以手动或自动创建，且永远不会被构建系统更改。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`，或者运行 `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。

2.7.4 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 40 个字符。
- 每行最长为 120 个字符。

备注： 菜单中不同配置的 help 小节将被视为 reStructuredText 格式，以便生成相应选项的参考文档。

格式检查器

tools/ci/check_kconfigs.py 可以检查 Kconfig 文件是否符合上述格式规定。检查器会检查 ESP-IDF 目录下的所有 Kconfig 和 Kconfig.projbuild 文件，如有格式错误，会生成后缀为 .new 的新文件以提供修改建议。请注意，检查器不能解决所有格式问题，开发人员仍需终审和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，但无法提供菜单内选项的常用前缀。

2.7.5 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 新版本中被重命名，原有设置将被忽略。以下功能可以避免上述情况发生：

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*。

2.7.6 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) (APP_BUILD_TYPE_APP_2NDBOOT)
- Build app runs entirely in RAM (EXPERIMENTAL) (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*
- *CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION*
- *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*
- *CONFIG_BOOTLOADER_REGION_PROTECTION_ENABLE*
- *CONFIG_BOOTLOADER_FLASH_XMC_SUPPORT*
- *CONFIG_BOOTLOADER_APP_TEST*
- *CONFIG_BOOTLOADER_FACTORY_RESET*
- *CONFIG_BOOTLOADER_HOLD_TIME_GPIO*
- *CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC*
- *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*

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.
- The "None" setting will add the -O0 flag to CFLAGS.

Note that custom optimization levels may be unsupported.

Available options:

- Size (-Os) (BOOTLOADER_COMPILER_OPTIMIZATION_SIZE)
- Debug (-Og) (BOOTLOADER_COMPILER_OPTIMIZATION_DEBUG)
- Optimize for performance (-O2) (BOOTLOADER_COMPILER_OPTIMIZATION_PERF)
- Debug without optimization (-O0) (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 (BOOTLOADER_LOG_LEVEL_NONE)
- Error (BOOTLOADER_LOG_LEVEL_ERROR)
- Warning (BOOTLOADER_LOG_LEVEL_WARN)
- Info (BOOTLOADER_LOG_LEVEL_INFO)
- Debug (BOOTLOADER_LOG_LEVEL_DEBUG)
- Verbose (BOOTLOADER_LOG_LEVEL_VERBOSE)

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 (BOOTLOADER_VDDSDIO_BOOST_1_8V)
- 1.9V (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 (BOOTLOADER_FACTORY_RESET_PIN_LOW)
- Reset on GPIO high (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.

Default value:
- No (disabled) if *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

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 (BOOTLOADER_APP_TEST_PIN_LOW)
- Enter test app on GPIO high (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 rtc_wdt_feed() for resetting counter of rtc_wdt. Use function rtc_wdt_disable() for disabling rtc_wdt.

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 4 if *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*
- from 1 to 16 if *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

Default value:
- 4 if *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*
- 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 implemention 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 SOC_RTC_FAST_MEM_SUPPORTED && ((*CONFIG_SECURE_BOOT* && *CONFIG_SECURE_BOOT_INSECURE*) || *CONFIG_SECURE_BOOT*)

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 bootoloader must be compiled with the same option). The RTC FAST memory has access only through PRO_CPU.

Default value:

- No (disabled) if SOC_RTC_FAST_MEM_SUPPORTED

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*

CONFIG_BOOTLOADER_FLASH_XMC_SUPPORT

Enable the support for flash chips of XMC (READ HELP FIRST)

Found in: Bootloader config

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.

Default value:

- Yes (enabled)

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_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 (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 (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) (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 (SECURE_BOOT_ECDSA_KEY_LEN_192_BITS)
- Using ECC curve NISTP256 (Recommended) (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* && 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 (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 (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 (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 (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. Secure Boot V2: This RSA public key is compiled into the signature block at the end of the bootloader/app.

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* && SOC_SUPPORT_SECURE_BOOT_REVOKE_KEY

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) (SECURE_BOOTLOADER_KEY_ENCODING_256BIT)
- 3/4 encoding (192 bit key) (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 AES-XTS key

Found in: Security features > CONFIG_SECURE_FLASH_ENC_ENABLED

Size of generated AES-XTS 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)) (SECURE_FLASH_ENCRYPTION_AES128_DERIVED)
- AES-128 (256-bit key) (SECURE_FLASH_ENCRYPTION_AES128)
- AES-256 (512-bit key) (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) (SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT)
- Release (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* || 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.

Default value:

- No (disabled) if *CONFIG_SECURE_BOOT_INSECURE* && SOC_EFUSE_REVOKE_BOOT_KEY_DIGESTS

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 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 SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT

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 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_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)) (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)) (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)) (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.

Default value:

- Yes (enabled)

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 this many hex characters from the embedded APP ELF SHA-256 hash value and store it in static RAM. This ensures the app ELF SHA-256 value is always available if it needs to be printed by the panic handler code. Changing this value will change the size of a static buffer, in bytes.

Range:

- from 8 to 64

Default value:

- 16

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 (BOOT_ROM_LOG_ALWAYS_ON)
  Always print ROM logs, this is the default behavior.
- Permanently disable logging (BOOT_ROM_LOG_ALWAYS_OFF)
  Don’ t print ROM logs.
- Log on GPIO High (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 (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

Default value:

- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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 (ESPTOOLPY_FLASHMODE_QIO)
- QOUT (ESPTOOLPY_FLASHMODE_QOUT)
- DIO (ESPTOOLPY_FLASHMODE_DIO)
- DOUT (ESPTOOLPY_FLASHMODE_DOUT)
- OPI (ESPTOOLPY_FLASHMODE_OPI)

CONFIG_ESPTOOLPY_FLASH_SAMPLE_MODE

Flash Sampling Mode

Found in: Serial flasher config

Available options:

- STR Mode (ESPTOOLPY_FLASH_SAMPLE_MODE_STR)
- DTR Mode (ESPTOOLPY_FLASH_SAMPLE_MODE_DTR)

CONFIG_ESPTOOLPY_FLASHFREQ

Flash SPI speed

Found in: Serial flasher config

Available options:

- 120 MHz (ESPTOOLPY_FLASHFREQ_120M)
  - 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 (ESPTOOLPY_FLASHFREQ_80M)
- 64 MHz (ESPTOOLPY_FLASHFREQ_64M)
- 60 MHz (ESPTOOLPY_FLASHFREQ_60M)
- 48 MHz (ESPTOOLPY_FLASHFREQ_48M)
- 40 MHz (ESPTOOLPY_FLASHFREQ_40M)
- 32 MHz (ESPTOOLPY_FLASHFREQ_32M)

- 30 MHz (ESPTOOLPY_FLASHFREQ_30M)
- 26 MHz (ESPTOOLPY_FLASHFREQ_26M)
- 24 MHz (ESPTOOLPY_FLASHFREQ_24M)
- 20 MHz (ESPTOOLPY_FLASHFREQ_20M)
- 16 MHz (ESPTOOLPY_FLASHFREQ_16M)
- 15 MHz (ESPTOOLPY_FLASHFREQ_15M)

CONFIG_ESPTOOLPY_FLASHSIZE

Flash size

Found in: Serial flasher config

SPI flash size, in megabytes

Available options:

- 1 MB (ESPTOOLPY_FLASHSIZE_1MB)
- 2 MB (ESPTOOLPY_FLASHSIZE_2MB)
- 4 MB (ESPTOOLPY_FLASHSIZE_4MB)
- 8 MB (ESPTOOLPY_FLASHSIZE_8MB)
- 16 MB (ESPTOOLPY_FLASHSIZE_16MB)
- 32 MB (ESPTOOLPY_FLASHSIZE_32MB)
- 64 MB (ESPTOOLPY_FLASHSIZE_64MB)
- 128 MB (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.

Default value:

- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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 (ESPTOOLPY_BEFORE_RESET)
- No reset (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 (ESPTOOLPY_AFTER_RESET)
- Stay in bootloader (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 (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 (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 (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 (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 (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 (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 (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.

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_OPTIMIZATION_CHECKS_SILENT*

- *CONFIG_COMPILER_DISABLE_GCC12_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_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 "Default" 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) (COMPILER_OPTIMIZATION_DEFAULT)
- Optimize for size (-Os) (COMPILER_OPTIMIZATION_SIZE)
- Optimize for performance (-O2) (COMPILER_OPTIMIZATION_PERF)
- Debug without optimization (-O0) (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 (COMPILER_OPTIMIZATION_ASSERTIONS_ENABLE)
  Enable assertions. Assertion content and line number will be printed on failure.
- Silent (saves code size) (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) (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 (COMPILER_FLOAT_LIB_FROM_GCCLIB)
- librvfp (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 (COMPILER_STACK_CHECK_MODE_NONE)
- Normal (COMPILER_STACK_CHECK_MODE_NORM)
- Strong (COMPILER_STACK_CHECK_MODE_STRONG)
- Overall (COMPILER_STACK_CHECK_MODE_ALL)

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_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.

Component config

Contains:

- *ADC and ADC Calibration*
- *Application Level Tracing*
- *Bluetooth*
- *Common ESP-related*
- *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-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*
- *High resolution timer (esp_timer)*
- *HTTP Server*
- *IEEE 802.15.4*
- *IPC (Inter-Processor Call)*
- *LCD and Touch Panel*
- *Log output*
- *LWIP*
- *mbedTLS*
- *Newlib*
- *NVS*
- *OpenThread*
- *Partition API Configuration*
- *PHY*
- *Power Management*
- *Protocomm*
- *PThreads*
- *SoC Settings*
- *SPI Flash driver*
- *SPIFFS Configuration*
- *TCP Transport*
- *Ultra Low Power (ULP) Co-processor*
- *Unity unit testing library*
- *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 (APPTRACE_DEST_JTAG)
- None (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 (APPTRACE_DEST_UART0)
- UART1 (APPTRACE_DEST_UART1)
- UART2 (APPTRACE_DEST_UART2)
- USB_CDC (APPTRACE_DEST_USB_CDC)
- None (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 (APPTRACE_SV_DEST_JTAG)
  Send SEGGER SystemView events through JTAG interface.
- Data destination UART (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 (APPTRACE_SV_DEST_CPU_0)
  Send SEGGER SystemView events for Pro CPU.
- CPU1 (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) (APPTRACE_SV_TS_SOURCE_CCOUNT)
- General Purpose Timer (Timer Group) (APPTRACE_SV_TS_SOURCE_GPTIMER)
- esp_timer high resolution timer (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.

Bluetooth Contains:

- *Bluedroid Options*
- *CONFIG_BT_ENABLED*
- *Controller Options*
- *NimBLE Options*

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 (BT_BLUEDROID_ENABLED)
  This option is recommended for classic Bluetooth or for dual-mode usecases
- NimBLE - BLE only (BT_NIMBLE_ENABLED)
  This option is recommended for BLE only usecases to save on memory
- Disabled (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 (BT_CONTROLLER_ENABLED)
  This option is recommended for Bluetooth controller usecases
- Disabled (BT_CONTROLLER_DISABLED)
  This option is recommended for Bluetooth Host only usecases

Bluedroid Options Contains:

- *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_ALLOCATION_FROM_SPIRAM_FIRST*
- *CONFIG_BT_STACK_NO_LOG*
- *CONFIG_BT_BLE_42_FEATURES_SUPPORTED*
- *CONFIG_BT_BLE_50_FEATURES_SUPPORTED*
- *CONFIG_BT_MULTI_CONNECTION_ENBALE*
- *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_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 BT_BLUEDROID_ENABLED && 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) (BT_BLUEDROID_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (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:

- 4096 if BT_BLUEDROID_ENABLED && 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 BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_BLE_ENABLED

Bluetooth Low Energy

Found in: Component config > Bluetooth > Bluedroid Options

This enables Bluetooth Low Energy

Default value:

- Yes (enabled) if BT_BLUEDROID_ENABLED && 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* && 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* && 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* && 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* && BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

Default value:

- 8 if *CONFIG_BT_GATTS_ENABLE* && BT_BLUEDROID_ENABLED && 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* && BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

Default value:
- 100 if *CONFIG_BT_GATTS_ENABLE* && BT_BLUEDROID_ENABLED && 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 (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 (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 enable gatt robust caching feature on server

Default value:
- No (disabled) if *CONFIG_BT_GATTS_ENABLE* && 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* && 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* && 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* && 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* && BT_BLUEDROID_ENABLED

Default value:

- 40 if *CONFIG_BT_GATTC_ENABLE* && 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* && 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 7 if *CONFIG_BT_GATTC_ENABLE* && BT_BLUEDROID_ENABLED

Default value:

- 3 if *CONFIG_BT_GATTC_ENABLE* && 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* && 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* && 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 BT_BLUEDROID_ENABLED && 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 (BT_LOG_HCI_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_HCI_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_HCI_TRACE_LEVEL_WARNING)
- API (BT_LOG_HCI_TRACE_LEVEL_API)
- EVENT (BT_LOG_HCI_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_HCI_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_BTM_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BTM_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BTM_TRACE_LEVEL_WARNING)
- API (BT_LOG_BTM_TRACE_LEVEL_API)
- EVENT (BT_LOG_BTM_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BTM_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_L2CAP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_L2CAP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_L2CAP_TRACE_LEVEL_WARNING)
- API (BT_LOG_L2CAP_TRACE_LEVEL_API)
- EVENT (BT_LOG_L2CAP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_L2CAP_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_RFCOMM_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_RFCOMM_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_RFCOMM_TRACE_LEVEL_WARNING)
- API (BT_LOG_RFCOMM_TRACE_LEVEL_API)
- EVENT (BT_LOG_RFCOMM_TRACE_LEVEL_EVENT)

- DEBUG (BT_LOG_RFCOMM_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_SDP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_SDP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_SDP_TRACE_LEVEL_WARNING)
- API (BT_LOG_SDP_TRACE_LEVEL_API)
- EVENT (BT_LOG_SDP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_SDP_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_GAP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_GAP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_GAP_TRACE_LEVEL_WARNING)
- API (BT_LOG_GAP_TRACE_LEVEL_API)
- EVENT (BT_LOG_GAP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_GAP_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_BNEP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BNEP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BNEP_TRACE_LEVEL_WARNING)
- API (BT_LOG_BNEP_TRACE_LEVEL_API)
- EVENT (BT_LOG_BNEP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BNEP_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_PAN_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_PAN_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_PAN_TRACE_LEVEL_WARNING)
- API (BT_LOG_PAN_TRACE_LEVEL_API)
- EVENT (BT_LOG_PAN_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_PAN_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_A2D_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_A2D_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_A2D_TRACE_LEVEL_WARNING)
- API (BT_LOG_A2D_TRACE_LEVEL_API)
- EVENT (BT_LOG_A2D_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_A2D_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_AVDT_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVDT_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVDT_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVDT_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVDT_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_AVDT_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_AVCT_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVCT_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVCT_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVCT_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVCT_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_AVCT_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_AVRC_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVRC_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVRC_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVRC_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVRC_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_AVRC_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_MCA_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_MCA_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_MCA_TRACE_LEVEL_WARNING)
- API (BT_LOG_MCA_TRACE_LEVEL_API)
- EVENT (BT_LOG_MCA_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_MCA_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_HID_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_HID_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_HID_TRACE_LEVEL_WARNING)
- API (BT_LOG_HID_TRACE_LEVEL_API)
- EVENT (BT_LOG_HID_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_HID_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_APPL_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_APPL_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_APPL_TRACE_LEVEL_WARNING)
- API (BT_LOG_APPL_TRACE_LEVEL_API)

- EVENT (BT_LOG_APPL_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_APPL_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_GATT_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_GATT_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_GATT_TRACE_LEVEL_WARNING)
- API (BT_LOG_GATT_TRACE_LEVEL_API)
- EVENT (BT_LOG_GATT_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_GATT_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_SMP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_SMP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_SMP_TRACE_LEVEL_WARNING)
- API (BT_LOG_SMP_TRACE_LEVEL_API)
- EVENT (BT_LOG_SMP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_SMP_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_BTIF_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BTIF_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BTIF_TRACE_LEVEL_WARNING)
- API (BT_LOG_BTIF_TRACE_LEVEL_API)
- EVENT (BT_LOG_BTIF_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BTIF_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_BTC_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BTC_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BTC_TRACE_LEVEL_WARNING)
- API (BT_LOG_BTC_TRACE_LEVEL_API)
- EVENT (BT_LOG_BTC_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BTC_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_OSI_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_OSI_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_OSI_TRACE_LEVEL_WARNING)
- API (BT_LOG_OSI_TRACE_LEVEL_API)
- EVENT (BT_LOG_OSI_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_OSI_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BT_LOG_BLUFI_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BLUFI_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BLUFI_TRACE_LEVEL_WARNING)
- API (BT_LOG_BLUFI_TRACE_LEVEL_API)
- EVENT (BT_LOG_BLUFI_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BLUFI_TRACE_LEVEL_DEBUG)
- VERBOSE (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 BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

Default value:

- 4 if BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_MULTI_CONNECTION_ENBALE

Enable BLE multi-conections

Found in: Component config > Bluetooth > Bluedroid Options

Enable this option if there are multiple connections

Default value:

- Yes (enabled) if BT_BLUEDROID_ENABLED && 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 BT_BLUEDROID_ENABLED && 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 BT_BLUEDROID_ENABLED && 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 BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

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 BT_BLUEDROID_ENABLED && *CONFIG_BT_BLE_ENABLED* && 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 BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

Default value:
- 30 if BT_BLUEDROID_ENABLED && 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 BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

Default value:
- 32 if BT_BLUEDROID_ENABLED && 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:
- No (disabled) if BT_BLUEDROID_ENABLED && 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 BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

Default value:
- 900 if BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_BLE_50_FEATURES_SUPPORTED

Enable BLE 5.0 features

Found in: Component config > Bluetooth > Bluedroid Options

This enables BLE 5.0 features, this option only support esp32c3/esp32s3 chip

Default value:

- Yes (enabled) if BT_BLUEDROID_ENABLED && 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 BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

NimBLE Options Contains:

- *CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME*
- *CONFIG_BT_NIMBLE_HS_STOP_TIMEOUT_MS*
- *CONFIG_BT_NIMBLE_WHITELIST_SIZE*
- *CONFIG_BT_NIMBLE_BLE_GATT_BLOB_TRANSFER*
- *CONFIG_BT_NIMBLE_COEX_PHY_CODED_TX_RX_TLIM*
- *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT*
- *CONFIG_BT_NIMBLE_ROLE_BROADCASTER*
- *CONFIG_BT_NIMBLE_ROLE_CENTRAL*
- *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_USE_ESP_TIMER*
- *CONFIG_BT_NIMBLE_DEBUG*
- *CONFIG_BT_NIMBLE_HOST_BASED_PRIVACY*
- *CONFIG_BT_NIMBLE_HS_FLOW_CTRL*
- *CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE*
- *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_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 (BT_NIMBLE_MEM_ALLOC_MODE_INTERNAL)
- External SPIRAM (BT_NIMBLE_MEM_ALLOC_MODE_EXTERNAL)
- Default alloc mode (BT_NIMBLE_MEM_ALLOC_MODE_DEFAULT)
- Internal IRAM (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 (BT_NIMBLE_LOG_LEVEL_NONE)
- Error logs (BT_NIMBLE_LOG_LEVEL_ERROR)
- Warning logs (BT_NIMBLE_LOG_LEVEL_WARNING)
- Info logs (BT_NIMBLE_LOG_LEVEL_INFO)
- Debug logs (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 2 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
- from 1 to 9 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Default value:

- 2 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
- 3 if BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Default value:

- 0 if BT_NIMBLE_ENABLED && 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) (BT_NIMBLE_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (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* && BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
- 4096 if BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Contains:

- *CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_ENCRYPTION*
- *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* && 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* && 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* && 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* && BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Memory Settings

Contains:

- *CONFIG_BT_NIMBLE_ACL_BUF_COUNT*
- *CONFIG_BT_NIMBLE_ACL_BUF_SIZE*
- *CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE*
- *CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT*
- *CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_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_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 BT_NIMBLE_ENABLED
- 12 if 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 BT_NIMBLE_ENABLED
- 256 if 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 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 BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_ACL_BUF_COUNT

ACL Buffer count

Found in: Component config > Bluetooth > NimBLE Options > Memory Settings

The number of ACL data buffers.

Default value:

- 24 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_ACL_BUF_SIZE

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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE

HCI 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* && BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
- 70 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT

High Priority HCI 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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT

Low Priority HCI 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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* && 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* && 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* && 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. This is applicable in case of Host based RPA

Range:

- from 1 to 41400 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Default value:

- 900 if BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && 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* && 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* && 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* && 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* && 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* && 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* && 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* && 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* && 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* && 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* && 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* && 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 BT_NIMBLE_ENABLED && 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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HOST_BASED_PRIVACY

Enable host based privacy for random address.

Found in: Component config > Bluetooth > NimBLE Options

Use this option to do host based Random Private Address resolution. If this option is disabled then controller based privacy is used.

Default value:

- No (disabled) if BT_NIMBLE_ENABLED && 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 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 7 if BT_NIMBLE_ENABLED && *CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT* && BT_NIMBLE_ENABLED

Default value:

- 3 if BT_NIMBLE_ENABLED && *CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT* && 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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Contains:

- *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_BLE_POWER_CONTROL*
- *CONFIG_BT_NIMBLE_MAX_PERIODIC_ADVERTISER_LIST*
- *CONFIG_BT_NIMBLE_MAX_PERIODIC_SYNCS*

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* && 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* && 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* && 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* && BT_NIMBLE_ENABLED

Default value:

- 1 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && BT_NIMBLE_ENABLED
- 0 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && 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* && BT_NIMBLE_ENABLED

Default value:

- 1650 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && BT_NIMBLE_ENABLED
- 0 if *CONFIG_BT_NIMBLE_EXT_ADV* && *CONFIG_BT_NIMBLE_EXT_ADV* && 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* && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_PERIODIC_ADV_SYNC_TRANSFER

Enable Transer 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* && 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

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 3 if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED
- from 0 to 8 if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED

Default value:

- 1 if *CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED
- 0 if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && 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

Set this option to set the upper limit for number of periodic advertiser list.

Range:

- from 1 to 5 if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED

Default value:

- 5 if *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && *CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT* && 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 && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_COEX_PHY_CODED_TX_RX_TLIM

Coexistence: limit on MAX Tx/Rx time for coded-PHY connection

Found in: Component config > Bluetooth > NimBLE Options

When using PHY-Coded in BLE connection, limitation on max tx/rx time can be applied to better avoid dramatic performance deterioration of Wi-Fi.

Available options:

- Force Enable (BT_NIMBLE_COEX_PHY_CODED_TX_RX_TLIM_EN)
  Always enable the limitation on max tx/rx time for Coded-PHY connection
- Force Disable (BT_NIMBLE_COEX_PHY_CODED_TX_RX_TLIM_DIS)
  Disable the limitation on max tx/rx time for Coded-PHY connection

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 BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Default value:
- 12 if BT_NIMBLE_ENABLED && 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 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 BT_NIMBLE_ENABLED && 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 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.

Controller Options Contains:

- *CONFIG_BT_LE_LL_DUP_SCAN_LIST_COUNT*
- *CONFIG_BT_LE_LL_RESOLV_LIST_SIZE*
- *CONFIG_BT_LE_LL_SCA*
- *CONFIG_BT_LE_WHITELIST_SIZE*
- *CONFIG_BT_LE_COEX_PHY_CODED_TX_RX_TLIM*
- *CONFIG_BT_LE_CONTROLLER_TASK_STACK_SIZE*
- *CONFIG_BT_LE_50_FEATURE_SUPPORT*
- *CONFIG_BT_LE_SLEEP_ENABLE*
- *CONFIG_BT_LE_SECURITY_ENABLE*
- *HCI Config*
- *CONFIG_BT_LE_MAX_CONNECTIONS*
- *Memory Settings*
- *CONFIG_BT_LE_CRYPTO_STACK_MBEDTLS*
- *CONFIG_BT_LE_USE_ESP_TIMER*

HCI Config Contains:

- *CONFIG_BT_LE_HCI_UART_BAUD*
- *CONFIG_BT_LE_HCI_UART_CTS_PIN*
- *CONFIG_BT_LE_HCI_UART_FLOWCTRL*
- *CONFIG_BT_LE_HCI_UART_PORT*
- *CONFIG_BT_LE_HCI_UART_RTS_PIN*
- *CONFIG_BT_LE_HCI_UART_RX_PIN*
- *CONFIG_BT_LE_HCI_UART_TASK_STACK_SIZE*
- *CONFIG_BT_LE_HCI_UART_TX_PIN*
- *CONFIG_BT_LE_HCI_INTERFACE*
- *CONFIG_BT_LE_HCI_UART_PARITY*

CONFIG_BT_LE_HCI_INTERFACE

Select HCI interface

Found in: Component config > Bluetooth > Controller Options > HCI Config

Available options:

- ram (BT_LE_HCI_INTERFACE_USE_RAM)
  Use RAM as HCI interface
- uart (BT_LE_HCI_INTERFACE_USE_UART)
  Use UART as HCI interface

CONFIG_BT_LE_HCI_UART_PORT

HCI UART port

Found in: Component config > Bluetooth > Controller Options > HCI Config

Set the port number of HCI UART

Default value:

- 1 if BT_LE_HCI_INTERFACE_USE_UART && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_UART_FLOWCTRL

HCI uart Hardware Flow ctrl

Found in: Component config > Bluetooth > Controller Options > HCI Config

Default value:

- No (disabled) if BT_LE_HCI_INTERFACE_USE_UART && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_UART_TX_PIN

HCI uart Tx gpio

Found in: Component config > Bluetooth > Controller Options > HCI Config

Default value:

- 19 if BT_LE_HCI_INTERFACE_USE_UART && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_UART_RX_PIN

HCI uart Rx gpio

Found in: Component config > Bluetooth > Controller Options > HCI Config

Default value:

- 10 if BT_LE_HCI_INTERFACE_USE_UART && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_UART_RTS_PIN

HCI uart RTS gpio

Found in: Component config > Bluetooth > Controller Options > HCI Config

Default value:

- 4 if *CONFIG_BT_LE_HCI_UART_FLOWCTRL* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_UART_CTS_PIN

HCI uart CTS gpio

Found in: Component config > Bluetooth > Controller Options > HCI Config

Default value:

- 5 if *CONFIG_BT_LE_HCI_UART_FLOWCTRL* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_UART_BAUD

HCI uart baudrate

Found in: Component config > Bluetooth > Controller Options > HCI Config

HCI uart baud rate 115200 ~ 1000000

Default value:

- 921600 if BT_LE_HCI_INTERFACE_USE_UART && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_UART_PARITY

select uart parity

Found in: Component config > Bluetooth > Controller Options > HCI Config

Available options:

- PARITY_DISABLE (BT_LE_HCI_UART_UART_PARITY_DISABLE) UART_PARITY_DISABLE
- PARITY_EVEN (BT_LE_HCI_UART_UART_PARITY_EVEN) UART_PARITY_EVEN
- PARITY_ODD (BT_LE_HCI_UART_UART_PARITY_ODD) UART_PARITY_ODD

CONFIG_BT_LE_HCI_UART_TASK_STACK_SIZE

HCI uart task stack size

Found in: Component config > Bluetooth > Controller Options > HCI Config

Set the size of uart task stack

Default value:

- 1000 if BT_LE_HCI_INTERFACE_USE_UART && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_50_FEATURE_SUPPORT

Enable BLE 5 feature

Found in: Component config > Bluetooth > Controller Options

Enable BLE 5 feature

Default value:

- Yes (enabled) if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

Contains:

- *CONFIG_BT_LE_LL_CFG_FEAT_LE_2M_PHY*
- *CONFIG_BT_LE_LL_CFG_FEAT_LE_CODED_PHY*
- *CONFIG_BT_LE_EXT_ADV*
- *CONFIG_BT_LE_MAX_PERIODIC_ADVERTISER_LIST*
- *CONFIG_BT_LE_MAX_PERIODIC_SYNCS*

CONFIG_BT_LE_LL_CFG_FEAT_LE_2M_PHY

Enable 2M Phy

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT

Enable 2M-PHY

Default value:

- Yes (enabled) if *CONFIG_BT_LE_50_FEATURE_SUPPORT* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_LL_CFG_FEAT_LE_CODED_PHY

Enable coded Phy

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT

Enable coded-PHY

Default value:

- Yes (enabled) if *CONFIG_BT_LE_50_FEATURE_SUPPORT* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_EXT_ADV

Enable extended advertising

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT

Enable this option to do extended advertising. Extended advertising will be supported from BLE 5.0 onwards.

Default value:

- Yes (enabled) if *CONFIG_BT_LE_50_FEATURE_SUPPORT* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_MAX_EXT_ADV_INSTANCES

Maximum number of extended advertising instances.

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT > CONFIG_BT_LE_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. Each extended advertising instance will take about 0.5k DRAM.

Range:

- from 0 to 4 if *CONFIG_BT_LE_EXT_ADV* && *CONFIG_BT_LE_EXT_ADV* && BT_CONTROLLER_ENABLED

Default value:

- 1 if *CONFIG_BT_LE_EXT_ADV* && *CONFIG_BT_LE_EXT_ADV* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_EXT_ADV_MAX_SIZE

Maximum length of the advertising data.

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT > CONFIG_BT_LE_EXT_ADV

Defines the length of the extended adv data. The value should not exceed 1650.

Range:

- from 0 to 1650 if *CONFIG_BT_LE_EXT_ADV* && *CONFIG_BT_LE_EXT_ADV* && BT_CONTROLLER_ENABLED

Default value:

- 1650 if *CONFIG_BT_LE_EXT_ADV* && *CONFIG_BT_LE_EXT_ADV* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_ENABLE_PERIODIC_ADV

Enable periodic advertisement.

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT > CONFIG_BT_LE_EXT_ADV

Enable this option to start periodic advertisement.

Default value:

- Yes (enabled) if *CONFIG_BT_LE_EXT_ADV* && *CONFIG_BT_LE_EXT_ADV* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_PERIODIC_ADV_SYNC_TRANSFER

Enable Transer Sync Events

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT > CONFIG_BT_LE_EXT_ADV > CONFIG_BT_LE_ENABLE_PERIODIC_ADV

This enables controller transfer periodic sync events to host

Default value:

- Yes (enabled) if *CONFIG_BT_LE_ENABLE_PERIODIC_ADV* && *CONFIG_BT_LE_EXT_ADV* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_MAX_PERIODIC_SYNCS

Maximum number of periodic advertising syncs

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT

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 3 if *CONFIG_BT_LE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

Default value:

- 1 if *CONFIG_BT_LE_ENABLE_PERIODIC_ADV* && *CONFIG_BT_LE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED
- 0 if *CONFIG_BT_LE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_MAX_PERIODIC_ADVERTISER_LIST

Maximum number of periodic advertiser list

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_50_FEATURE_SUPPORT

Set this option to set the upper limit for number of periodic advertiser list.

Range:

- from 1 to 5 if *CONFIG_BT_LE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

Default value:

- 5 if *CONFIG_BT_LE_50_FEATURE_SUPPORT* && BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

Memory Settings Contains:

- *CONFIG_BT_LE_ACL_BUF_COUNT*
- *CONFIG_BT_LE_ACL_BUF_SIZE*
- *CONFIG_BT_LE_HCI_EVT_BUF_SIZE*
- *CONFIG_BT_LE_HCI_EVT_HI_BUF_COUNT*
- *CONFIG_BT_LE_HCI_EVT_LO_BUF_COUNT*
- *CONFIG_BT_LE_MSYS_1_BLOCK_COUNT*
- *CONFIG_BT_LE_MSYS_1_BLOCK_SIZE*
- *CONFIG_BT_LE_MSYS_2_BLOCK_COUNT*
- *CONFIG_BT_LE_MSYS_2_BLOCK_SIZE*

CONFIG_BT_LE_MSYS_1_BLOCK_COUNT

MSYS_1 Block Count

Found in: Component config > Bluetooth > Controller 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:

- 12 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_MSYS_1_BLOCK_SIZE

MSYS_1 Block Size

Found in: Component config > Bluetooth > Controller Options > Memory Settings

Dynamic memory size of block 1

Default value:

- 256 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_MSYS_2_BLOCK_COUNT

MSYS_2 Block Count

Found in: Component config > Bluetooth > Controller Options > Memory Settings

Dynamic memory count

Default value:

- 24 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_MSYS_2_BLOCK_SIZE

MSYS_2 Block Size

Found in: Component config > Bluetooth > Controller Options > Memory Settings

Dynamic memory size of block 2

Default value:

- 320 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_ACL_BUF_COUNT

ACL Buffer count

Found in: Component config > Bluetooth > Controller Options > Memory Settings

The number of ACL data buffers.

Default value:

- 10 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_ACL_BUF_SIZE

ACL Buffer size

Found in: Component config > Bluetooth > Controller 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:

- 517 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_EVT_BUF_SIZE

HCI Event Buffer size

Found in: Component config > Bluetooth > Controller 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_LE_EXT_ADV* && BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED
- 70 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_EVT_HI_BUF_COUNT

High Priority HCI Event Buffer count

Found in: Component config > Bluetooth > Controller 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 BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_HCI_EVT_LO_BUF_COUNT

Low Priority HCI Event Buffer count

Found in: Component config > Bluetooth > Controller 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 BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_CONTROLLER_TASK_STACK_SIZE

Controller task stack size

Found in: Component config > Bluetooth > Controller Options

This configures stack size of NimBLE controller task

Default value:

- 5120 if *CONFIG_BLE_MESH* && BT_CONTROLLER_ENABLED
- 4096 if BT_CONTROLLER_ENABLED

CONFIG_BT_LE_LL_RESOLV_LIST_SIZE

BLE LL Resolving list size

Found in: Component config > Bluetooth > Controller Options

Configure the size of resolving list used in link layer.

Range:

- from 1 to 5 if BT_CONTROLLER_ENABLED

Default value:

- 4 if BT_CONTROLLER_ENABLED

CONFIG_BT_LE_SECURITY_ENABLE

Enable BLE SM feature

Found in: Component config > Bluetooth > Controller Options

Enable BLE sm feature

Default value:

- Yes (enabled) if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

Contains:

- *CONFIG_BT_LE_LL_CFG_FEAT_LE_ENCRYPTION*
- *CONFIG_BT_LE_SM_LEGACY*
- *CONFIG_BT_LE_SM_SC*

CONFIG_BT_LE_SM_LEGACY

Security manager legacy pairing

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_SECURITY_ENABLE

Enable security manager legacy pairing

Default value:

- Yes (enabled) if *CONFIG_BT_LE_SECURITY_ENABLE* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_SM_SC

Security manager secure connections (4.2)

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_SECURITY_ENABLE

Enable security manager secure connections

Default value:

- Yes (enabled) if *CONFIG_BT_LE_SECURITY_ENABLE* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_SM_SC_DEBUG_KEYS

Use predefined public-private key pair

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_SECURITY_ENABLE > CONFIG_BT_LE_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_LE_SECURITY_ENABLE* && *CONFIG_BT_LE_SM_SC* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_LL_CFG_FEAT_LE_ENCRYPTION

Enable LE encryption

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_SECURITY_ENABLE

Enable encryption connection

Default value:

- Yes (enabled) if *CONFIG_BT_LE_SECURITY_ENABLE* && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_CRYPTO_STACK_MBEDTLS

Override TinyCrypt with mbedTLS for crypto computations

Found in: Component config > Bluetooth > Controller Options

Enable this option to choose mbedTLS instead of TinyCrypt for crypto computations.

Default value:

- Yes (enabled) if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_WHITELIST_SIZE

BLE white list size

Found in: Component config > Bluetooth > Controller Options

BLE list size

Range:

- from 1 to 15 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

Default value:

- 12 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_LL_DUP_SCAN_LIST_COUNT

BLE duplicate scan list count

Found in: Component config > Bluetooth > Controller Options

config the max count of duplicate scan list

Range:

- from 1 to 100 if BT_CONTROLLER_ENABLED

Default value:

- 20 if BT_CONTROLLER_ENABLED

CONFIG_BT_LE_LL_SCA

BLE Sleep clock accuracy

Found in: Component config > Bluetooth > Controller Options

Sleep clock accuracy of our device (in ppm)

Range:

- from 0 to 500 if BT_CONTROLLER_ENABLED

Default value:

- 60 if BT_CONTROLLER_ENABLED

CONFIG_BT_LE_MAX_CONNECTIONS

Maximum number of concurrent connections

Found in: Component config > Bluetooth > Controller 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. Each connection will take about 1k DRAM.

Range:

- from 1 to 2 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

Default value:

- 2 if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

CONFIG_BT_LE_COEX_PHY_CODED_TX_RX_TLIM

Coexistence: limit on MAX Tx/Rx time for coded-PHY connection

Found in: Component config > Bluetooth > Controller Options

When using PHY-Coded in BLE connection, limitation on max tx/rx time can be applied to better avoid dramatic performance deterioration of Wi-Fi.

Available options:

- Force Enable (BT_LE_COEX_PHY_CODED_TX_RX_TLIM_EN)
  Always enable the limitation on max tx/rx time for Coded-PHY connection
- Force Disable (BT_LE_COEX_PHY_CODED_TX_RX_TLIM_DIS)
  Disable the limitation on max tx/rx time for Coded-PHY connection

CONFIG_BT_LE_SLEEP_ENABLE

Enable BLE sleep

Found in: Component config > Bluetooth > Controller Options

Enable BLE sleep

Default value:

- No (disabled) if BT_CONTROLLER_ENABLED

CONFIG_BT_LE_WAKEUP_SOURCE

BLE light sleep wakeup source

Found in: Component config > Bluetooth > Controller Options > CONFIG_BT_LE_SLEEP_ENABLE

Available options:

- Use ESP timer to wakeup CPU (BT_LE_WAKEUP_SOURCE_CPU_RTC_TIMER)
  Use esp timer to wakeup CPU
- Use BLE rtc timer to wakeup CPU (BT_LE_WAKEUP_SOURCE_BLE_RTC_TIMER)
  Use BLE rtc timer to wakeup CPU

CONFIG_BT_LE_USE_ESP_TIMER

Use Esp Timer for callout

Found in: Component config > Bluetooth > Controller Options

Set this option to use Esp Timer which has higher priority timer instead of FreeRTOS timer

Default value:

- Yes (enabled) if BT_NIMBLE_ENABLED && BT_CONTROLLER_ENABLED

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_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_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_SETTINGS*
- *CONFIG_BLE_MESH_DEINIT*
- *CONFIG_BLE_MESH_USE_DUPLICATE_SCAN*
- *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_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_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 BT_BLUEDROID_ENABLED && *CONFIG_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 (BLE_MESH_MEM_ALLOC_MODE_INTERNAL)
- External SPIRAM (BLE_MESH_MEM_ALLOC_MODE_EXTERNAL)
- Default alloc mode (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 (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 (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 (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_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_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_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_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_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 (BLE_MESH_TRACE_LEVEL_NONE)
- ERROR (BLE_MESH_TRACE_LEVEL_ERROR)
- WARNING (BLE_MESH_TRACE_LEVEL_WARNING)
- INFO (BLE_MESH_TRACE_LEVEL_INFO)
- DEBUG (BLE_MESH_TRACE_LEVEL_DEBUG)
- VERBOSE (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 (BLE_MESH_NET_BUF_TRACE_LEVEL_NONE)
- ERROR (BLE_MESH_NET_BUF_TRACE_LEVEL_ERROR)
- WARNING (BLE_MESH_NET_BUF_TRACE_LEVEL_WARNING)
- INFO (BLE_MESH_NET_BUF_TRACE_LEVEL_INFO)
- DEBUG (BLE_MESH_NET_BUF_TRACE_LEVEL_DEBUG)
- VERBOSE (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_CFG_CLI*
- *CONFIG_BLE_MESH_HEALTH_CLI*
- *CONFIG_BLE_MESH_HEALTH_SRV*

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*

Support for BLE Mesh Client/Server models

Contains:

- *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_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.

Driver Configurations Contains:

- *Analog Comparator Configuration*
- *DAC Configuration*
- *GPIO Configuration*
- *GPTimer Configuration*
- *I2S Configuration*
- *Legacy ADC Configuration*
- *MCPWM Configuration*
- *Parallel IO Configuration*
- *PCNT Configuration*
- *RMT Configuration*
- *Sigma Delta Modulator Configuration*
- *SPI Configuration*
- *Temperature sensor Configuration*
- *TWAI Configuration*
- *UART Configuration*
- *USB Serial/JTAG Configuration*

Legacy ADC 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 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 Configuration

Wether 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 Configuration > Legacy ADC Calibration Configuration

Wether 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)

SPI Configuration

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 > Driver Configurations > SPI Configuration

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.

Default value:

- No (disabled) if *CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH*

CONFIG_SPI_MASTER_ISR_IN_IRAM

Place SPI master ISR function into IRAM

Found in: Component config > Driver Configurations > SPI Configuration

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.

Default value:

- Yes (enabled) if *CONFIG_HEAP_PLACE_FUNCTION_INTO_FLASH*

CONFIG_SPI_SLAVE_IN_IRAM

Place transmitting functions of SPI slave into IRAM

Found in: Component config > Driver Configurations > SPI Configuration

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 > Driver Configurations > SPI Configuration

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)

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) if SOC_TWAI_SUPPORTED

Temperature sensor Configuration Contains:

- *CONFIG_TEMP_SENSOR_ENABLE_DEBUG_LOG*
- *CONFIG_TEMP_SENSOR_SUPPRESS_DEPRECATE_WARN*
- *CONFIG_TEMP_SENSOR_ISR_IRAM_SAFE*

CONFIG_TEMP_SENSOR_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Temperature sensor Configuration

Wether 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)

CONFIG_TEMP_SENSOR_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > Temperature sensor Configuration

Wether 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 > Driver Configurations > Temperature sensor Configuration

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) if SOC_TEMPERATURE_SENSOR_INTR_SUPPORT

UART Configuration Contains:

- *CONFIG_UART_ISR_IN_IRAM*

CONFIG_UART_ISR_IN_IRAM

Place UART ISR function into IRAM

Found in: Component config > Driver Configurations > UART Configuration

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.

Default value:

- No (disabled) if *CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH*

GPIO Configuration Contains:

- *CONFIG_GPIO_CTRL_FUNC_IN_IRAM*

CONFIG_GPIO_CTRL_FUNC_IN_IRAM

Place GPIO control functions into IRAM

Found in: Component config > Driver Configurations > GPIO Configuration

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)

Sigma Delta Modulator Configuration Contains:

- *CONFIG_SDM_ENABLE_DEBUG_LOG*
- *CONFIG_SDM_CTRL_FUNC_IN_IRAM*
- *CONFIG_SDM_SUPPRESS_DEPRECATE_WARN*

CONFIG_SDM_CTRL_FUNC_IN_IRAM

Place SDM control functions into IRAM

Found in: Component config > Driver Configurations > Sigma Delta Modulator Configuration

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) if SOC_SDM_SUPPORTED

CONFIG_SDM_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > Sigma Delta Modulator Configuration

Wether to suppress the deprecation warnings when using legacy sigma delta driver. 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_SDM_SUPPORTED

CONFIG_SDM_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > Sigma Delta Modulator Configuration

Wether 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) if SOC_SDM_SUPPORTED

Analog Comparator Configuration

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 > Driver Configurations > Analog Comparator Configuration

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) if SOC_ANA_CMPR_SUPPORTED

CONFIG_ANA_CMPR_CTRL_FUNC_IN_IRAM

Place Analog Comparator control functions into IRAM

Found in: Component config > Driver Configurations > Analog Comparator Configuration

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) if SOC_ANA_CMPR_SUPPORTED

CONFIG_ANA_CMPR_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > Analog Comparator Configuration

Wether 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) if SOC_ANA_CMPR_SUPPORTED

GPTimer Configuration

Contains:

- *CONFIG_GPTIMER_ENABLE_DEBUG_LOG*
- *CONFIG_GPTIMER_ISR_IRAM_SAFE*
- *CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM*
- *CONFIG_GPTIMER_SUPPRESS_DEPRECATE_WARN*

CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM

Place GPTimer control functions into IRAM

Found in: Component config > Driver Configurations > GPTimer Configuration

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 > Driver Configurations > GPTimer Configuration

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_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > GPTimer Configuration

Wether 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)

CONFIG_GPTIMER_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > GPTimer Configuration

Wether 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)

PCNT Configuration Contains:

- *CONFIG_PCNT_ENABLE_DEBUG_LOG*
- *CONFIG_PCNT_ISR_IRAM_SAFE*
- *CONFIG_PCNT_CTRL_FUNC_IN_IRAM*
- *CONFIG_PCNT_SUPPRESS_DEPRECATE_WARN*

CONFIG_PCNT_CTRL_FUNC_IN_IRAM

Place PCNT control functions into IRAM

Found in: Component config > Driver Configurations > PCNT Configuration

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) if SOC_PCNT_SUPPORTED

CONFIG_PCNT_ISR_IRAM_SAFE

PCNT ISR IRAM-Safe

Found in: Component config > Driver Configurations > PCNT Configuration

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) if SOC_PCNT_SUPPORTED

CONFIG_PCNT_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > PCNT Configuration

Wether 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) if SOC_PCNT_SUPPORTED

CONFIG_PCNT_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > PCNT Configuration

Wether 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) if SOC_PCNT_SUPPORTED

RMT Configuration Contains:

- *CONFIG_RMT_ENABLE_DEBUG_LOG*
- *CONFIG_RMT_ISR_IRAM_SAFE*
- *CONFIG_RMT_SUPPRESS_DEPRECATE_WARN*

CONFIG_RMT_ISR_IRAM_SAFE

RMT ISR IRAM-Safe

Found in: Component config > Driver Configurations > RMT Configuration

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) if SOC_RMT_SUPPORTED

CONFIG_RMT_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > RMT Configuration

Wether 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) if SOC_RMT_SUPPORTED

CONFIG_RMT_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > RMT Configuration

Wether 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) if SOC_RMT_SUPPORTED

MCPWM Configuration

Contains:

- *CONFIG_MCPWM_ENABLE_DEBUG_LOG*
- *CONFIG_MCPWM_CTRL_FUNC_IN_IRAM*
- *CONFIG_MCPWM_ISR_IRAM_SAFE*
- *CONFIG_MCPWM_SUPPRESS_DEPRECATE_WARN*

CONFIG_MCPWM_ISR_IRAM_SAFE

Place MCPWM ISR function into IRAM

Found in: Component config > Driver Configurations > MCPWM Configuration

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) if SOC_MCPWM_SUPPORTED

CONFIG_MCPWM_CTRL_FUNC_IN_IRAM

Place MCPWM control functions into IRAM

Found in: Component config > Driver Configurations > MCPWM Configuration

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) if SOC_MCPWM_SUPPORTED

CONFIG_MCPWM_SUPPRESS_DEPRECATE_WARN

Suppress leagcy driver deprecated warning

Found in: Component config > Driver Configurations > MCPWM Configuration

Wether 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) if SOC_MCPWM_SUPPORTED

CONFIG_MCPWM_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > MCPWM Configuration

Wether 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) if SOC_MCPWM_SUPPORTED

I2S Configuration

Contains:

- *CONFIG_I2S_ENABLE_DEBUG_LOG*
- *CONFIG_I2S_ISR_IRAM_SAFE*
- *CONFIG_I2S_SUPPRESS_DEPRECATE_WARN*

CONFIG_I2S_ISR_IRAM_SAFE

I2S ISR IRAM-Safe

Found in: Component config > Driver Configurations > I2S Configuration

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) if SOC_I2S_SUPPORTED

CONFIG_I2S_SUPPRESS_DEPRECATE_WARN

Suppress leagcy driver deprecated warning

Found in: Component config > Driver Configurations > I2S Configuration

Enable this option will suppress the deprecation warnings of using APIs in legacy I2S driver.

Default value:

- No (disabled) if SOC_I2S_SUPPORTED

CONFIG_I2S_ENABLE_DEBUG_LOG

Enable I2S debug log

Found in: Component config > Driver Configurations > I2S Configuration

Wether 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) if SOC_I2S_SUPPORTED

DAC Configuration 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_SUPPRESS_DEPRECATE_WARN*

CONFIG_DAC_CTRL_FUNC_IN_IRAM

Place DAC control functions into IRAM

Found in: Component config > Driver Configurations > DAC Configuration

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 > Driver Configurations > DAC Configuration

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_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

Found in: Component config > Driver Configurations > DAC Configuration

Wether 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

CONFIG_DAC_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > DAC Configuration

Wether 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 > Driver Configurations > DAC Configuration

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

USB Serial/JTAG Configuration

Contains:

- *CONFIG_USJ_NO_AUTO_LS_ON_CONNECTION*

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 > Driver Configurations > USB Serial/JTAG Configuration

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.

Parallel IO Configuration

Contains:

- *CONFIG_PARLIO_ENABLE_DEBUG_LOG*
- *CONFIG_PARLIO_ISR_IRAM_SAFE*

CONFIG_PARLIO_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > Driver Configurations > Parallel IO Configuration

Wether 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) if SOC_PARLIO_SUPPORTED

CONFIG_PARLIO_ISR_IRAM_SAFE

Parallel IO ISR IRAM-Safe

Found in: Component config > Driver Configurations > Parallel IO Configuration

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) if SOC_PARLIO_SUPPORTED

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_LIBRARY_CHOOSE*
- *CONFIG_ESP_TLS_CLIENT_SESSION_TICKETS*
- *CONFIG_ESP_DEBUG_WOLFSSL*
- *CONFIG_ESP_TLS_SERVER*
- *CONFIG_ESP_TLS_PSK_VERIFICATION*
- *CONFIG_ESP_WOLFSSL_SMALL_CERT_VERIFY*
- *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 (ESP_TLS_USING_MBEDTLS)
- wolfSSL (License info in wolfSSL directory README) (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) if ESP_TLS_USING_MBEDTLS && SOC_DIG_SIGN_SUPPORTED

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

Enable ESP-TLS Server

Found in: Component config > ESP-TLS

Enable support for creating server side SSL/TLS session, available for mbedTLS as well as wolfSSL TLS library.

CONFIG_ESP_TLS_SERVER_SESSION_TICKETS

Enable server session tickets

Found in: Component config > ESP-TLS > CONFIG_ESP_TLS_SERVER

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 > 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 > CONFIG_ESP_TLS_SERVER

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 > CONFIG_ESP_TLS_SERVER

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 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_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) if SOC_ADC_DMA_SUPPORTED

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

Wireless Coexistence Contains:

- *CONFIG_ESP_COEX_EXTERNAL_COEXIST_ENABLE*
- *CONFIG_ESP_COEX_SW_COEXIST_ENABLE*

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.

Default value:

- Yes (enabled) if *CONFIG_BT_ENABLED*

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.

Default value:

- No (disabled) if *CONFIG_BT_ENABLED* || NIMBLE_ENABLED

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)

Ethernet Contains:

- *CONFIG_ETH_TRANSMIT_MUTEX*
- *CONFIG_ETH_USE_OPENETH*
- *CONFIG_ETH_USE_SPI_ETHERNET*

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_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.

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 if *CONFIG_ESP_GDBSTUB_SUPPORT_TASKS*

ESP HTTP client Contains:

- *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)

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:

- *Chip revision*
- *ESP_SLEEP_WORKAROUND*
- *ETM Configuration*
- *GDMA Configuration*
- *MAC Config*
- *Main XTAL Config*
- *Peripheral Control*

- *RTC Clock Config*
- *Sleep Config*

Chip revision Contains:

- *CONFIG_ESP32C2_REV2_DEVELOPMENT*
- *CONFIG_ESP_REV_NEW_CHIP_TEST*
- *CONFIG_ESP32C2_REV_MIN*

CONFIG_ESP32C2_REV_MIN

Minimum Supported ESP32-C2 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 v1.0 (ECO1) (ESP32C2_REV_MIN_1)
- Rev v1.1 (ECO2) (ESP32C2_REV_MIN_1_1)

CONFIG_ESP32C2_REV2_DEVELOPMENT

Develop on ESP32-C2 v2.0 (Preview)

Found in: Component config > Hardware Settings > Chip revision

Default value:

- Yes (enabled)

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_ESP32C2_UNIVERSAL_MAC_ADDRESSES*

CONFIG_ESP32C2_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 four, all four interfaces (WiFi station, WiFi softap, Bluetooth and Ethernet) receive a universally administered MAC address. These are generated sequentially by adding 0, 1, 2 and 3 (respectively) to the final octet of the base MAC address.

If the number of universal MAC addresses is two, only two interfaces (WiFi station and Bluetooth) receive a universally administered MAC address. These are generated sequentially by adding 0 and 1 (respectively) to the base MAC address. The remaining two interfaces (WiFi softap and Ethernet) receive local MAC addresses. These are derived from the universal WiFi station and Bluetooth MAC addresses, respectively.

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 (either 2 or 4 per device.)

Note that ESP32-C2 has no integrated Ethernet MAC. Although it ' s possible to use the esp_read_mac() API to return a MAC for Ethernet, this can only be used with an external MAC peripheral.

Available options:

- Two (ESP32C2_UNIVERSAL_MAC_ADDRESSES_TWO)
- Four (ESP32C2_UNIVERSAL_MAC_ADDRESSES_FOUR)

Sleep Config Contains:

- *CONFIG_ESP_SLEEP_MSPI_NEED_ALL_IO_PU*
- *CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND*
- *CONFIG_ESP_SLEEP_GPIO_RESET_WORKAROUND*
- *CONFIG_ESP_SLEEP_POWER_DOWN_FLASH*
- *CONFIG_ESP_SLEEP_PSRAM_LEAKAGE_WORKAROUND*

CONFIG_ESP_SLEEP_POWER_DOWN_FLASH

Power down flash in light sleep when there is no SPIRAM

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.

Default value:

- No (disabled) if SPIRAM

CONFIG_ESP_SLEEP_GPIO_RESET_WORKAROUND

light sleep GPIO reset workaround

Found in: Component config > Hardware Settings > Sleep Config

esp32c2, esp32c3 and esp32s3 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.

Default value:

- Yes (enabled)

CONFIG_ESP_SLEEP_PSRAM_LEAKAGE_WORKAROUND

PSRAM leakage current workaround in light sleep

Found in: Component config > Hardware Settings > Sleep Config

When the CS pin of SPIRAM is not pulled up, the sleep current will increase during light sleep. If the CS pin of SPIRAM has an external pull-up, you do not need to select this option, otherwise, you should enable this option.

Default value:

- Yes (enabled) if SPIRAM

CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND

Flash leakage current workaround in light sleep

Found in: Component config > Hardware Settings > Sleep Config

When the CS pin of Flash is not pulled up, the sleep current will increase during light sleep. If the CS pin of Flash has an external pull-up, you do not need to select this option, otherwise, you should enable this option.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

CONFIG_ESP_SLEEP_MSPI_NEED_ALL_IO_PU

All pins of mspi need pull up

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.

Default value:

- Yes (enabled)

ESP_SLEEP_WORKAROUND

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 136kHz RC oscillator (RTC_CLK_SRC_INT_RC)
- External 32kHz oscillator at pin0 (RTC_CLK_SRC_EXT_OSC)

- Internal 17.5MHz oscillator, divided by 256 (RTC_CLK_SRC_INT_8MD256)

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 27000 if RTC_CLK_SRC_EXT_OSC || RTC_CLK_SRC_INT_8MD256
- from 0 to 32766

Default value:
- 3000 if RTC_CLK_SRC_EXT_OSC || RTC_CLK_SRC_INT_8MD256
- 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

Wether 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) if SOC_ETM_SUPPORTED

GDMA Configuration Contains:

- *CONFIG_GDMA_ISR_IRAM_SAFE*
- *CONFIG_GDMA_CTRL_FUNC_IN_IRAM*

CONFIG_GDMA_CTRL_FUNC_IN_IRAM

Place GDMA control functions into IRAM

Found in: Component config > Hardware Settings > GDMA Configuration

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. Enabling this option can improve driver performance as well.

Default value:

- No (disabled)

CONFIG_GDMA_ISR_IRAM_SAFE

GDMA ISR IRAM-Safe

Found in: Component config > Hardware Settings > GDMA Configuration

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)

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 (XTAL_FREQ_24)
- 26 MHz (XTAL_FREQ_26)
- 32 MHz (XTAL_FREQ_32)
- 40 MHz (XTAL_FREQ_40)
- Autodetect (XTAL_FREQ_AUTO)

LCD and Touch Panel Contains:

- *LCD Peripheral Configuration*

LCD Peripheral Configuration Contains:

- *CONFIG_LCD_ENABLE_DEBUG_LOG*
- *CONFIG_LCD_PANEL_IO_FORMAT_BUF_SIZE*
- *CONFIG_LCD_RGB_RESTART_IN_VSYNC*
- *CONFIG_LCD_RGB_ISR_IRAM_SAFE*

CONFIG_LCD_PANEL_IO_FORMAT_BUF_SIZE

LCD panel io format buffer size

Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration

LCD driver allocates an internal buffer to transform the data into a proper format, because of the endian order mismatch. This option is to set the size of the buffer, in bytes.

Default value:

- 32

CONFIG_LCD_ENABLE_DEBUG_LOG

Enable debug log

Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration

Wether 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

ESP NETIF Adapter Contains:

- *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_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 everytime 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 (ESP_NETIF_TCPIP_LWIP)
  lwIP is a small independent implementation of the TCP/IP protocol suite.
- Loopback (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_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 avaiable!

Default value:
- No (disabled)

Partition API Configuration

PHY Contains:

- *CONFIG_ESP_PHY_CALIBRATION_MODE*
- *CONFIG_ESP_PHY_ENABLE_USB*
- *CONFIG_ESP_PHY_MAX_WIFI_TX_POWER*
- *CONFIG_ESP_PHY_MAC_BB_PD*
- *CONFIG_ESP_PHY_REDUCE_TX_POWER*
- *CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE*
- *CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*

CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE

Store phy calibration data in NVS

Found in: Component config > PHY

If this option is enabled, NVS will be initialized and calibration data will be loaded from there. PHY calibration will be skipped on deep sleep wakeup. If calibration data is not found, full calibration will be performed and stored in NVS. Normally, only partial calibration will be performed. If this option is disabled, full calibration will be performed.

If it's easy that your board calibrate bad data, choose 'n'. Two cases for example, you should choose 'n': 1.If your board is easy to be booted up with antenna disconnected. 2.Because of your board design, each time when you do calibration, the result are too unstable. If unsure, choose 'y'.

Default value:
- Yes (enabled)

CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION

Use a partition to store PHY init data

Found in: Component config > PHY

If enabled, PHY init data will be loaded from a partition. When using a custom partition table, make sure that PHY data partition is included (type: 'data', subtype: 'phy'). With default partition tables, this is done automatically. If PHY init data is stored in a partition, it has to be flashed there, otherwise runtime error will occur.

If this option is not enabled, PHY init data will be embedded into the application binary.

If unsure, choose ‘n’.

Default value:

- No (disabled)

Contains:

- *CONFIG_ESP_PHY_DEFAULT_INIT_IF_INVALID*
- *CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN*

CONFIG_ESP_PHY_DEFAULT_INIT_IF_INVALID

Reset default PHY init data if invalid

Found in: Component config > PHY > CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION

If enabled, PHY init data will be restored to default if it cannot be verified successfully to avoid endless bootloops.

If unsure, choose ‘n’.

Default value:

- No (disabled) if *CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*

CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN

Support multiple PHY init data bin

Found in: Component config > PHY > CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION

If enabled, the corresponding PHY init data type can be automatically switched according to the country code. China’s PHY init data bin is used by default. Can be modified by country information in API esp_wifi_set_country(). The priority of switching the PHY init data type is: 1. Country configured by API esp_wifi_set_country() and the parameter policy is WIFI_COUNTRY_POLICY_MANUAL. 2. Country notified by the connected AP. 3. Country configured by API esp_wifi_set_country() and the parameter policy is WIFI_COUNTRY_POLICY_AUTO.

Default value:

- No (disabled) if *CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION* && *CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*

CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN_EMBED

Support embedded multiple phy init data bin to app bin

Found in: Component config > PHY > CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION > CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN

If enabled, multiple phy init data bin will embedded into app bin If not enabled, multiple phy init data bin will still leave alone, and need to be flashed by users.

Default value:

- No (disabled) if *CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN* && *CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*

CONFIG_ESP_PHY_INIT_DATA_ERROR

Terminate operation when PHY init data error

Found in: Component config > PHY > CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION > CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN

If enabled, when an error occurs while the PHY init data is updated, the program will terminate and restart. If not enabled, the PHY init data will not be updated when an error occurs.

Default value:

- No (disabled) if *CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN* && *CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*

CONFIG_ESP_PHY_MAX_WIFI_TX_POWER

Max WiFi TX power (dBm)

Found in: Component config > PHY

Set maximum transmit power for WiFi radio. Actual transmit power for high data rates may be lower than this setting.

Range:

- from 10 to 20

Default value:

- 20

CONFIG_ESP_PHY_MAC_BB_PD

Power down MAC and baseband of Wi-Fi and Bluetooth when PHY is disabled

Found in: Component config > PHY

If enabled, the MAC and baseband of Wi-Fi and Bluetooth will be powered down when PHY is disabled. Enabling this setting reduces power consumption by a small amount but increases RAM use by approximately 4 KB(Wi-Fi only), 2 KB(Bluetooth only) or 5.3 KB(Wi-Fi + Bluetooth).

Default value:

- No (disabled) if SOC_PM_SUPPORT_MAC_BB_PD && *CONFIG_FREERTOS_USE_TICKLESS_IDLE*

CONFIG_ESP_PHY_REDUCE_TX_POWER

Reduce PHY TX power when brownout reset

Found in: Component config > PHY

When brownout reset occurs, reduce PHY TX power to keep the code running.

Default value:

- No (disabled)

CONFIG_ESP_PHY_ENABLE_USB

Enable USB when phy init

Found in: Component config > PHY

When using USB Serial/JTAG/OTG/CDC, PHY should enable USB, otherwise USB module can not work properly. Notice: Enabling this configuration option will slightly impact wifi performance.

Default value:

- No (disabled) if ESP_CONSOLE_USB_SERIAL_JTAG || ESP_CONSOLE_SECONDARY_USB_SERIAL_JTAG

CONFIG_ESP_PHY_CALIBRATION_MODE

Calibration mode

Found in: Component config > PHY

Select PHY calibration mode. During RF initialization, the partial calibration method is used by default for RF calibration. Full calibration takes about 100ms more than partial calibration. If boot duration is not critical, it is suggested to use the full calibration method. No calibration method is only used when the device wakes up from deep sleep.

Available options:

- Calibration partial (ESP_PHY_RF_CAL_PARTIAL)
- Calibration none (ESP_PHY_RF_CAL_NONE)
- Calibration full (ESP_PHY_RF_CAL_FULL)

Power Management Contains:

- *CONFIG_PM_SLP_DISABLE_GPIO*
- *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) if *CONFIG_FREERTOS_SMP* || __DOXYGEN__

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 1.8KB of lightsleep related source code would be in IRAM and chip would sleep longer for 760us at 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 260B of RTOS_IDLE related source code would be in IRAM and chip would sleep longer for 40us at 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. Waring: 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_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 use ESP_PM_NO_LIGHT_SLEEP to protect your UART transmission, fflush the stdout after printing, or increase FREERTOS_IDLE_TIME_BEFORE_SLEEP threshold in menuconfig.

Default value:

- No (disabled) if SOC_PAU_SUPPORTED

ESP PSRAM

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_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 (ESP_DEFAULT_CPU_FREQ_MHZ_40)
- 80 MHz (ESP_DEFAULT_CPU_FREQ_MHZ_80)
- 120 MHz (ESP_DEFAULT_CPU_FREQ_MHZ_120)

Cache config Contains:

- *CONFIG_ESP32C2_INSTRUCTION_CACHE_WRAP*

CONFIG_ESP32C2_INSTRUCTION_CACHE_WRAP

Instruction cache wrap

Found in: Component config > ESP System Settings > Cache config

If enabled, instruction cache will use wrap mode to read spi flash. The wrap length is fixed to 32B

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 (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 (ESP_SYSTEM_PANIC_PRINT_REBOOT)
  Outputs the relevant registers over the serial port and immediately reset the processor.
- Silent reboot (ESP_SYSTEM_PANIC_SILENT_REBOOT)
  Just resets the processor without outputting anything

- GDBStub on panic (ESP_SYSTEM_PANIC_GDBSTUB)
  Invoke gdbstub on the serial port, allowing for gdb to attach to it to do a postmortem of the crash.
- GDBStub at runtime (ESP_SYSTEM_GDBSTUB_RUNTIME)
  Invoke gdbstub on the serial port, allowing for gdb to attach to it and to do a debug on runtime.

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.

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 (ESP_MAIN_TASK_AFFINITY_CPU0)
- CPU1 (ESP_MAIN_TASK_AFFINITY_CPU1)
- No affinity (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 (ESP_CONSOLE_UART_DEFAULT)
- USB CDC (ESP_CONSOLE_USB_CDC)
- USB Serial/JTAG Controller (ESP_CONSOLE_USB_SERIAL_JTAG)
- Custom UART (ESP_CONSOLE_UART_CUSTOM)
- None (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 (ESP_CONSOLE_SECONDARY_NONE)
- USB_SERIAL_JTAG PORT (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 (ESP_CONSOLE_UART_CUSTOM_NUM_0)
- UART1 (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 46 if ESP_CONSOLE_UART_CUSTOM

Default value:

- 20 if ESP_CONSOLE_UART_CUSTOM
- 43 if 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 46 if ESP_CONSOLE_UART_CUSTOM

Default value:
- 19 if ESP_CONSOLE_UART_CUSTOM
- 44 if 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 4000000 if *CONFIG_PM_ENABLE*
- from 1200 to 1000000 if *CONFIG_PM_ENABLE*

Default value:
- 74880 if XTAL_FREQ_26
- 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 miliseconds. 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.

Default value:
- Yes (enabled)

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.

Default value:

- Yes (enabled)

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.

Default value:

- "COMPILER_OPTIMIZATION_LEVEL_DEBUG" if ESP32_TRAX && ESP32S2_TRAX && ESP32S3_TRAX

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 and other system checks.

Available options:

- Level 5 interrupt (ESP_SYSTEM_CHECK_INT_LEVEL_5)
  Using level 5 interrupt for Interrupt Watchdog and other system checks.
- Level 4 interrupt (ESP_SYSTEM_CHECK_INT_LEVEL_4)
  Using level 4 interrupt for Interrupt Watchdog 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-C2 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.

#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:

- 2.51V (ESP_BROWNOUT_DET_LVL_SEL_7)
- 2.64V (ESP_BROWNOUT_DET_LVL_SEL_6)
- 2.76V (ESP_BROWNOUT_DET_LVL_SEL_5)
- 2.92V (ESP_BROWNOUT_DET_LVL_SEL_4)
- 3.10V (ESP_BROWNOUT_DET_LVL_SEL_3)
- 3.27V (ESP_BROWNOUT_DET_LVL_SEL_2)

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.

Default value:

- Yes (enabled)

High resolution timer (esp_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 > High resolution timer (esp_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 > High resolution timer (esp_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 > High resolution timer (esp_timer)

It sets the interrupt level for esp_timer ISR in range 1..3. A higher level (3) helps to decrease the ISR esp_timer latency.

Range:

- from 1 to 1

Default value:

- 1

CONFIG_ESP_TIMER_SHOW_EXPERIMENTAL

show esp_timer’ s experimental features

Found in: Component config > High resolution timer (esp_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 > High resolution timer (esp_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 (ESP_TIMER_TASK_AFFINITY_CPU0)
- CPU1 (ESP_TIMER_TASK_AFFINITY_CPU1)
- No affinity (ESP_TIMER_TASK_AFFINITY_NO_AFFINITY)

CONFIG_ESP_TIMER_ISR_AFFINITY

timer interrupt core affinity

Found in: Component config > High resolution timer (esp_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 (ESP_TIMER_ISR_AFFINITY_CPU0)
- CPU1 (ESP_TIMER_ISR_AFFINITY_CPU1)
- No affinity (ESP_TIMER_ISR_AFFINITY_NO_AFFINITY)

CONFIG_ESP_TIMER_SUPPORTS_ISR_DISPATCH_METHOD

Support ISR dispatch method

Found in: Component config > High resolution timer (esp_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_11KV_SUPPORT*
- *CONFIG_ESP_WIFI_11R_SUPPORT*

- *CONFIG_ESP_WIFI_DPP_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_WIFI_RX_STATS*
- *CONFIG_ESP_WIFI_ENABLE_WIFI_TX_STATS*
- *CONFIG_ESP_WIFI_ENABLE_WPA3_SAE*
- *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_STATIC_RX_BUFFER_NUM*
- *CONFIG_ESP_WIFI_STATIC_TX_BUFFER_NUM*
- *CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM*
- *CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE*
- *CONFIG_ESP_WIFI_DEBUG_PRINT*
- *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_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*
- *WPS Configuration Options*

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 25 if SOC_WIFI_HE_SUPPORT
- from 2 to 128 if SOC_WIFI_HE_SUPPORT

Default value:

- 10 if SPIRAM_TRY_ALLOCATE_WIFI_LWIP
- 16 if SPIRAM_TRY_ALLOCATE_WIFI_LWIP

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 128 if *CONFIG_LWIP_WND_SCALE*
- from 0 to 1024 if *CONFIG_LWIP_WND_SCALE*

Default value:

- 32

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 (ESP_WIFI_STATIC_TX_BUFFER)
- Dynamic (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 ESP_WIFI_STATIC_TX_BUFFER

Default value:

- 16 if ESP_WIFI_STATIC_TX_BUFFER

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 SPIRAM

Default value:
- 32 if SPIRAM

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

Default value:
- 32

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_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)

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 32 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_WIFI_AMPDU_TX_ENABLED*
- from 2 to 64 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_WIFI_AMPDU_TX_ENABLED*

Default value:

- 6

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)

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 prefered to allocat 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 32 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_WIFI_AMPDU_RX_ENABLED*
- from 2 to 64 if SOC_WIFI_HE_SUPPORT && *CONFIG_ESP_WIFI_AMPDU_RX_ENABLED*

Default value:

- 6 if SPIRAM_TRY_ALLOCATE_WIFI_LWIP && *CONFIG_ESP_WIFI_AMPDU_RX_ENABLED*
- 16 if SPIRAM_TRY_ALLOCATE_WIFI_LWIP && *CONFIG_ESP_WIFI_AMPDU_RX_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 SPIRAM

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)

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 (ESP_WIFI_TASK_PINNED_TO_CORE_0)
- Core 1 (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 occurence 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 ontop 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

Default value:

- 752

CONFIG_ESP_WIFI_MGMT_SBUF_NUM

WiFi mgmt short buffer number

Found in: Component config > Wi-Fi

Set the number of WiFi management short buffer.

Range:

- from 6 to 32

Default value:

- 32

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)

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)

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)

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)

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)

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)

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.

CONFIG_ESP_WIFI_SLP_DEFAULT_MIN_ACTIVE_TIME

Minimum active time

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_IRAM_OPT

The minimum timeout for waiting to receive data, unit: milliseconds.

Range:

- from 8 to 60 if *CONFIG_ESP_WIFI_SLP_IRAM_OPT*

Default value:

- 50 if *CONFIG_ESP_WIFI_SLP_IRAM_OPT*

CONFIG_ESP_WIFI_SLP_DEFAULT_MAX_ACTIVE_TIME

Maximum keep alive time

Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_IRAM_OPT

The maximum time that wifi keep alive, unit: seconds.

Range:

- from 10 to 60 if *CONFIG_ESP_WIFI_SLP_IRAM_OPT*

Default value:

- 10 if *CONFIG_ESP_WIFI_SLP_IRAM_OPT*

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_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_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_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)

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_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:

- No (disabled)

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)

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 *CONFIG_ESP_PHY_MAC_BB_PD* && SOC_PM_SUPPORT_BEACON_WAKEUP

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*

Default value:

- 10 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*

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*

Default value:

- 3 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*

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*

Default value:

- 2 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*

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*

Default value:
- 2 if *CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*

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 4
- from 0 to 17

Default value:
- 2
- 7

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_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:
- Yes (enabled) if SOC_WIFI_HE_SUPPORT

CONFIG_ESP_WIFI_MBEDTLS_CRYPTO

Use MbedTLS crypto APIs

Found in: Component config > Wi-Fi

Select this option to use MbedTLS crypto APIs which utilize hardware acceleration.

Default value:

- Yes (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. Incase your server is using one of these version, it is advisable to update your server. Please disable this option for compatibilty with older TLS versions.

Default value:

- Yes (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_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)

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)

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_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)

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)

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)

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)

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:

- Yes (enabled) if SOC_WIFI_HE_SUPPORT

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:

- Yes (enabled) if *CONFIG_ESP_WIFI_ENABLE_WIFI_RX_STATS*

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 workaorunds with various APs. Enabling this may cause inter operability issues with some APs.

Default value:

- No (disabled)

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 capabilites to get the configuration in passphrase mode. Not recommanded 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)

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)

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)

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_DECODE*
- *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 (ESP_COREDUMP_ENABLE_TO_FLASH)
- UART (ESP_COREDUMP_ENABLE_TO_UART)
- None (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 (ESP_COREDUMP_DATA_FORMAT_BIN)
- ELF format (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 (ESP_COREDUMP_CHECKSUM_CRC32)
- Use SHA256 for integrity verification (ESP_COREDUMP_CHECKSUM_SHA256)

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 ESP_COREDUMP_ENABLE_TO_FLASH

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 ESP_COREDUMP_ENABLE_TO_UART

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 above 800. 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 ESP_COREDUMP_DATA_FORMAT_ELF && ESP_COREDUMP_ENABLE_TO_FLASH

Default value:

- 1024 if ESP_COREDUMP_DATA_FORMAT_ELF && 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) (ESP_COREDUMP_DECODE_INFO)
- Don’t decode (ESP_COREDUMP_DECODE_DISABLE)

FAT Filesystem support Contains:

- *CONFIG_FATFS_API_ENCODING*
- *CONFIG_FATFS_VFS_FSTAT_BLKSIZE*
- *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_ALLOC_PREFER_EXTRAM*
- *CONFIG_FATFS_SECTOR_SIZE*
- *CONFIG_FATFS_TIMEOUT_MS*
- *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 (FATFS_LFN_NONE)
- Long filename buffer in heap (FATFS_LFN_HEAP)
- Long filename buffer on stack (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 (FATFS_SECTOR_512)
- 4096 (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) (FATFS_CODEPAGE_DYNAMIC)
- US (CP437) (FATFS_CODEPAGE_437)
- Arabic (CP720) (FATFS_CODEPAGE_720)
- Greek (CP737) (FATFS_CODEPAGE_737)
- KBL (CP771) (FATFS_CODEPAGE_771)
- Baltic (CP775) (FATFS_CODEPAGE_775)
- Latin 1 (CP850) (FATFS_CODEPAGE_850)
- Latin 2 (CP852) (FATFS_CODEPAGE_852)
- Cyrillic (CP855) (FATFS_CODEPAGE_855)
- Turkish (CP857) (FATFS_CODEPAGE_857)
- Portugese (CP860) (FATFS_CODEPAGE_860)
- Icelandic (CP861) (FATFS_CODEPAGE_861)
- Hebrew (CP862) (FATFS_CODEPAGE_862)
- Canadian French (CP863) (FATFS_CODEPAGE_863)
- Arabic (CP864) (FATFS_CODEPAGE_864)
- Nordic (CP865) (FATFS_CODEPAGE_865)
- Russian (CP866) (FATFS_CODEPAGE_866)
- Greek 2 (CP869) (FATFS_CODEPAGE_869)
- Japanese (DBCS) (CP932) (FATFS_CODEPAGE_932)
- Simplified Chinese (DBCS) (CP936) (FATFS_CODEPAGE_936)
- Korean (DBCS) (CP949) (FATFS_CODEPAGE_949)
- Traditional Chinese (DBCS) (CP950) (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.

Range:

- from 12 to 255

Default value:

- 255

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 (FATFS_API_ENCODING_ANSI_OEM)
- API uses UTF-8 encoding (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 lenghty 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

Perfer 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 SPIRAM_USE_CAPS_ALLOC || 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

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_TASK_PRIORITY*
- *CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH*
- *CONFIG_FREERTOS_USE_IDLE_HOOK*
- *CONFIG_FREERTOS_OPTIMIZED_SCHEDULER*
- *CONFIG_FREERTOS_USE_TICK_HOOK*
- *CONFIG_FREERTOS_USE_TICKLESS_IDLE*
- *CONFIG_FREERTOS_USE_TRACE_FACILITY*

- *CONFIG_FREERTOS_UNICORE*
- *CONFIG_FREERTOS_SMP*
- *CONFIG_FREERTOS_USE_MINIMAL_IDLE_HOOK*

CONFIG_FREERTOS_SMP

Run the Amazon SMP FreeRTOS kernel instead (FEATURE UNDER DEVELOPMENT)

Found in: Component config > FreeRTOS > Kernel

Amazon has released an SMP version of the FreeRTOS Kernel which can be found via the following link: https://github.com/FreeRTOS/FreeRTOS-Kernel/tree/smp

IDF has added an experimental port of this SMP kernel located in components/freertos/FreeRTOS-Kernel-SMP. Enabling this option will cause IDF to use the Amazon SMP kernel. Note that THIS FEATURE IS UNDER ACTIVE DEVELOPMENT, users use this at their own risk.

Leaving this option disabled will mean the IDF FreeRTOS kernel is used instead, which is located in: components/freertos/FreeRTOS-Kernel. Both kernel versions are SMP capable, but differ in their implementation and features.

Default value:

- No (disabled)

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_SMP*

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 (FREERTOS_CHECK_STACKOVERFLOW_NONE)
  Do not check for stack overflows (configCHECK_FOR_STACK_OVERFLOW = 0)
- Check by stack pointer value (Method 1) (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) (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_MINIMAL_IDLE_HOOK

Use FreeRTOS minimal idle hook

Found in: Component config > FreeRTOS > Kernel

Enables the minimal idle task application hook (see configUSE_IDLE_HOOK documentation for more details).

Note:

- The application must provide the hook function `void vApplicationMinimalIdleHook( void );`
- `vApplicationMinimalIdleHook()` is called from FreeRTOS minimal idle task(s)

Default value:

- No (disabled) if *CONFIG_FREERTOS_SMP*

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_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_VTASKLIST_INCLUDE_COREID

Enable display of xCoreID in vTaskList

Found in: Component config > FreeRTOS > Kernel > CONFIG_FREERTOS_USE_TRACE_FACILITY > CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS

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_SMP* && *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_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*

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_ENABLE_TASK_SNAPSHOT*
- *CONFIG_FREERTOS_TLSP_DELETION_CALLBACKS*

- *CONFIG_FREERTOS_ISR_STACKSIZE*
- *CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH*
- *CONFIG_FREERTOS_PLACE_SNAPSHOT_FUNS_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_ENABLE_STATIC_TASK_CLEAN_UP

Enable static task clean up hook

Found in: Component config > FreeRTOS > Port

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) if *CONFIG_FREERTOS_SMP*

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 ESP_COREDUMP_DATA_FORMAT_ELF
- from 1536 to 32768

Default value:

- 2096 if 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) (FREERTOS_CORETIMER_0)
  Select this to use timer 0
- Timer 1 (int 15, level 3) (FREERTOS_CORETIMER_1)
  Select this to use timer 1
- SYSTIMER 0 (level 1) (FREERTOS_CORETIMER_SYSTIMER_LVL1)
  Select this to use systimer with the 1 interrupt priority.
- SYSTIMER 0 (level 3) (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 (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 (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_PLACE_SNAPSHOT_FUNS_INTO_FLASH

Place task snapshot functions into flash

Found in: Component config > FreeRTOS > Port

When enabled, the functions related to snapshots, such as vTaskGetSnapshot or uxTaskGetSnapshotAll, will be placed in flash. Note that if enabled, these functions cannot be called when cache is disabled.

Default value:

- No (disabled) if *CONFIG_FREERTOS_ENABLE_TASK_SNAPSHOT* && *CONFIG_ESP_PANIC_HANDLER_IRAM*

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)

CONFIG_FREERTOS_ENABLE_TASK_SNAPSHOT

Enable task snapshot functions

Found in: Component config > FreeRTOS > Port

When enabled, the functions related to snapshots, such as vTaskGetSnapshot or uxTaskGetSnapshotAll, are compiled and linked. Task snapshots are used by Task Watchdog (TWDT), GDB Stub and Core dump.

Default value:

- Yes (enabled)

Hardware Abstraction Layer (HAL) and Low Level (LL)

Contains:

- *CONFIG_HAL_DEFAULT_ASSERTION_LEVEL*
- *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 (HAL_ASSERTION_EQUALS_SYSTEM)
- Disabled (HAL_ASSERTION_DISABLE)
- Silent (HAL_ASSERTION_SILENT)
- Enabled (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 (HAL_LOG_LEVEL_NONE)
- Error (HAL_LOG_LEVEL_ERROR)
- Warning (HAL_LOG_LEVEL_WARN)
- Info (HAL_LOG_LEVEL_INFO)
- Debug (HAL_LOG_LEVEL_DEBUG)
- Verbose (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)

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) (HEAP_POISONING_DISABLED)
- Light impact (HEAP_POISONING_LIGHT)
- Comprehensive (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 (HEAP_TRACING_OFF)
- Standalone (HEAP_TRACING_STANDALONE)
- Host-based (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.

Keeping this as “n” in your project will save RAM and heap memory but will lower the performance of the heap trace in adding, retrieving and removing trace records. Making this as “y” in your project, you will decrease free RAM and heap memory but, the heap trace performances in adding retrieving and removing trace records will be enhanced.

Default value:

- No (disabled) if HEAP_TRACING_STANDALONE

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. The bigger this number is, the bigger the hash map will be in the memory. In case the tracing mode is set to HEAP_TRACE_ALL, the bigger the hashmap is, the better the performances are.

Range:
- from 1 to 10000 if *CONFIG_HEAP_TRACE_HASH_MAP*

Default value:
- 10 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)

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.

Default value:
- No (disabled)

IEEE 802.15.4 Contains:

- *CONFIG_IEEE802154_CCA_THRESHOLD*
- *CONFIG_IEEE802154_CCA_MODE*
- *CONFIG_IEEE802154_MULTI_PAN_ENABLE*
- *CONFIG_IEEE802154_TIMING_OPTIMIZATION*
- *CONFIG_IEEE802154_PENDING_TABLE_SIZE*
- *CONFIG_IEEE802154_RX_BUFFER_SIZE*

CONFIG_IEEE802154_RX_BUFFER_SIZE

The number of 802.15.4 receive buffers

Found in: Component config > IEEE 802.15.4

The number of 802.15.4 receive buffers

CONFIG_IEEE802154_CCA_MODE

Clear Channel Assessment (CCA) mode

Found in: Component config > IEEE 802.15.4

configure the CCA mode

Available options:

- Carrier sense only (IEEE802154_CCA_CARRIER)
  configure the CCA mode to Energy above threshold
- Energy above threshold (IEEE802154_CCA_ED)
  configure the CCA mode to Energy above threshold
- Carrier sense OR energy above threshold (IEEE802154_CCA_CARRIER_OR_ED)
  configure the CCA mode to Carrier sense OR energy above threshold
- Carrier sense AND energy above threshold (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

set the CCA threshold, in dB

Range:

- from -120 to 0

Default value:

- “-60”

CONFIG_IEEE802154_PENDING_TABLE_SIZE

Pending table size

Found in: Component config > IEEE 802.15.4

set the pending table size

Range:

- from 1 to 100

Default value:

- 20

CONFIG_IEEE802154_MULTI_PAN_ENABLE

Enable multi-pan feature for frame filter

Found in: Component config > IEEE 802.15.4

Enable IEEE802154 multi-pan

Default value:

- No (disabled)

CONFIG_IEEE802154_TIMING_OPTIMIZATION

Enable throughput optimization

Found in: Component config > IEEE 802.15.4

Enabling this option increases throughput by ~5% at the expense of ~2.1k IRAM code size increase.

Default value:

- No (disabled)

Log output Contains:

- *CONFIG_LOG_DEFAULT_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 (LOG_DEFAULT_LEVEL_NONE)
- Error (LOG_DEFAULT_LEVEL_ERROR)
- Warning (LOG_DEFAULT_LEVEL_WARN)
- Info (LOG_DEFAULT_LEVEL_INFO)
- Debug (LOG_DEFAULT_LEVEL_DEBUG)
- Verbose (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 (LOG_MAXIMUM_EQUALS_DEFAULT)
- Error (LOG_MAXIMUM_LEVEL_ERROR)
- Warning (LOG_MAXIMUM_LEVEL_WARN)
- Info (LOG_MAXIMUM_LEVEL_INFO)
- Debug (LOG_MAXIMUM_LEVEL_DEBUG)
- Verbose (LOG_MAXIMUM_LEVEL_VERBOSE)

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 (LOG_TIMESTAMP_SOURCE_RTOS)
- System Time (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*
- *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_STATS*
- *CONFIG_LWIP_TIMERS_ONDEMAND*
- *CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES*

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_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_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_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_MAX_SOCKETS

Max number of open sockets

Found in: Component config > LWIP

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 16.

Range:

- from 1 to 16

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_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 (new/experimental)

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_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 64 if *CONFIG_LWIP_WND_SCALE*
- 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
- 108

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_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 (coap and 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_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:

- 6
- 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 65535 if *CONFIG_LWIP_WND_SCALE*
- from 2440 to 1024000 if *CONFIG_LWIP_WND_SCALE*

Default value:

- 5744

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 65535 if *CONFIG_LWIP_WND_SCALE*
- from 2440 to 1024000 if *CONFIG_LWIP_WND_SCALE*

Default value:

- 5744

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 receiv 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 64 if *CONFIG_LWIP_WND_SCALE*
- from 6 to 1024 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_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 (LWIP_TCP_OVERSIZE_MSS)
- 25% MSS (LWIP_TCP_OVERSIZE_QUARTER_MSS)
- Disabled (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 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:

- 3000

- 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 receiv 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 (LWIP_TCPIP_TASK_AFFINITY_NO_AFFINITY)
- CPU0 (LWIP_TCPIP_TASK_AFFINITY_CPU0)
- CPU1 (LWIP_TCPIP_TASK_AFFINITY_CPU1)

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_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_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_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 aquired via DHCP

Found in: Component config > LWIP > SNTP > CONFIG_LWIP_DHCP_GET_NTP_SRV

Set maximum number of NTP servers aquired 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_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.

Default value:

- Yes (enabled) if COMPILER_OPTIMIZATION_ASSERTIONS_DISABLE

Hooks Contains:

- *CONFIG_LWIP_HOOK_ND6_GET_GW*
- *CONFIG_LWIP_HOOK_IP6_INPUT*
- *CONFIG_LWIP_HOOK_IP6_ROUTE*
- *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 (LWIP_HOOK_TCP_ISN_NONE)
- Default implementation (LWIP_HOOK_TCP_ISN_DEFAULT)
- Custom implementation (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 (LWIP_HOOK_IP6_ROUTE_NONE)
- Default (weak) implementation (LWIP_HOOK_IP6_ROUTE_DEFAULT)
- Custom implementation (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 (LWIP_HOOK_ND6_GET_GW_NONE)
- Default (weak) implementation (LWIP_HOOK_ND6_GET_GW_DEFAULT)
- Custom implementation (LWIP_HOOK_ND6_GET_GW_CUSTOM)

CONFIG_LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE

Netconn external resolve Hook

Found in: Component config > LWIP > Hooks

Enables custom DNS resolve 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 (LWIP_HOOK_NETCONN_EXT_RESOLVE_NONE)
- Default (weak) implementation (LWIP_HOOK_NETCONN_EXT_RESOLVE_DEFAULT)
- Custom implementation (LWIP_HOOK_NETCONN_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 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 (LWIP_HOOK_IP6_INPUT_NONE)
- Default (weak) implementation (LWIP_HOOK_IP6_INPUT_DEFAULT)
- Custom implementation (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_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_THREADING_C*
- *CONFIG_MBEDTLS_LARGE_KEY_SOFTWARE_MPI*
- *CONFIG_MBEDTLS_HKDF_C*
- *mbedTLS v3.x related*
- *CONFIG_MBEDTLS_MEM_ALLOC_MODE*
- *CONFIG_MBEDTLS_ECP_NIST_OPTIM*
- *CONFIG_MBEDTLS_POLY1305_C*
- *CONFIG_MBEDTLS_SECURITY_RISKS*
- *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_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 (MBEDTLS_INTERNAL_MEM_ALLOC)
- External SPIRAM (MBEDTLS_EXTERNAL_MEM_ALLOC)
- Default alloc mode (MBEDTLS_DEFAULT_MEM_ALLOC)
- Custom alloc mode (MBEDTLS_CUSTOM_MEM_ALLOC)
- Internal IRAM (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).

Range:

- from 512 to 16384

Default value:

- 16384

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.

Default value:
- No (disabled) if *CONFIG_MBEDTLS_SSL_PROTO_DTLS* && *CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH*

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:

Becasue 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 (MBEDTLS_DEBUG_LEVEL_WARN)
- Info (MBEDTLS_DEBUG_LEVEL_INFO)
- Debug (MBEDTLS_DEBUG_LEVEL_DEBUG)
- Verbose (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_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

Default value:

- No (disabled) if *CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE* && *CONFIG_MBEDTLS_DYNAMIC_BUFFER*

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) if MBEDTLS_DYNAMIC_FREE_PEER_CERT

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)

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_CID_PADDING_GRANULARITY

Record plaintext padding (for DTLS 1.2)

Found in: Component config > mbedTLS > mbedTLS v3.x related > DTLS-based configurations > CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID

Controls the use of record plaintext padding 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_DTLS_CONNECTION_ID* && *CONFIG_MBEDTLS_SSL_PROTO_DTLS*

Default value:

- 16 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 (MBEDTLS_CERTIFICATE_BUNDLE_DEFAULT_FULL)
- Use only the most common certificates from the default bundles (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 (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_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.

Default value:

- Yes (enabled) if SPIRAM_CACHE_WORKAROUND_STRATEGY_DUPLDST && SOC_AES_SUPPORTED

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) if *CONFIG_MBEDTLS_HARDWARE_AES*

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) if SOC_AES_SUPPORT_GCM && *CONFIG_MBEDTLS_HARDWARE_AES*

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.

Default value:

- Yes (enabled) if SPIRAM_CACHE_WORKAROUND_STRATEGY_DUPLDST && SOC_MPI_SUPPORTED

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) if *CONFIG_MBEDTLS_HARDWARE_MPI*

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.

Default value:

- Yes (enabled) if SPIRAM_CACHE_WORKAROUND_STRATEGY_DUPLDST

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

Default value:

- No (disabled) if SOC_ECDSA_SUPPORTED

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.

Default value:

- Yes (enabled) if SOC_ECDSA_SUPPORTED

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 (integrated with ESP32-WROOM-32SE)

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 (integrated with ESP32-WROOM-32SE)

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_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 (MBEDTLS_TLS_SERVER_AND_CLIENT)
- Server (MBEDTLS_TLS_SERVER_ONLY)
- Client (MBEDTLS_TLS_CLIENT_ONLY)
- None (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 Certifificate 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 Certifificate 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.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_SECP224R1_ENABLED

Enable SECP224R1 curve

Found in: Component config > mbedTLS

Enable support for SECP224R1 Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

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.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_SECP521R1_ENABLED

Enable SECP521R1 curve

Found in: Component config > mbedTLS

Enable support for SECP521R1 Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_SECP192K1_ENABLED

Enable SECP192K1 curve

Found in: Component config > mbedTLS

Enable support for SECP192K1 Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_SECP224K1_ENABLED

Enable SECP224K1 curve

Found in: Component config > mbedTLS

Enable support for SECP224K1 Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_SECP256K1_ENABLED

Enable SECP256K1 curve

Found in: Component config > mbedTLS

Enable support for SECP256K1 Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_BP256R1_ENABLED

Enable BP256R1 curve

Found in: Component config > mbedTLS

support for DP Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_BP384R1_ENABLED

Enable BP384R1 curve

Found in: Component config > mbedTLS

support for DP Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_BP512R1_ENABLED

Enable BP512R1 curve

Found in: Component config > mbedTLS

support for DP Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

CONFIG_MBEDTLS_ECP_DP_CURVE25519_ENABLED

Enable CURVE25519 curve

Found in: Component config > mbedTLS

Enable support for CURVE25519 Elliptic Curve.

Default value:

- Yes (enabled) if (*CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN* || *CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY*) && *CONFIG_MBEDTLS_ECP_C*

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.

end of Elliptic Curve options

Default value:

- Yes (enabled)

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_LARGE_KEY_SOFTWARE_MPI

Fallback to software implementation for larger MPI values

Found in: Component config > mbedTLS

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:

- Yes (enabled) if *CONFIG_MBEDTLS_HARDWARE_MPI*
- No (disabled) if *CONFIG_MBEDTLS_HARDWARE_MPI*

CONFIG_MBEDTLS_SECURITY_RISKS

Show configurations with potential security risks

Found in: Component config > mbedTLS

Default value:

- No (disabled)

Contains:

- *CONFIG_MBEDTLS_ALLOW_UNSUPPORTED_CRITICAL_EXT*

CONFIG_MBEDTLS_ALLOW_UNSUPPORTED_CRITICAL_EXT

X.509 CRT parsing with unsupported critical extensions

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_SECURITY_RISKS

Allow the X.509 certificate parser to load certificates with unsupported critical extensions

Default value:

- No (disabled) if *CONFIG_MBEDTLS_SECURITY_RISKS*

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_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_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 (MQTT_USE_CORE_0)
- Core 1 (MQTT_USE_CORE_1)

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 occurence of LF is replaced with CR

This option doesn't affect behavior of the UART driver (drivers/uart.h).

Available options:

- CRLF (NEWLIB_STDOUT_LINE_ENDING_CRLF)
- LF (NEWLIB_STDOUT_LINE_ENDING_LF)
- CR (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 occurence of CR is replaced with LF

This option doesn't affect behavior of the UART driver (drivers/uart.h).

Available options:

- CRLF (NEWLIB_STDIN_LINE_ENDING_CRLF)
- LF (NEWLIB_STDIN_LINE_ENDING_LF)
- CR (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/newlib/README

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.

Default value:

- Yes (enabled)

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.
- **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 (NEWLIB_TIME_SYSCALL_USE_RTC_HRT)
- RTC (NEWLIB_TIME_SYSCALL_USE_RTC)
- High-resolution timer (NEWLIB_TIME_SYSCALL_USE_HRT)
- None (NEWLIB_TIME_SYSCALL_USE_NONE)

NVS Contains:

- *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, AES-XTS is used to encrypt the complete NVS data, except the page headers. It requires XTS encryption keys to be stored in an encrypted partition. This means enabling flash encryption is a pre-requisite for this feature.

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)

OpenThread Contains:

- *CONFIG_OPENTHREAD_BORDER_ROUTER*
- *CONFIG_OPENTHREAD_COMMISSIONER*
- *CONFIG_OPENTHREAD_CSL_ENABLE*
- *CONFIG_OPENTHREAD_DIAG*
- *CONFIG_OPENTHREAD_DNS_CLIENT*
- *CONFIG_OPENTHREAD_JOINER*
- *CONFIG_OPENTHREAD_LINK_METRICS*
- *CONFIG_OPENTHREAD_MACFILTER_ENABLE*
- *CONFIG_OPENTHREAD_CLI*
- *CONFIG_OPENTHREAD_SRP_CLIENT*
- *CONFIG_OPENTHREAD_ENABLED*
- *CONFIG_OPENTHREAD_NUM_MESSAGE_BUFFERS*
- *CONFIG_OPENTHREAD_RCP_TRANSPORT*
- *CONFIG_OPENTHREAD_UART_BUFFER_SIZE*
- *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_LOG_LEVEL

OpenThread log verbosity

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED

Select OpenThread log level.

Available options:

- No logs (OPENTHREAD_LOG_LEVEL_NONE)
- Error logs (OPENTHREAD_LOG_LEVEL_CRIT)
- Warning logs (OPENTHREAD_LOG_LEVEL_WARN)
- Notice logs (OPENTHREAD_LOG_LEVEL_NOTE)
- Info logs (OPENTHREAD_LOG_LEVEL_INFO)
- Debug logs (OPENTHREAD_LOG_LEVEL_DEBG)

CONFIG_OPENTHREAD_RADIO_TYPE

Config the Thread radio type

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED

Configure how OpenThread connects to the 15.4 radio

Available options:

- Native 15.4 radio (OPENTHREAD_RADIO_NATIVE)
  Select this to use the native 15.4 radio.
- Connect via UART (OPENTHREAD_RADIO_SPINEL_UART)
  Select this to connect to a Radio Co-Processor via UART.
- Connect via SPI (OPENTHREAD_RADIO_SPINEL_SPI)
  Select this to connect to a Radio Co-Processor via SPI.

CONFIG_OPENTHREAD_DEVICE_TYPE

Config the Thread device type

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED

OpenThread can be configured to different device types (FTD, MTD, Radio)

Available options:

- Full Thread Device (OPENTHREAD_FTD)
  Select this to enable Full Thread Device which can act as router and leader in a Thread network.
- Minimal Thread Device (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 (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 (OPENTHREAD_RCP_UART)
  Select this to enable UART connection to host.
- SPI RCP (OPENTHREAD_RCP_SPI)
  Select this to enable SPI connection to host.

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_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_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*

CONFIG_OPENTHREAD_NUM_MESSAGE_BUFFERS

The number of openthread message buffers

Found in: Component config > OpenThread

Range:
- from 50 to 100 if *CONFIG_OPENTHREAD_ENABLED*

Default value:
- 65 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.

Range:
- from 128 to 1024 if *CONFIG_OPENTHREAD_ENABLED*

Default value:
- 256 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_CSL_DEBUG_ENABLE

Enable CSL debug

Found in: Component config > OpenThread > CONFIG_OPENTHREAD_CSL_ENABLE

Select this option to set rx on when sleep in CSL feature, only for debug

Default value:

- No (disabled) if *CONFIG_OPENTHREAD_CSL_ENABLE*

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 (PTHREAD_DEFAULT_CORE_NO_AFFINITY)
- Core 0 (PTHREAD_DEFAULT_CORE_0)
- Core 1 (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

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*
- *SPI Flash behavior when brownout*
- *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.

Default value:

- No (disabled) if *CONFIG_SPI_FLASH_ROM_IMPL* && *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- No (disabled) if *CONFIG_SPI_FLASH_VERIFY_WRITE* && *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- No (disabled) if *CONFIG_SPI_FLASH_VERIFY_WRITE* && *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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)

Default value:

- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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 (SPI_FLASH_DANGEROUS_WRITE_ABORTS)
- Fails (SPI_FLASH_DANGEROUS_WRITE_FAILS)
- Allowed (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.

Default value:

- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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 documenation under section *OS Function*.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- 20 if *CONFIG_SPI_FLASH_YIELD_DURING_ERASE* && *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- 1 if *CONFIG_SPI_FLASH_YIELD_DURING_ERASE* && *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Range:

- from 256 to 8192 if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

Default value:

- 8192 if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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 > SPI Flash driver > 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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*
- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*
- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*
- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*
- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*
- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*
- No (disabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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.

Default value:

- Yes (enabled) if *CONFIG_APP_BUILD_TYPE_PURE_RAM_APP*

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 mesages 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 mesages 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 mesages 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 mesages 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 mesages 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_COPROC_ENABLED*
- *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) if SOC_ULP_SUPPORTED || SOC_RISCV_COPROC_SUPPORTED || SOC_LP_CORE_SUPPORTED

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) (ULP_COPROC_TYPE_FSM)
- ULP RISC-V (ULP_COPROC_TYPE_RISCV)
- LP core RISC-V (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 8176 if *CONFIG_ULP_COPROC_ENABLED* && (SOC_ULP_SUPPORTED || SOC_RISCV_COPROC_SUPPORTED || SOC_LP_CORE_SUPPORTED)

Default value:

- 4096 if *CONFIG_ULP_COPROC_ENABLED* && (SOC_ULP_SUPPORTED || SOC_RISCV_COPROC_SUPPORTED || SOC_LP_CORE_SUPPORTED)

ULP RISC-V Settings Contains:

- *CONFIG_ULP_RISCV_UART_BAUDRATE*
- *CONFIG_ULP_RISCV_I2C_RW_TIMEOUT*

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 ULP_COPROC_TYPE_RISCV && (SOC_ULP_SUPPORTED || SOC_RISCV_COPROC_SUPPORTED || SOC_LP_CORE_SUPPORTED)

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 ULP_COPROC_TYPE_RISCV && (SOC_ULP_SUPPORTED || SOC_RISCV_COPROC_SUPPORTED || SOC_LP_CORE_SUPPORTED)

Default value:

- 500 if ULP_COPROC_TYPE_RISCV && (SOC_ULP_SUPPORTED || SOC_RISCV_COPROC_SUPPORTED || SOC_LP_CORE_SUPPORTED)

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)

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.

Default value:

- Yes (enabled) if *CONFIG_VFS_SUPPORT_IO* && *CONFIG_LWIP_USE_ONLY_LWIP_SELECT*

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 unconveniently 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_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)

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 (WL_SECTOR_SIZE_512)
- 4096 (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:
- Perfomance (WL_SECTOR_MODE_PERF)
- Safety (WL_SECTOR_MODE_SAFE)

Wi-Fi Provisioning Manager

Contains:

- *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 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_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 (WIFI_PROV_STA_ALL_CHANNEL_SCAN)
  Scan will end after scanning the entire channel. This option is useful in Mesh WiFi Systems.
- Fast Scan (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.

Default value:

- No (disabled)

Deprecated options and their replacements

- **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_BROWNOUT_DET_LVL_SEL_4
  - CONFIG_BROWNOUT_DET_LVL_SEL_3
  - CONFIG_BROWNOUT_DET_LVL_SEL_2
- **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_MSYS1_BLOCK_COUNT (*CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT*)
- CONFIG_BT_NIMBLE_TASK_STACK_SIZE (*CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE*)
- **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_PHY_CALIBRATION_AND_DATA_STORAGE (*CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE*)
- CONFIG_ESP32_PHY_DEFAULT_INIT_IF_INVALID (*CONFIG_ESP_PHY_DEFAULT_INIT_IF_INVALID*)
- CONFIG_ESP32_PHY_INIT_DATA_ERROR (*CONFIG_ESP_PHY_INIT_DATA_ERROR*)
- CONFIG_ESP32_PHY_INIT_DATA_IN_PARTITION (*CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*)
- CONFIG_ESP32_PHY_MAC_BB_PD (*CONFIG_ESP_PHY_MAC_BB_PD*)
- CONFIG_ESP32_PHY_MAX_WIFI_TX_POWER (*CONFIG_ESP_PHY_MAX_WIFI_TX_POWER*)
- 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_REDUCE_PHY_TX_POWER (*CONFIG_ESP_PHY_REDUCE_TX_POWER*)
- CONFIG_ESP32_RTC_XTAL_BOOTSTRAP_CYCLES (*CONFIG_ESP_SYSTEM_RTC_EXT_XTAL_BOOTSTRAP_CYCLES*)
- CONFIG_ESP32_SUPPORT_MULTIPLE_PHY_INIT_DATA_BIN (*CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN*)
- 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_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_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_MAC_BB_PD (*CONFIG_ESP_PHY_MAC_BB_PD*)
- 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_ACL_BUF_COUNT (*CONFIG_BT_NIMBLE_ACL_BUF_COUNT*)
- CONFIG_NIMBLE_ACL_BUF_SIZE (*CONFIG_BT_NIMBLE_ACL_BUF_SIZE*)
- 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_HCI_EVT_BUF_SIZE (*CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE*)
- CONFIG_NIMBLE_HCI_EVT_HI_BUF_COUNT (*CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT*)

- CONFIG_NIMBLE_HCI_EVT_LO_BUF_COUNT (*CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT*)
- 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_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_REDUCE_PHY_TX_POWER (*CONFIG_ESP_PHY_REDUCE_TX_POWER*)
- **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_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*)

2.8 配网 API

2.8.1 Protocol Communication

Overview

Protocol Communication (protocomm) component manages secure sessions and provides framework for multiple transports. The application can also use protocomm layer directly to have application specific extensions for the provisioning (or non-provisioning) use cases.

Following features are available for provisioning :

- **Communication security at application level -**
  - protocomm_security0 (no security)
  - protocomm_security1 (Curve25519 key exchange + AES-CTR encryption/decryption)
  - protocomm_security2 (SRP6a-based key exchange + AES-GCM encryption/decryption)
- Proof-of-possession (support with protocomm_security1 only)
- Salt and Verifier (support with protocomm_security2 only)

Protocomm internally uses protobuf (protocol buffers) for secure session establishment. Though users can implement their own security (even without using protobuf). One can even use protocomm without any security layer.

Protocomm provides framework for various transports :

- BLE
- WiFi (SoftAP+HTTPD)
- console, in which case the handler invocation is automatically taken care of on the device side (see Transport Examples below for code snippets).

Note that the client still needs to establish session (for protocomm_security1 and protocomm_security2) by performing the two way handshake. See *Unified Provisioning* for more details about the secure handshake logic.

Enabling protocomm security version

Protocomm component provides project configuration menu to enable/disable support of respective security versions. The respective configuration options can be found as follows:

- Support protocomm security version 0 (no security): *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0* (this option is enabled by default)
- Support protocomm security version 1 (Curve25519 key exchange + AES-CTR encryption/decryption): *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1* (this option is enabled by default)
- Support protocomm security version 2 (SRP6a-based key exchange + AES-GCM encryption/decryption): *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2*

备注： Enabling multiple security versions allow to control them dynamically but also increases firmware size.

Transport Example (SoftAP + HTTP) with Security 2

For sample usage, see wifi_provisioning/src/scheme_softap.c

```
/* Endpoint handler to be registered with protocomm.
 * This simply echoes back the received data. */
esp_err_t echo_req_handler (uint32_t session_id,
                            const uint8_t *inbuf, ssize_t inlen,
```

(下页继续)

(续上页)

```
                            uint8_t **outbuf, ssize_t *outlen,
                            void *priv_data)
{
    /* Session ID may be used for persistence */
    printf("Session ID : %d", session_id);

    /* Echo back the received data */
    *outlen = inlen;            /* Output data length updated */
    *outbuf = malloc(inlen);    /* This will be deallocated outside */
    memcpy(*outbuf, inbuf, inlen);

    /* Private data that was passed at the time of endpoint creation */
    uint32_t *priv = (uint32_t *) priv_data;
    if (priv) {
        printf("Private data : %d", *priv);
    }

    return ESP_OK;
}

static const char sec2_salt[] = {0xf7, 0x5f, 0xe2, 0xbe, 0xba, 0x7c, 0x81, 0xcd};
static const char sec2_verifier[] = {0xbf, 0x86, 0xce, 0x63, 0x8a, 0xbb, 0x7e,␣
↪0x2f, 0x38, 0xa8, 0x19, 0x1b, 0x35,
    0xc9, 0xe3, 0xbe, 0xc3, 0x2b, 0x45, 0xee, 0x10, 0x74, 0x22, 0x1a, 0x95, 0xbe,␣
↪0x62, 0xf7, 0x0c, 0x65, 0x83, 0x50,
    0x08, 0xef, 0xaf, 0xa5, 0x94, 0x4b, 0xcb, 0xe1, 0xce, 0x59, 0x2a, 0xe8, 0x7b,␣
↪0x27, 0xc8, 0x72, 0x26, 0x71, 0xde,
    0xb2, 0xf2, 0x80, 0x02, 0xdd, 0x11, 0xf0, 0x38, 0x0e, 0x95, 0x25, 0x00, 0xcf,␣
↪0xb3, 0x3f, 0xf0, 0x73, 0x2a, 0x25,
    0x03, 0xe8, 0x51, 0x72, 0xef, 0x6d, 0x3e, 0x14, 0xb9, 0x2e, 0x9f, 0x2a, 0x90,␣
↪0x9e, 0x26, 0xb6, 0x3e, 0xc7, 0xe4,
    0x9f, 0xe3, 0x20, 0xce, 0x28, 0x7c, 0xbf, 0x89, 0x50, 0xc9, 0xb6, 0xec, 0xdd,␣
↪0x81, 0x18, 0xf1, 0x1a, 0xd9, 0x7a,
    0x21, 0x99, 0xf1, 0xee, 0x71, 0x2f, 0xcc, 0x93, 0x16, 0x34, 0x0c, 0x79, 0x46,␣
↪0x23, 0xe4, 0x32, 0xec, 0x2d, 0x9e,
    0x18, 0xa6, 0xb9, 0xbb, 0x0a, 0xcf, 0xc4, 0xa8, 0x32, 0xc0, 0x1c, 0x32, 0xa3,␣
↪0x97, 0x66, 0xf8, 0x30, 0xb2, 0xda,
    0xf9, 0x8d, 0xc3, 0x72, 0x72, 0x5f, 0xe5, 0xee, 0xc3, 0x5c, 0x24, 0xc8, 0xdd,␣
↪0x54, 0x49, 0xfc, 0x12, 0x91, 0x81,
    0x9c, 0xc3, 0xac, 0x64, 0x5e, 0xd6, 0x41, 0x88, 0x2f, 0x23, 0x66, 0xc8, 0xac,␣
↪0xb0, 0x35, 0x0b, 0xf6, 0x9c, 0x88,
    0x6f, 0xac, 0xe1, 0xf4, 0xca, 0xc9, 0x07, 0x04, 0x11, 0xda, 0x90, 0x42, 0xa9,␣
↪0xf1, 0x97, 0x3d, 0x94, 0x65, 0xe4,
    0xfb, 0x52, 0x22, 0x3b, 0x7a, 0x7b, 0x9e, 0xe9, 0xee, 0x1c, 0x44, 0xd0, 0x73,␣
↪0x72, 0x2a, 0xca, 0x85, 0x19, 0x4a,
    0x60, 0xce, 0x0a, 0xc8, 0x7d, 0x57, 0xa4, 0xf8, 0x77, 0x22, 0xc1, 0xa5, 0xfa,␣
↪0xfb, 0x7b, 0x91, 0x3b, 0xfe, 0x87,
    0x5f, 0xfe, 0x05, 0xd2, 0xd6, 0xd3, 0x74, 0xe5, 0x2e, 0x68, 0x79, 0x34, 0x70,␣
↪0x40, 0x12, 0xa8, 0xe1, 0xb4, 0x6c,
    0xaa, 0x46, 0x73, 0xcd, 0x8d, 0x17, 0x72, 0x67, 0x32, 0x42, 0xdc, 0x10, 0xd3,␣
↪0x71, 0x7e, 0x8b, 0x00, 0x46, 0x9b,
    0x0a, 0xe9, 0xb4, 0x0f, 0xeb, 0x70, 0x52, 0xdd, 0x0a, 0x1c, 0x7e, 0x2e, 0xb0,␣
↪0x61, 0xa6, 0xe1, 0xa3, 0x34, 0x4b,
    0x2a, 0x3c, 0xc4, 0x5d, 0x42, 0x05, 0x58, 0x25, 0xd3, 0xca, 0x96, 0x5c, 0xb9,␣
↪0x52, 0xf9, 0xe9, 0x80, 0x75, 0x3d,
    0xc8, 0x9f, 0xc7, 0xb2, 0xaa, 0x95, 0x2e, 0x76, 0xb3, 0xe1, 0x48, 0xc1, 0x0a,␣
↪0xa1, 0x0a, 0xe8, 0xaf, 0x41, 0x28,
    0xd2, 0x16, 0xe1, 0xa6, 0xd0, 0x73, 0x51, 0x73, 0x79, 0x98, 0xd9, 0xb9, 0x00,␣
↪0x50, 0xa2, 0x4d, 0x99, 0x18, 0x90,
    0x70, 0x27, 0xe7, 0x8d, 0x56, 0x45, 0x34, 0x1f, 0xb9, 0x30, 0xda, 0xec, 0x4a,␣
↪0x08, 0x27, 0x9f, 0xfa, 0x59, 0x2e,
```

(下页继续)

(续上页)

```
    0x36, 0x77, 0x00, 0xe2, 0xb6, 0xeb, 0xd1, 0x56, 0x50, 0x8e};

/* Example function for launching a protocomm instance over HTTP */
protocomm_t *start_pc()
{
    protocomm_t *pc = protocomm_new();


    /* Config for protocomm_httpd_start() */
    protocomm_httpd_config_t pc_config = {
        .data = {
        .config = PROTOCOMM_HTTPD_DEFAULT_CONFIG()
        }
    };

    /* Start protocomm server on top of HTTP */
    protocomm_httpd_start(pc, &pc_config);

    /* Create Security2 params object from salt and verifier. It must be valid
     * throughout the scope of protocomm endpoint. This need not be static,
     * ie. could be dynamically allocated and freed at the time of endpoint
     * removal */
    const static protocomm_security2_params_t sec2_params = {
        .salt = (const uint8_t *) salt,
        .salt_len = sizeof(salt),
        .verifier = (const uint8_t *) verifier,
        .verifier_len = sizeof(verifier),
    };

    /* Set security for communication at application level. Just like for
     * request handlers, setting security creates an endpoint and registers
     * the handler provided by protocomm_security1. One can similarly use
     * protocomm_security0. Only one type of security can be set for a
     * protocomm instance at a time. */
    protocomm_set_security(pc, "security_endpoint", &protocomm_security2, &sec2_
↪params);

    /* Private data passed to the endpoint must be valid throughout the scope
     * of protocomm endpoint. This need not be static, ie. could be dynamically
     * allocated and freed at the time of endpoint removal */
    static uint32_t priv_data = 1234;

    /* Add a new endpoint for the protocomm instance, identified by a unique name
     * and register a handler function along with private data to be passed at the
     * time of handler execution. Multiple endpoints can be added as long as they
     * are identified by unique names */
    protocomm_add_endpoint(pc, "echo_req_endpoint",
                           echo_req_handler, (void *) &priv_data);
    return pc;
}

/* Example function for stopping a protocomm instance */
void stop_pc(protocomm_t *pc)
{
    /* Remove endpoint identified by it's unique name */
    protocomm_remove_endpoint(pc, "echo_req_endpoint");

    /* Remove security endpoint identified by it's name */
    protocomm_unset_security(pc, "security_endpoint");

    /* Stop HTTP server */
```

(下页继续)

(续上页)

```
    protocomm_httpd_stop(pc);

    /* Delete (deallocate) the protocomm instance */
    protocomm_delete(pc);
}
```

Transport Example (SoftAP + HTTP) with Security 1

For sample usage, see wifi_provisioning/src/scheme_softap.c

```
/* Endpoint handler to be registered with protocomm.
 * This simply echoes back the received data. */
esp_err_t echo_req_handler (uint32_t session_id,
                            const uint8_t *inbuf, ssize_t inlen,
                            uint8_t **outbuf, ssize_t *outlen,
                            void *priv_data)
{
    /* Session ID may be used for persistence */
    printf("Session ID : %d", session_id);

    /* Echo back the received data */
    *outlen = inlen;            /* Output data length updated */
    *outbuf = malloc(inlen);    /* This will be deallocated outside */
    memcpy(*outbuf, inbuf, inlen);

    /* Private data that was passed at the time of endpoint creation */
    uint32_t *priv = (uint32_t *) priv_data;
    if (priv) {
        printf("Private data : %d", *priv);
    }

    return ESP_OK;
}

/* Example function for launching a protocomm instance over HTTP */
protocomm_t *start_pc(const char *pop_string)
{
    protocomm_t *pc = protocomm_new();


    /* Config for protocomm_httpd_start() */
    protocomm_httpd_config_t pc_config = {
        .data = {
        .config = PROTOCOMM_HTTPD_DEFAULT_CONFIG()
        }
    };

    /* Start protocomm server on top of HTTP */
    protocomm_httpd_start(pc, &pc_config);

    /* Create security1 params object from pop_string. It must be valid
     * throughout the scope of protocomm endpoint. This need not be static,
     * ie. could be dynamically allocated and freed at the time of endpoint
     * removal */
    const static protocomm_security1_params_t sec1_params = {
        .data = (const uint8_t *) strdup(pop_string),
        .len = strlen(pop_string)
    };

    /* Set security for communication at application level. Just like for
```

(下页继续)

(续上页)

```
     * request handlers, setting security creates an endpoint and registers
     * the handler provided by protocomm_security1. One can similarly use
     * protocomm_security0. Only one type of security can be set for a
     * protocomm instance at a time. */
    protocomm_set_security(pc, "security_endpoint", &protocomm_security1, &sec1_
↪params);

    /* Private data passed to the endpoint must be valid throughout the scope
     * of protocomm endpoint. This need not be static, ie. could be dynamically
     * allocated and freed at the time of endpoint removal */
    static uint32_t priv_data = 1234;

    /* Add a new endpoint for the protocomm instance, identified by a unique name
     * and register a handler function along with private data to be passed at the
     * time of handler execution. Multiple endpoints can be added as long as they
     * are identified by unique names */
    protocomm_add_endpoint(pc, "echo_req_endpoint",
                           echo_req_handler, (void *) &priv_data);
    return pc;
}

/* Example function for stopping a protocomm instance */
void stop_pc(protocomm_t *pc)
{
    /* Remove endpoint identified by it's unique name */
    protocomm_remove_endpoint(pc, "echo_req_endpoint");

    /* Remove security endpoint identified by it's name */
    protocomm_unset_security(pc, "security_endpoint");

    /* Stop HTTP server */
    protocomm_httpd_stop(pc);

    /* Delete (deallocate) the protocomm instance */
    protocomm_delete(pc);
}
```

Transport Example (BLE) with Security 0

For sample usage, see wifi_provisioning/src/scheme_ble.c

```
/* Example function for launching a secure protocomm instance over BLE */
protocomm_t *start_pc()
{
    protocomm_t *pc = protocomm_new();

    /* Endpoint UUIDs */
    protocomm_ble_name_uuid_t nu_lookup_table[] = {
        {"security_endpoint", 0xFF51},
        {"echo_req_endpoint", 0xFF52}
    };

    /* Config for protocomm_ble_start() */
    protocomm_ble_config_t config = {
        .service_uuid = {
            /* LSB <---------------------------------------
            * ---------------------------------------> MSB */
            0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 0x00, 0x80,
            0x00, 0x10, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00,
        },
```

(下页继续)

(续上页)

```
        .nu_lookup_count = sizeof(nu_lookup_table)/sizeof(nu_lookup_table[0]),
        .nu_lookup = nu_lookup_table
    };

    /* Start protocomm layer on top of BLE */
    protocomm_ble_start(pc, &config);

    /* For protocomm_security0, Proof of Possession is not used, and can be kept␣
↪NULL */
    protocomm_set_security(pc, "security_endpoint", &protocomm_security0, NULL);
    protocomm_add_endpoint(pc, "echo_req_endpoint", echo_req_handler, NULL);
    return pc;
}

/* Example function for stopping a protocomm instance */
void stop_pc(protocomm_t *pc)
{
    protocomm_remove_endpoint(pc, "echo_req_endpoint");
    protocomm_unset_security(pc, "security_endpoint");

    /* Stop BLE protocomm service */
    protocomm_ble_stop(pc);

    protocomm_delete(pc);
}
```

API Reference

Header File

- components/protocomm/include/common/protocomm.h

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 con-

tents 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

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

Header File

- components/protocomm/include/security/protocomm_security0.h

Header File

- components/protocomm/include/security/protocomm_security1.h

Header File

- components/protocomm/include/transports/protocomm_httpd.h

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()

Header File

- components/protocomm/include/transports/protocomm_ble.h

Functions

esp_err_t **protocomm_ble_start**(*protocomm_t* *pc, const *protocomm_ble_config_t* *config)

Start Bluetooth Low Energy based transport layer for provisioning.

Initialize and start required BLE service for provisioning. This includes the initialization for characteristics/service for BLE.

参数

- **pc** –**[in]** Protocomm instance pointer obtained from protocomm_new()
- **config** –**[in]** Pointer to config structure for initializing BLE

返回

- ESP_OK : Success

- ESP_FAIL : Simple BLE start error
- ESP_ERR_NO_MEM : Error allocating memory for internal resources
- ESP_ERR_INVALID_STATE : Error in ble config
- ESP_ERR_INVALID_ARG : Null arguments

esp_err_t **protocomm_ble_stop**(*protocomm_t* *pc)

Stop Bluetooth Low Energy based transport layer for provisioning.

Stops service/task responsible for BLE based interactions for provisioning

---

备注: You might want to optionally reclaim memory from Bluetooth. Refer to the documentation of `esp_bt_mem_release` in that case.

参数 **pc** –**[in]** Same protocomm instance that was passed to protocomm_ble_start()

返回

- ESP_OK : Success
- ESP_FAIL : Simple BLE stop error
- ESP_ERR_INVALID_ARG : Null / incorrect protocomm instance

Structures

struct **name_uuid**

This structure maps handler required by protocomm layer to UUIDs which are used to uniquely identify BLE characteristics from a smartphone or a similar client device.

Public Members

const char ***name**

Name of the handler, which is passed to protocomm layer

uint16_t **uuid**

UUID to be assigned to the BLE characteristic which is mapped to the handler

struct **protocomm_ble_config**

Config parameters for protocomm BLE service.

Public Members

char **device_name**[MAX_BLE_DEVNAME_LEN + 1]

BLE device name being broadcast at the time of provisioning

uint8_t **service_uuid**[BLE_UUID128_VAL_LENGTH]

128 bit UUID of the provisioning service

uint8_t ***manufacturer_data**

BLE device manufacturer data pointer in advertisement

ssize_t **manufacturer_data_len**

BLE device manufacturer data length in advertisement

ssize_t **nu_lookup_count**

Number of entries in the Name-UUID lookup table

protocomm_ble_name_uuid_t ***nu_lookup**

Pointer to the Name-UUID lookup table

unsigned **ble_bonding**

BLE bonding

unsigned **ble_sm_sc**

BLE security flag

unsigned **ble_link_encryption**

BLE security flag

Macros

MAX_BLE_DEVNAME_LEN

BLE device name cannot be larger than this value 31 bytes (max scan response size) - 1 byte (length) - 1 byte (type) = 29 bytes

BLE_UUID128_VAL_LENGTH

MAX_BLE_MANUFACTURER_DATA_LEN

Theoretically, the limit for max manufacturer length remains same as BLE device name i.e. 31 bytes (max scan response size) - 1 byte (length) - 1 byte (type) = 29 bytes However, manufacturer data goes along with BLE device name in scan response. So, it is important to understand the actual length should be smaller than (29 - (BLE device name length) - 2).

Type Definitions

typedef struct *name_uuid* **protocomm_ble_name_uuid_t**

This structure maps handler required by protocomm layer to UUIDs which are used to uniquely identify BLE characteristics from a smartphone or a similar client device.

typedef struct *protocomm_ble_config* **protocomm_ble_config_t**

Config parameters for protocomm BLE service.

Enumerations

enum **protocomm_transport_ble_event_t**

Events generated by BLE transport.

These events are generated when the BLE transport is paired and disconnected.

Values:

enumerator **PROTOCOMM_TRANSPORT_BLE_CONNECTED**

enumerator **PROTOCOMM_TRANSPORT_BLE_DISCONNECTED**

2.8.2 Unified Provisioning

Overview

Unified provisioning support in the ESP-IDF provides an extensible mechanism to the developers to configure the device with the Wi-Fi credentials and/or other custom configuration using various transports and different security schemes. Depending on the use-case it provides a complete and ready solution for Wi-Fi network provisioning along with example iOS and Android applications. Or developers can extend the device-side and phone-app side implementations to accommodate their requirements for sending additional configuration data. Following are the important features of this implementation.

1. *Extensible Protocol:* The protocol is completely flexible and it offers the ability for the developers to send custom configuration in the provisioning process. The data representation too is left to the application to decide.
2. *Transport Flexibility:* The protocol can work on Wi-Fi (SoftAP + HTTP server) or on BLE as a transport protocol. The framework provides an ability to add support for any other transport easily as long as command-response behaviour can be supported on the transport.
3. *Security Scheme Flexibility:* It's understood that each use-case may require different security scheme to secure the data that is exchanged in the provisioning process. Some applications may work with SoftAP that's WPA2 protected or BLE with "just-works" security. Or the applications may consider the transport to be insecure and may want application level security. The unified provisioning framework allows application to choose the security as deemed suitable.
4. *Compact Data Representation:* The protocol uses Google Protobufs as a data representation for session setup and Wi-Fi provisioning. They provide a compact data representation and ability to parse the data in multiple programming languages in native format. Please note that this data representation is not forced on application specific data and the developers may choose the representation of their choice.

Typical Provisioning Process

Deciding on Transport

Unified provisioning subsystem supports Wi-Fi (SoftAP+HTTP server) and BLE (GATT based) transport schemes. Following points need to be considered while selecting the best possible transport for provisioning.

1. BLE based transport has an advantage that in the provisioning process, the BLE communication channel stays intact between the device and the client. That provides reliable provisioning feedback.
2. BLE based provisioning implementation makes the user-experience better from the phone apps as on Android and iOS both, the phone app can discover and connect to the device without requiring user to go out of the phone app
3. BLE transport however consumes ~110KB memory at runtime. If the product does not use the BLE or BT functionality after provisioning is done, almost all the memory can be reclaimed back and can be added into the heap.
4. SoftAP based transport is highly interoperable; however as the same radio is shared between SoftAP and Station interface, the transport is not reliable in the phase when the Wi-Fi connection to external AP is attempted. Also, the client may roam back to different network when the SoftAP changes the channel at the time of Station connection.
5. SoftAP transport does not require much additional memory for the Wi-Fi use-cases
6. SoftAP based provisioning requires the phone app user to go to "System Settings" to connect to Wi-Fi network hosted by the device in case of iOS. The discovery (scanning) as well as connection API is not available for the iOS applications.

Deciding on Security

Depending on the transport and other constraints the security scheme needs to be selected by the application developers. Following considerations need to be given from the provisioning security perspective: 1. The configuration data sent from the client to the device and the response has to be secured. 2. The client should authenticate the device it is connected to. 3. The device manufacturer may choose proof-of-possession - a unique per device secret to be

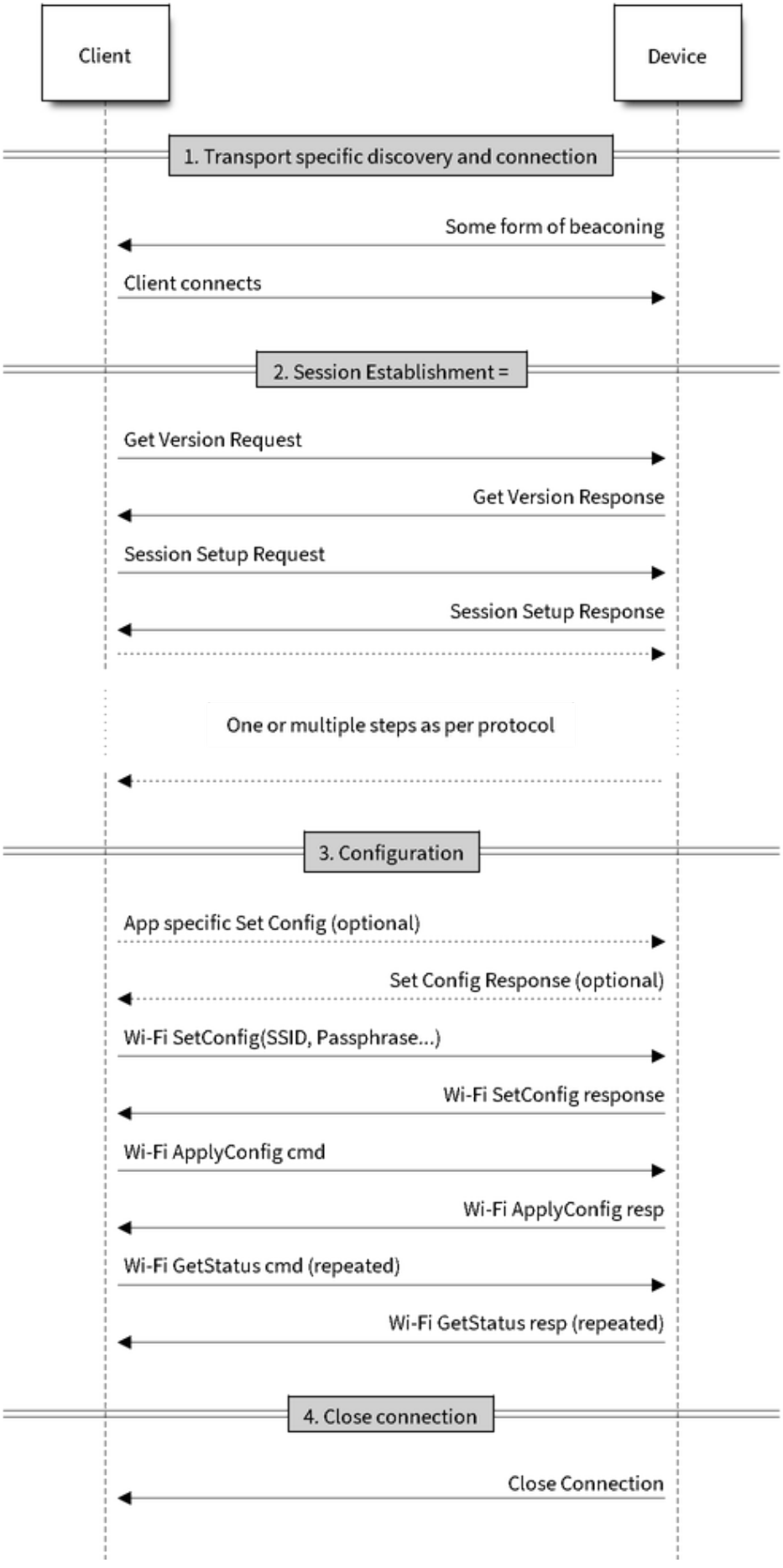


图 6: Typical Provisioning Process

entered on the provisioning client as a security measure to make sure that the user can provisions the device in the possession.

There are two levels of security schemes. The developer may select one or combination depending on requirements.

1. *Transport Security:* SoftAP provisioning may choose WPA2 protected security with unique per-device passphrase. Per-device unique passphrase can also act as a proof-of-possession. For BLE, "just-works" security can be used as a transport level security after understanding the level of security it provides.
2. *Application Security:* The unified provisioning subsystem provides application level security (*security1*) that provides data protection and authentication (through proof-of-possession) if the application does not use the transport level security or if the transport level security is not sufficient for the use-case.

Device Discovery

The advertisement and device discovery is left to the application and depending on the protocol chosen, the phone apps and device firmware application can choose appropriate method to advertise and discovery.

For the SoftAP+HTTP transport, typically the SSID (network name) of the AP hosted by the device can be used for discovery.

For the BLE transport device name or primary service included in the advertisement or combination of both can be used for discovery.

Architecture

The below diagram shows architecture of unified provisioning.

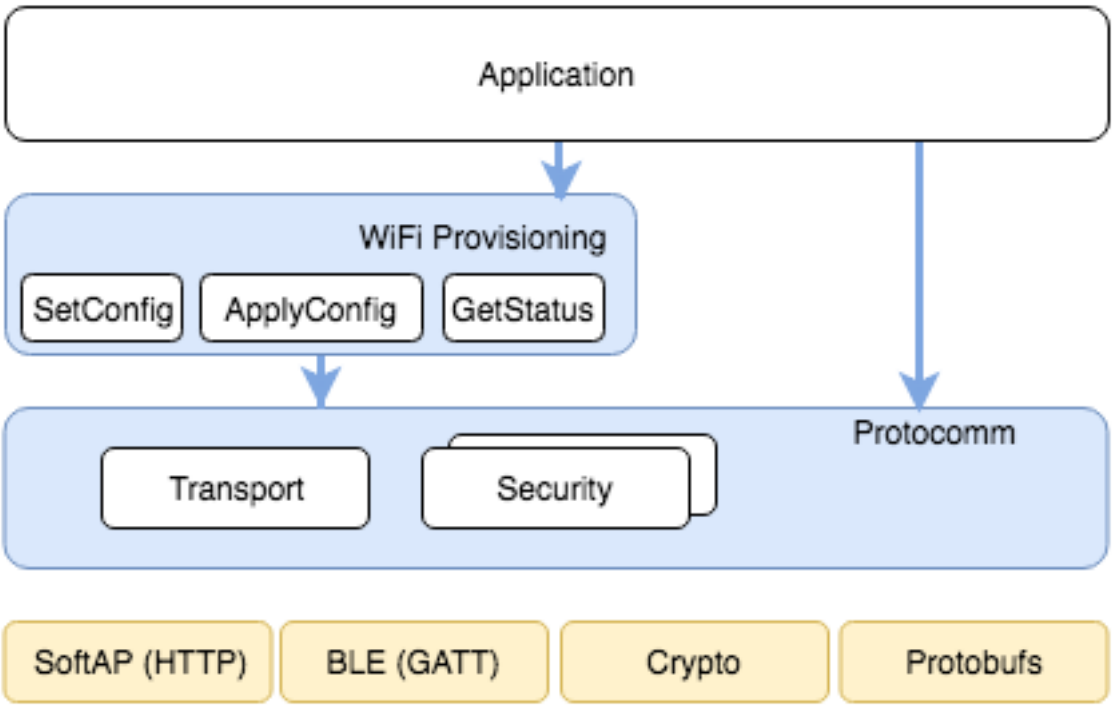


图 7: Unified Provisioning Architecture

It relies on the base layer called *Protocol Communication* (Protocol Communication) which provides a framework for security schemes and transport mechanisms. Wi-Fi Provisioning layer uses Protocomm to provide simple call-backs to the application for setting the configuration and getting the Wi-Fi status. The application has control over implementation of these callbacks. In addition application can directly use protocomm to register custom handlers.

Application creates a protocomm instance which is mapped to a specific transport and specific security scheme. Each transport in the protocomm has a concept of an "end-point" which corresponds to logical channel for communication for specific type of information. For example security handshake happens on a different endpoint than the Wi-Fi configuration endpoint. Each end-point is identified using a string and depending on the transport internal representation of the end-point changes. In case of SoftAP+HTTP transport the end-point corresponds to URI whereas in case of BLE the end-point corresponds to GATT characteristic with specific UUID. Developers can create custom end-points and implement handler for the data that is received or sent over the same end-point.

Security Schemes

At present, unified provisioning supports the following security schemes:

1. Security0 - No security (No encryption)
2. **Security1 - Curve25519-based key exchange, shared key derivation and AES256-CTR mode encryption of the data. It s**
    a. Authorized - Proof of Possession (PoP) string used to authorize session and derive shared key
    b. No Auth (Null PoP) - Shared key derived through key exchange only
3. Security2 - SRP6a-based shared key derivation and AES256-GCM mode encryption of the data.

备注: The respective security schemes need to be enabled through the project configuration menu. Please refer to the Enabling protocom security version section in *Protocol Communication* (Protocol Communication) for more details.

Security1 Scheme

Security1 scheme details are shown in the below sequence diagram -

Security2 Scheme

Security2 scheme is based on the Secure Remote Password (SRP6a) protocol - RFC 5054. The protocol requires the Salt and Verifier to be generated beforehand with help of the identifying username `I` and the plaintext password `p`. The Salt and Verifier are then stored on ESP32-C2. - The password `p` and username `I` are to be provided to the Phone App (Provisioning entity) by suitable means for example QR code sticker.

Security2 scheme details are shown in the below sequence diagram -

Sample Code

Please refer to *Protocol Communication* and *Wi-Fi Provisioning* for API guides and code snippets on example usage.

Application implementation can be found as an example under provisioning.

Provisioning Tools

Provisioning applications are available for various platforms, along with source code:

- **Android:**
    - BLE Provisioning app on Play Store.
    - SoftAP Provisioning app on Play Store.
    - Source code on GitHub: esp-idf-provisioning-android.
- **iOS:**
    - BLE Provisioning app on app store.
    - SoftAP Provisioning app on app Store.
    - Source code on GitHub: esp-idf-provisioning-ios.
- Linux/MacOS/Windows : tools/esp_prov (a python based command line tool for provisioning)

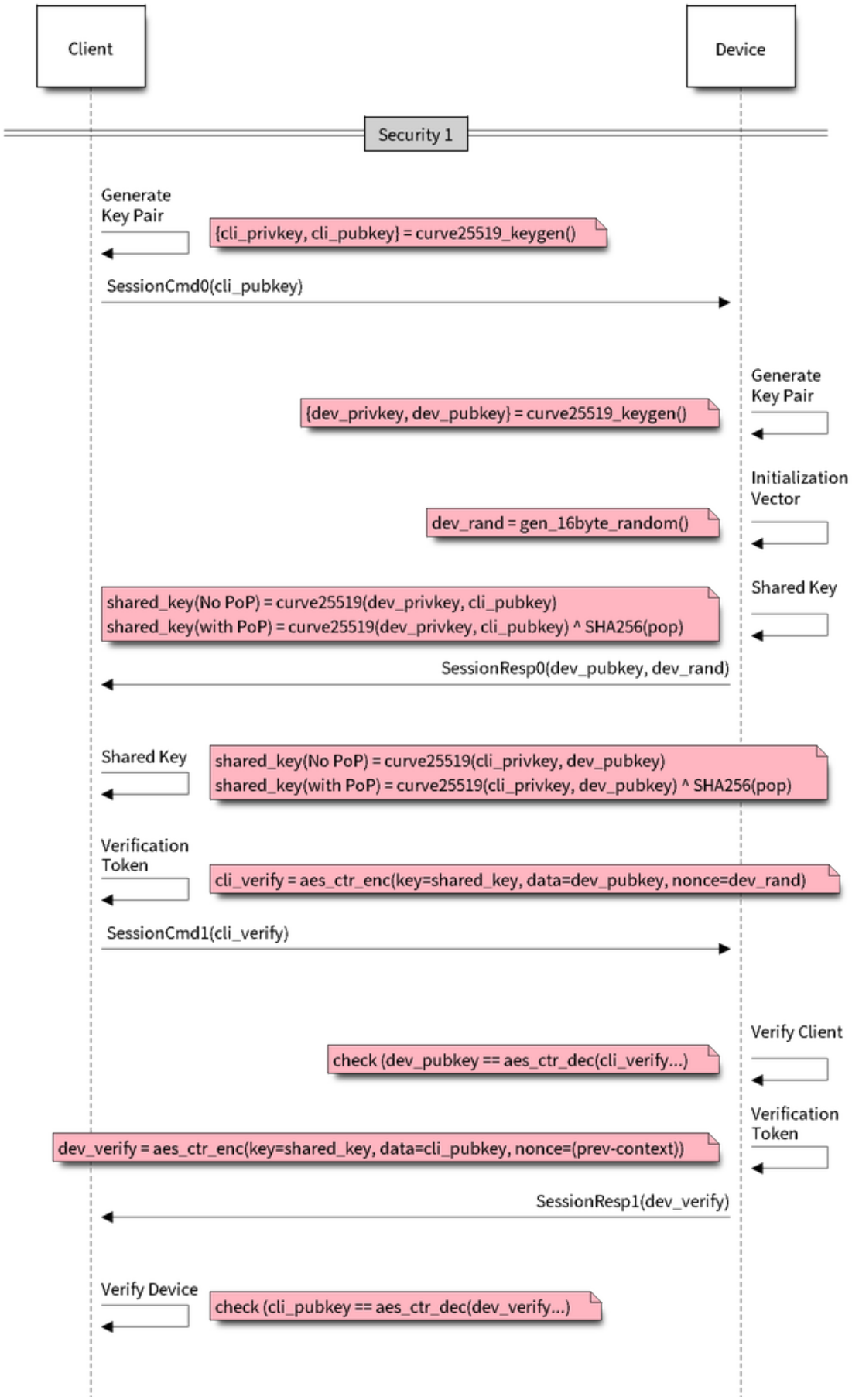


图 8: Security1

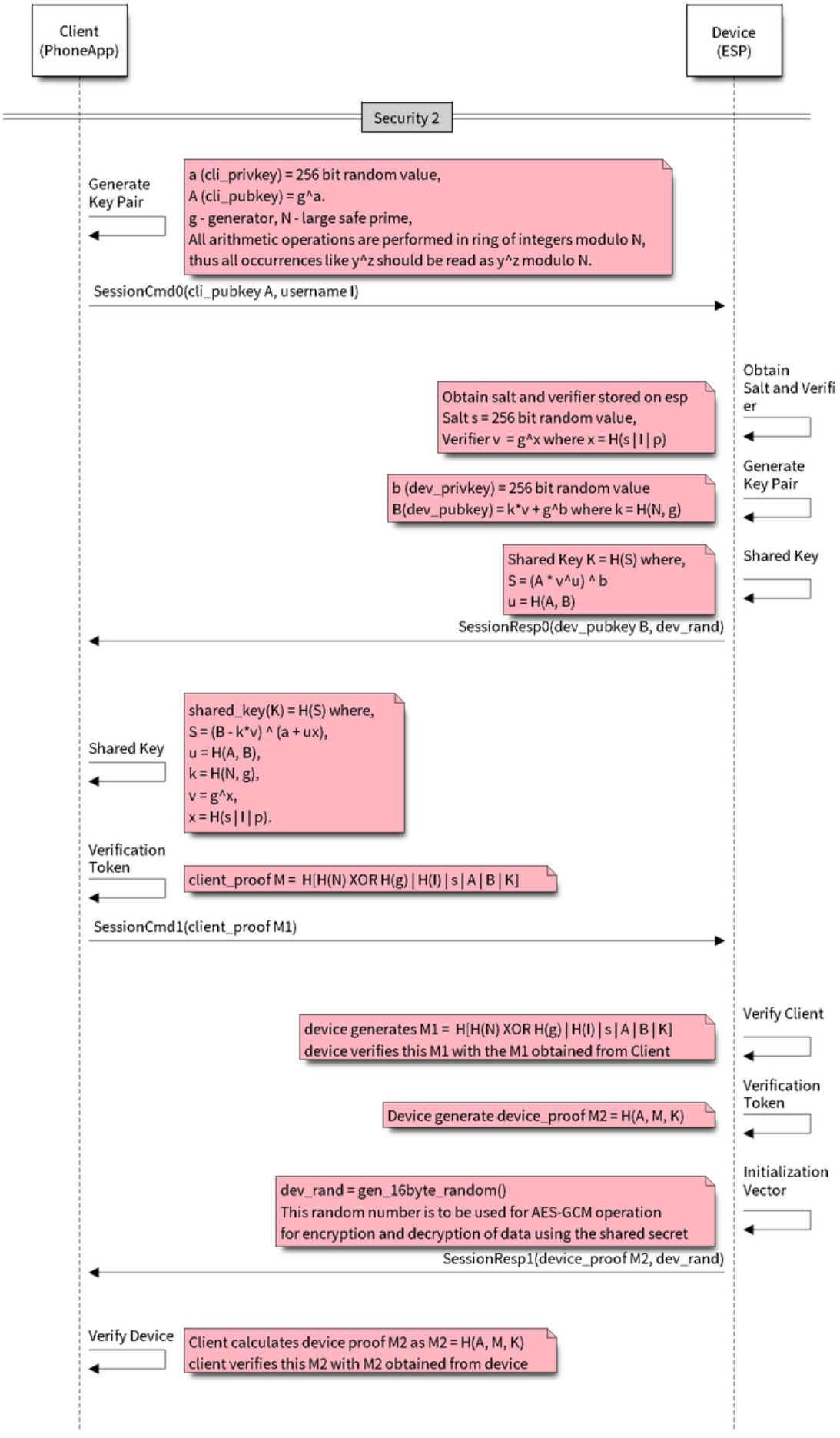


图 9: Security2

The phone applications offer simple UI and thus more user centric, while the command line application is useful as a debugging tool for developers.

2.8.3 Wi-Fi Provisioning

Overview

This component provides APIs that control Wi-Fi provisioning service for receiving and configuring Wi-Fi credentials over SoftAP or BLE transport via secure *Protocol Communication (protocomm)* sessions. The set of `wifi_prov_mgr_` APIs help in quickly implementing a provisioning service having necessary features with minimal amount of code and sufficient flexibility.

Initialization *`wifi_prov_mgr_init()`* is called to configure and initialize the provisioning manager and thus this must be called prior to invoking any other `wifi_prov_mgr_` APIs. Note that the manager relies on other components of IDF, namely NVS, TCP/IP, Event Loop and Wi-Fi (and optionally mDNS), hence these must be initialized beforehand. The manager can be de-initialized at any moment by making a call to *`wifi_prov_mgr_deinit()`*.

```
wifi_prov_mgr_config_t config = {
    .scheme = wifi_prov_scheme_ble,
    .scheme_event_handler = WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM
};

ESP_ERROR_CHECK( wifi_prov_mgr_init(config) );
```

The configuration structure `wifi_prov_mgr_config_t` has a few fields to specify the behavior desired of the manager :

- *scheme* : This is used to specify the provisioning scheme. Each scheme corresponds to one of the modes of transport supported by protocomm. Hence, we have three options :
  - `wifi_prov_scheme_ble` : BLE transport and GATT Server for handling provisioning commands
  - `wifi_prov_scheme_softap` : Wi-Fi SoftAP transport and HTTP Server for handling provisioning commands
  - `wifi_prov_scheme_console` : Serial transport and console for handling provisioning commands
- *scheme_event_handler* : An event handler defined along with scheme. Choosing appropriate scheme specific event handler allows the manager to take care of certain matters automatically. Presently this is not used for either SoftAP or Console based provisioning, but is very convenient for BLE. To understand how, we must recall that Bluetooth requires quite some amount of memory to function and once provisioning is finished, the main application may want to reclaim back this memory (or part of it, if it needs to use either BLE or classic BT). Also, upon every future reboot of a provisioned device, this reclamation of memory needs to be performed again. To reduce this complication in using `wifi_prov_scheme_ble`, the scheme specific handlers have been defined, and depending upon the chosen handler, the BLE / classic BT / BTDM memory will be freed automatically when the provisioning manager is de-initialized. The available options are:
  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM` - Free both classic BT and BLE (BTDM) memory. Used when main application doesn't require Bluetooth at all.
  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE` - Free only BLE memory. Used when main application requires classic BT.
  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT` - Free only classic BT. Used when main application requires BLE. In this case freeing happens right when the manager is initialized.
  - `WIFI_PROV_EVENT_HANDLER_NONE` - Don't use any scheme specific handler. Used when provisioning scheme is not BLE (i.e. SoftAP or Console), or when main application wants to handle the memory reclaiming on its own, or needs both BLE and classic BT to function.
- *app_event_handler* (Deprecated) : It is now recommended to catch `WIFI_PROV_EVENT``s that are emitted to the default event loop handler. See`

definition of ``wifi_prov_cb_event_t for the list of events that are generated by the provisioning service. Here is an excerpt showing some of the provisioning events:

```
static void event_handler(void* arg, esp_event_base_t event_base,
                          int event_id, void* event_data)
{
    if (event_base == WIFI_PROV_EVENT) {
        switch (event_id) {
            case WIFI_PROV_START:
                ESP_LOGI(TAG, "Provisioning started");
                break;
            case WIFI_PROV_CRED_RECV: {
                wifi_sta_config_t *wifi_sta_cfg = (wifi_sta_config_t␣
↪*)event_data;
                ESP_LOGI(TAG, "Received Wi-Fi credentials"
                         "\n\tSSID     : %s\n\tPassword : %s",
                         (const char *) wifi_sta_cfg->ssid,
                         (const char *) wifi_sta_cfg->password);
                break;
            }
            case WIFI_PROV_CRED_FAIL: {
                wifi_prov_sta_fail_reason_t *reason = (wifi_prov_sta_fail_
↪reason_t *)event_data;
                ESP_LOGE(TAG, "Provisioning failed!\n\tReason : %s"
                         "\n\tPlease reset to factory and retry␣
↪provisioning",
                         (*reason == WIFI_PROV_STA_AUTH_ERROR) ?
                         "Wi-Fi station authentication failed" : "Wi-Fi␣
↪access-point not found");
                break;
            }
            case WIFI_PROV_CRED_SUCCESS:
                ESP_LOGI(TAG, "Provisioning successful");
                break;
            case WIFI_PROV_END:
                /* De-initialize manager once provisioning is finished */
                wifi_prov_mgr_deinit();
                break;
            default:
                break;
        }
    }
}
```

The manager can be de-initialized at any moment by making a call to *wifi_prov_mgr_deinit()*.

Check Provisioning State Whether device is provisioned or not can be checked at runtime by calling *wifi_prov_mgr_is_provisioned()*. This internally checks if the Wi-Fi credentials are stored in NVS.

Note that presently manager does not have its own NVS namespace for storage of Wi-Fi credentials, instead it relies on the esp_wifi_ APIs to set and get the credentials stored in NVS from the default location.

If provisioning state needs to be reset, any of the following approaches may be taken :

- the associated part of NVS partition has to be erased manually
- main application must implement some logic to call esp_wifi_ APIs for erasing the credentials at runtime
- main application must implement some logic to force start the provisioning irrespective of the provisioning state

```
bool provisioned = false;
ESP_ERROR_CHECK( wifi_prov_mgr_is_provisioned(&provisioned) );
```

Start Provisioning Service At the time of starting provisioning we need to specify a service name and the corresponding key. These translate to :

- Wi-Fi SoftAP SSID and passphrase, respectively, when scheme is `wifi_prov_scheme_softap`
- BLE Device name (service key is ignored) when scheme is `wifi_prov_scheme_ble`

Also, since internally the manager uses *protocomm*, we have the option of choosing one of the security features provided by it :

- Security 1 is secure communication which consists of a prior handshake involving X25519 key exchange along with authentication using a proof of possession (*pop*), followed by AES-CTR for encryption/decryption of subsequent messages
- Security 0 is simply plain text communication. In this case the *pop* is simply ignored

See *Provisioning* for details about the security features.

```
const char *service_name = "my_device";
const char *service_key  = "password";

wifi_prov_security_t security = WIFI_PROV_SECURITY_1;
const char *pop = "abcd1234";

ESP_ERROR_CHECK( wifi_prov_mgr_start_provisioning(security, pop, service_
↪name, service_key) );
```

The provisioning service will automatically finish only if it receives valid Wi-Fi AP credentials followed by successfully connection of device to the AP (IP obtained). Regardless of that, the provisioning service can be stopped at any moment by making a call to *`wifi_prov_mgr_stop_provisioning()`*.

备注: If the device fails to connect with the provided credentials, it won't accept new credentials anymore, but the provisioning service will keep on running (only to convey failure to the client), until the device is restarted. Upon restart the provisioning state will turn out to be true this time (as credentials will be found in NVS), but device will again fail to connect with those same credentials (unless an AP with the matching credentials somehow does become available). This situation can be fixed by resetting the credentials in NVS or force starting the provisioning service. This has been explained above in *Check Provisioning State*.

Waiting For Completion Typically, the main application will wait for the provisioning to finish, then de-initialize the manager to free up resources and finally start executing its own logic.

There are two ways for making this possible. The simpler way is to use a blocking call to *`wifi_prov_mgr_wait()`*.

```
// Start provisioning service
ESP_ERROR_CHECK( wifi_prov_mgr_start_provisioning(security, pop, service_
↪name, service_key) );

// Wait for service to complete
wifi_prov_mgr_wait();

// Finally de-initialize the manager
wifi_prov_mgr_deinit();
```

The other way is to use the default event loop handler to catch `WIFI_PROV_EVENT``s and call :cpp:func:`wifi_prov_mgr_deinit()` when event ID is ``WIFI_PROV_END:

```
static void event_handler(void* arg, esp_event_base_t event_base,
                          int event_id, void* event_data)
{
    if (event_base == WIFI_PROV_EVENT && event_id == WIFI_PROV_END) {
        /* De-initialize manager once provisioning is finished */
```

(下页继续)

(续上页)

```
        wifi_prov_mgr_deinit();
    }
}
```

User Side Implementation When the service is started, the device to be provisioned is identified by the advertised service name which, depending upon the selected transport, is either the BLE device name or the SoftAP SSID.

When using SoftAP transport, for allowing service discovery, mDNS must be initialized before starting provisioning. In this case the hostname set by the main application is used, and the service type is internally set to *_esp_wifi_prov*.

When using BLE transport, a custom 128 bit UUID should be set using `wifi_prov_scheme_ble_set_service_uuid()`. This UUID will be included in the BLE advertisement and will correspond to the primary GATT service that provides provisioning endpoints as GATT characteristics. Each GATT characteristic will be formed using the primary service UUID as base, with different auto assigned 12th and 13th bytes (assume counting starts from 0th byte). Since, an endpoint characteristic UUID is auto assigned, it shouldn't be used to identify the endpoint. Instead, client side applications should identify the endpoints by reading the User Characteristic Description (0x2901) descriptor for each characteristic, which contains the endpoint name of the characteristic. For example, if the service UUID is set to *55cc035e-fb27-4f80-be02-3c60828b7451*, each endpoint characteristic will be assigned a UUID like *55cc____-fb27-4f80-be02-3c60828b7451*, with unique values at the 12th and 13th bytes.

Once connected to the device, the provisioning related protocomm endpoints can be identified as follows :

表 5: Endpoints provided by Provisioning Service

Endpoint Name (BLE + GATT Server)	URI (SoftAP + HTTP Server + mDNS)	Description
prov-session	http://<mdns-hostname>.local/prov-session	Security endpoint used for session establishment
prov-scan	http://wifi-prov.local/prov-scan	Endpoint used for starting Wi-Fi scan and receiving scan results
prov-ctrl	http://wifi-prov.local/prov-ctrl	Endpoint used for controlling Wi-Fi provisioning state
prov-config	http://<mdns-hostname>.local/prov-config	Endpoint used for configuring Wi-Fi credentials on device
proto-ver	http://<mdns-hostname>.local/proto-ver	Endpoint for retrieving version info

Immediately after connecting, the client application may fetch the version / capabilities information from the *proto-ver* endpoint. All communications to this endpoint are un-encrypted, hence necessary information (that may be relevant for deciding compatibility) can be retrieved before establishing a secure session. The response is in JSON format and looks like : `prov: { ver: v1.1, cap: [no_pop] }, my_app: { ver: 1.345, cap: [cloud, local_ctrl] },....` Here label *prov* provides provisioning service version (*ver*) and capabilities (*cap*). For now, only *no_pop* capability is supported, which indicates that the service doesn't require proof of possession for authentication. Any application related version / capabilities will be given by other labels (like *my_app* in this example). These additional fields are set using `wifi_prov_mgr_set_app_info()`.

User side applications need to implement the signature handshaking required for establishing and authenticating secure protocomm sessions as per the security scheme configured for use (this is not needed when manager is configured to use protocomm security 0).

See Unified Provisioning for more details about the secure handshake and encryption used. Applications must use the *.proto* files found under protocomm/proto, which define the Protobuf message structures supported by *prov-session* endpoint.

Once a session is established, Wi-Fi credentials are configured using the following set of *wifi_config* commands, serialized as Protobuf messages (the corresponding *.proto* files can be found under wifi_provisioning/proto) :

- *get_status* - For querying the Wi-Fi connection status. The device will respond with a status which will be one of connecting / connected / disconnected. If status is disconnected, a disconnection reason will also be included in the status response.
- *set_config* - For setting the Wi-Fi connection credentials
- *apply_config* - For applying the credentials saved during *set_config* and start the Wi-Fi station

After session establishment, client can also request Wi-Fi scan results from the device. The results returned is a list of AP SSIDs, sorted in descending order of signal strength. This allows client applications to display APs nearby to the device at the time of provisioning, and users can select one of the SSIDs and provide the password which is then sent using the *wifi_config* commands described above. The *wifi_scan* endpoint supports the following protobuf commands :

- *scan_start* - For starting Wi-Fi scan with various options :
  - *blocking* (input) - If true, the command returns only when the scanning is finished
  - *passive* (input) - If true scan is started in passive mode (this may be slower) instead of active mode
  - *group_channels* (input) - This specifies whether to scan all channels in one go (when zero) or perform scanning of channels in groups, with 120ms delay between scanning of consecutive groups, and the value of this parameter sets the number of channels in each group. This is useful when transport mode is SoftAP, where scanning all channels in one go may not give the Wi-Fi driver enough time to send out beacons, and hence may cause disconnection with any connected stations. When scanning in groups, the manager will wait for atleast 120ms after completing scan on a group of channels, and thus allow the driver to send out the beacons. For example, given that the total number of Wi-Fi channels is 14, then setting group_channels to 4, will create 5 groups, with each group having 3 channels, except the last one which will have 14 % 3 = 2 channels. So, when scan is started, the first 3 channels will be scanned, followed by a 120ms delay, and then the next 3 channels, and so on, until all the 14 channels have been scanned. One may need to adjust this parameter as having only few channels in a group may slow down the overall scan time, while having too many may again cause disconnection. Usually a value of 4 should work for most cases. Note that for any other mode of transport, e.g. BLE, this can be safely set to 0, and hence achieve the fastest overall scanning time.
  - *period_ms* (input) - Scan parameter specifying how long to wait on each channel
- *scan_status* - Gives the status of scanning process :
  - *scan_finished* (output) - When scan has finished this returns true
  - *result_count* (output) - This gives the total number of results obtained till now. If scan is yet happening this number will keep on updating
- *scan_result* - For fetching scan results. This can be called even if scan is still on going
  - *start_index* (input) - Starting index from where to fetch the entries from the results list
  - *count* (input) - Number of entries to fetch from the starting index
  - *entries* (output) - List of entries returned. Each entry consists of *ssid*, *channel* and *rssi* information

The client can also control the provisioning state of the device using *wifi_ctrl* endpoint. The *wifi_ctrl* endpoint supports the following protobuf commands:

- *ctrl_reset* - Resets internal state machine of the device and clears provisioned credentials only in case of provisioning failures.
- *ctrl_reprov* - Resets internal state machine of the device and clears provisioned credentials only in case the device is to be provisioned again for new credentials after a previous successful provisioning

Additional Endpoints In case users want to have some additional protocomm endpoints customized to their requirements, this is done in two steps. First is creation of an endpoint with a specific name, and the second step is the registration of a handler for this endpoint. See *protocomm* for the function signature of an endpoint handler. A custom endpoint must be created after initialization and before starting the provisioning service. Whereas, the protocomm handler is registered for this endpoint only after starting the provisioning service.

```
wifi_prov_mgr_init(config);
wifi_prov_mgr_endpoint_create("custom-endpoint");
wifi_prov_mgr_start_provisioning(security, pop, service_name, service_
↪key);
```

(下页继续)

(续上页)

```
wifi_prov_mgr_endpoint_register("custom-endpoint", custom_ep_handler,␣
↪custom_ep_data);
```

When the provisioning service stops, the endpoint is unregistered automatically.

One can also choose to call *wifi_prov_mgr_endpoint_unregister()* to manually deactivate an endpoint at runtime. This can also be used to deactivate the internal endpoints used by the provisioning service.

When / How To Stop Provisioning Service? The default behavior is that once the device successfully connects using the Wi-Fi credentials set by the *apply_config* command, the provisioning service will be stopped (and BLE / SoftAP turned off) automatically after responding to the next *get_status* command. If *get_status* command is not received by the device, the service will be stopped after a 30s timeout.

On the other hand, if device was not able to connect using the provided Wi-Fi credentials, due to incorrect SSID / passphrase, the service will keep running, and *get_status* will keep responding with disconnected status and reason for disconnection. Any further attempts to provide another set of Wi-Fi credentials, will be rejected. These credentials will be preserved, unless the provisioning service is force started, or NVS erased.

If this default behavior is not desired, it can be disabled by calling *wifi_prov_mgr_disable_auto_stop()*. Now the provisioning service will only be stopped after an explicit call to *wifi_prov_mgr_stop_provisioning()*, which returns immediately after scheduling a task for stopping the service. The service stops after a certain delay and WIFI_PROV_END event gets emitted. This delay is specified by the argument to *wifi_prov_mgr_disable_auto_stop()*.

The customized behavior is useful for applications which want the provisioning service to be stopped some time after the Wi-Fi connection is successfully established. For example, if the application requires the device to connect to some cloud service and obtain another set of credentials, and exchange this credentials over a custom protocomm endpoint, then after successfully doing so stop the provisioning service by calling *wifi_prov_mgr_stop_provisioning()* inside the protocomm handler itself. The right amount of delay ensures that the transport resources are freed only after the response from the protocomm handler reaches the client side application.

Application Examples

For complete example implementation see provisioning/wifi_prov_mgr

Provisioning Tools

Provisioning applications are available for various platforms, along with source code:

- **Android:**
  - BLE Provisioning app on Play Store.
  - SoftAP Provisioning app on Play Store.
  - Source code on GitHub: esp-idf-provisioning-android.
- **iOS:**
  - BLE Provisioning app on app store.
  - SoftAP Provisioning app on app Store.
  - Source code on GitHub: esp-idf-provisioning-ios.
- Linux/MacOS/Windows : tools/esp_prov (a python based command line tool for provisioning)

The phone applications offer simple UI and thus more user centric, while the command line application is useful as a debugging tool for developers.

API Reference

Header File

- components/wifi_provisioning/include/wifi_provisioning/manager.h

Functions

esp_err_t **wifi_prov_mgr_init** (*wifi_prov_mgr_config_t* config)

Initialize provisioning manager instance.

Configures the manager and allocates internal resources

Configuration specifies the provisioning scheme (transport) and event handlers

Event WIFI_PROV_INIT is emitted right after initialization is complete

参数 **config** –**[in]** Configuration structure

返回

- ESP_OK : Success
- ESP_FAIL : Fail

void **wifi_prov_mgr_deinit** (void)

Stop provisioning (if running) and release resource used by the manager.

Event WIFI_PROV_DEINIT is emitted right after de-initialization is finished

If provisioning service is still active when this API is called, it first stops the service, hence emitting WIFI_PROV_END, and then performs the de-initialization

esp_err_t **wifi_prov_mgr_is_provisioned** (bool *provisioned)

Checks if device is provisioned.

This checks if Wi-Fi credentials are present on the NVS

The Wi-Fi credentials are assumed to be kept in the same NVS namespace as used by esp_wifi component

If one were to call esp_wifi_set_config() directly instead of going through the provisioning process, this function will still yield true (i.e. device will be found to be provisioned)

备注: Calling wifi_prov_mgr_start_provisioning() automatically resets the provision state, irrespective of what the state was prior to making the call.

参数 **provisioned** –**[out]** True if provisioned, else false

返回

- ESP_OK : Retrieved provision state successfully
- ESP_FAIL : Wi-Fi not initialized
- ESP_ERR_INVALID_ARG : Null argument supplied

esp_err_t **wifi_prov_mgr_start_provisioning** (*wifi_prov_security_t* security, const void *wifi_prov_sec_params, const char *service_name, const char *service_key)

Start provisioning service.

This starts the provisioning service according to the scheme configured at the time of initialization. For scheme :

- wifi_prov_scheme_ble : This starts protocomm_ble, which internally initializes BLE transport and starts GATT server for handling provisioning requests
- wifi_prov_scheme_softap : This activates SoftAP mode of Wi-Fi and starts protocomm_httpd, which internally starts an HTTP server for handling provisioning requests (If mDNS is active it also starts advertising service with type _esp_wifi_prov._tcp)

Event WIFI_PROV_START is emitted right after provisioning starts without failure

备注: This API will start provisioning service even if device is found to be already provisioned, i.e. wifi_prov_mgr_is_provisioned() yields true

参数

- **security** –**[in]** Specify which protocomm security scheme to use :
  - WIFI_PROV_SECURITY_0 : For no security
  - WIFI_PROV_SECURITY_1 : x25519 secure handshake for session establishment followed by AES-CTR encryption of provisioning messages
  - WIFI_PROV_SECURITY_2: SRP6a based authentication and key exchange followed by AES-GCM encryption/decryption of provisioning messages
- **wifi_prov_sec_params** –**[in]** Pointer to security params (NULL if not needed). This is not needed for protocomm security 0 This pointer should hold the struct of type wifi_prov_security1_params_t for protocomm security 1 and wifi_prov_security2_params_t for protocomm security 2 respectively. This pointer and its contents should be valid till the provisioning service is running and has not been stopped or de-inited.
- **service_name** –**[in]** Unique name of the service. This translates to:
  - Wi-Fi SSID when provisioning mode is softAP
  - Device name when provisioning mode is BLE
- **service_key** –**[in]** Key required by client to access the service (NULL if not needed). This translates to:
  - Wi-Fi password when provisioning mode is softAP
  - ignored when provisioning mode is BLE

返回

- ESP_OK : Provisioning started successfully
- ESP_FAIL : Failed to start provisioning service
- ESP_ERR_INVALID_STATE : Provisioning manager not initialized or already started

void **wifi_prov_mgr_stop_provisioning**(void)

Stop provisioning service.

If provisioning service is active, this API will initiate a process to stop the service and return. Once the service actually stops, the event WIFI_PROV_END will be emitted.

If wifi_prov_mgr_deinit() is called without calling this API first, it will automatically stop the provisioning service and emit the WIFI_PROV_END, followed by WIFI_PROV_DEINIT, before returning.

This API will generally be used along with wifi_prov_mgr_disable_auto_stop() in the scenario when the main application has registered its own endpoints, and wishes that the provisioning service is stopped only when some protocomm command from the client side application is received.

Calling this API inside an endpoint handler, with sufficient cleanup_delay, will allow the response / acknowledgment to be sent successfully before the underlying protocomm service is stopped.

Cleaup_delay is set when calling wifi_prov_mgr_disable_auto_stop(). If not specified, it defaults to 1000ms.

For straightforward cases, using this API is usually not necessary as provisioning is stopped automatically once WIFI_PROV_CRED_SUCCESS is emitted. Stopping is delayed (maximum 30 seconds) thus allowing the client side application to query for Wi-Fi state, i.e. after receiving the first query and sending `Wi-Fi state connected` response the service is stopped immediately.

void **wifi_prov_mgr_wait**(void)

Wait for provisioning service to finish.

Calling this API will block until provisioning service is stopped i.e. till event WIFI_PROV_END is emitted.

This will not block if provisioning is not started or not initialized.

esp_err_t **wifi_prov_mgr_disable_auto_stop**(uint32_t cleanup_delay)

Disable auto stopping of provisioning service upon completion.

By default, once provisioning is complete, the provisioning service is automatically stopped, and all endpoints (along with those registered by main application) are deactivated.

This API is useful in the case when main application wishes to close provisioning service only after it receives some protocomm command from the client side app. For example, after connecting to Wi-Fi, the device may want to connect to the cloud, and only once that is successfully, the device is said to be fully configured. But, then it is upto the main application to explicitly call wifi_prov_mgr_stop_provisioning() later when the device is fully configured and the provisioning service is no longer required.

备注: This must be called before executing wifi_prov_mgr_start_provisioning()

参数 **cleanup_delay** –**[in]** Sets the delay after which the actual cleanup of transport related resources is done after a call to wifi_prov_mgr_stop_provisioning() returns. Minimum allowed value is 100ms. If not specified, this will default to 1000ms.

返回

- ESP_OK : Success
- ESP_ERR_INVALID_STATE : Manager not initialized or provisioning service already started

esp_err_t **wifi_prov_mgr_set_app_info**(const char *label, const char *version, const char **capabilities, size_t total_capabilities)

Set application version and capabilities in the JSON data returned by proto-ver endpoint.

This function can be called multiple times, to specify information about the various application specific services running on the device, identified by unique labels.

The provisioning service itself registers an entry in the JSON data, by the label "prov", containing only provisioning service version and capabilities. Application services should use a label other than "prov" so as not to overwrite this.

备注: This must be called before executing wifi_prov_mgr_start_provisioning()

参数

- **label** –**[in]** String indicating the application name.
- **version** –**[in]** String indicating the application version. There is no constraint on format.
- **capabilities** –**[in]** Array of strings with capabilities. These could be used by the client side app to know the application registered endpoint capabilities
- **total_capabilities** –**[in]** Size of capabilities array

返回

- ESP_OK : Success
- ESP_ERR_INVALID_STATE : Manager not initialized or provisioning service already started
- ESP_ERR_NO_MEM : Failed to allocate memory for version string
- ESP_ERR_INVALID_ARG : Null argument

esp_err_t **wifi_prov_mgr_endpoint_create**(const char *ep_name)

Create an additional endpoint and allocate internal resources for it.

This API is to be called by the application if it wants to create an additional endpoint. All additional endpoints will be assigned UUIDs starting from 0xFF54 and so on in the order of execution.

protocomm handler for the created endpoint is to be registered later using wifi_prov_mgr_endpoint_register() after provisioning has started.

备注: This API can only be called BEFORE provisioning is started

备注： Additional endpoints can be used for configuring client provided parameters other than Wi-Fi credentials, that are necessary for the main application and hence must be set prior to starting the application

备注： After session establishment, the additional endpoints must be targeted first by the client side application before sending Wi-Fi configuration, because once Wi-Fi configuration finishes the provisioning service is stopped and hence all endpoints are unregistered

参数 **ep_name** –**[in]** unique name of the endpoint

返回

- ESP_OK : Success
- ESP_FAIL : Failure

esp_err_t **wifi_prov_mgr_endpoint_register**(const char *ep_name, *protocomm_req_handler_t* handler, void *user_ctx)

Register a handler for the previously created endpoint.

This API can be called by the application to register a protocomm handler to any endpoint that was created using wifi_prov_mgr_endpoint_create().

备注： This API can only be called AFTER provisioning has started

备注： Additional endpoints can be used for configuring client provided parameters other than Wi-Fi credentials, that are necessary for the main application and hence must be set prior to starting the application

备注： After session establishment, the additional endpoints must be targeted first by the client side application before sending Wi-Fi configuration, because once Wi-Fi configuration finishes the provisioning service is stopped and hence all endpoints are unregistered

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

void **wifi_prov_mgr_endpoint_unregister**(const char *ep_name)

Unregister the handler for an endpoint.

This API can be called if the application wants to selectively unregister the handler of an endpoint while the provisioning is still in progress.

All the endpoint handlers are unregistered automatically when the provisioning stops.

参数 **ep_name** –**[in]** Name of the endpoint

esp_err_t **wifi_prov_mgr_get_wifi_state**(*wifi_prov_sta_state_t* *state)

Get state of Wi-Fi Station during provisioning.

参数 **state** –**[out]** Pointer to wifi_prov_sta_state_t variable to be filled

返回

- ESP_OK : Successfully retrieved Wi-Fi state

- ESP_FAIL : Provisioning app not running

esp_err_t **wifi_prov_mgr_get_wifi_disconnect_reason**(*wifi_prov_sta_fail_reason_t* *reason)

Get reason code in case of Wi-Fi station disconnection during provisioning.

参数 **reason** –**[out]** Pointer to wifi_prov_sta_fail_reason_t variable to be filled

返回

- ESP_OK : Successfully retrieved Wi-Fi disconnect reason
- ESP_FAIL : Provisioning app not running

esp_err_t **wifi_prov_mgr_configure_sta**(*wifi_config_t* *wifi_cfg)

Runs Wi-Fi as Station with the supplied configuration.

Configures the Wi-Fi station mode to connect to the AP with SSID and password specified in config structure and sets Wi-Fi to run as station.

This is automatically called by provisioning service upon receiving new credentials.

If credentials are to be supplied to the manager via a different mode other than through protocomm, then this API needs to be called.

Event WIFI_PROV_CRED_RECV is emitted after credentials have been applied and Wi-Fi station started

参数 **wifi_cfg** –**[in]** Pointer to Wi-Fi configuration structure

返回

- ESP_OK : Wi-Fi configured and started successfully
- ESP_FAIL : Failed to set configuration

esp_err_t **wifi_prov_mgr_reset_provisioning**(void)

Reset Wi-Fi provisioning config.

Calling this API will restore WiFi stack persistent settings to default values.

返回

- ESP_OK : Reset provisioning config successfully
- ESP_FAIL : Failed to reset provisioning config

esp_err_t **wifi_prov_mgr_reset_sm_state_on_failure**(void)

Reset internal state machine and clear provisioned credentials.

This API should be used to restart provisioning ONLY in the case of provisioning failures without rebooting the device.

返回

- ESP_OK : Reset provisioning state machine successfully
- ESP_FAIL : Failed to reset provisioning state machine
- ESP_ERR_INVALID_STATE : Manager not initialized

esp_err_t **wifi_prov_mgr_reset_sm_state_for_reprovision**(void)

Reset internal state machine and clear provisioned credentials.

This API can be used to restart provisioning ONLY in case the device is to be provisioned again for new credentials after a previous successful provisioning without rebooting the device.

备 注: This API can be used only if provisioning auto-stop has been disabled using wifi_prov_mgr_disable_auto_stop()

返回

- ESP_OK : Reset provisioning state machine successfully
- ESP_FAIL : Failed to reset provisioning state machine
- ESP_ERR_INVALID_STATE : Manager not initialized

Structures

struct **wifi_prov_event_handler_t**

Event handler that is used by the manager while provisioning service is active.

Public Members

wifi_prov_cb_func_t **event_cb**

Callback function to be executed on provisioning events

void ***user_data**

User context data to pass as parameter to callback function

struct **wifi_prov_scheme**

Structure for specifying the provisioning scheme to be followed by the manager.

备注: Ready to use schemes are available:

- wifi_prov_scheme_ble : for provisioning over BLE transport + GATT server
- wifi_prov_scheme_softap : for provisioning over SoftAP transport + HTTP server
- wifi_prov_scheme_console : for provisioning over Serial UART transport + Console (for debugging)

Public Members

esp_err_t (***prov_start**)(*protocomm_t* *pc, void *config)

Function which is to be called by the manager when it is to start the provisioning service associated with a protocomm instance and a scheme specific configuration

esp_err_t (***prov_stop**)(*protocomm_t* *pc)

Function which is to be called by the manager to stop the provisioning service previously associated with a protocomm instance

void *(***new_config**)(void)

Function which is to be called by the manager to generate a new configuration for the provisioning service, that is to be passed to *prov_start()*

void (***delete_config**)(void *config)

Function which is to be called by the manager to delete a configuration generated using *new_config()*

esp_err_t (***set_config_service**)(void *config, const char *service_name, const char *service_key)

Function which is to be called by the manager to set the service name and key values in the configuration structure

esp_err_t (***set_config_endpoint**)(void *config, const char *endpoint_name, uint16_t uuid)

Function which is to be called by the manager to set a protocomm endpoint with an identifying name and UUID in the configuration structure

wifi_mode_t **wifi_mode**

Sets mode of operation of Wi-Fi during provisioning This is set to :

- WIFI_MODE_APSTA for SoftAP transport
- WIFI_MODE_STA for BLE transport

struct **wifi_prov_mgr_config_t**

Structure for specifying the manager configuration.

Public Members

wifi_prov_scheme_t **scheme**

Provisioning scheme to use. Following schemes are already available:

- wifi_prov_scheme_ble : for provisioning over BLE transport + GATT server
- wifi_prov_scheme_softap : for provisioning over SoftAP transport + HTTP server + mDNS (optional)
- wifi_prov_scheme_console : for provisioning over Serial UART transport + Console (for debugging)

wifi_prov_event_handler_t **scheme_event_handler**

Event handler required by the scheme for incorporating scheme specific behavior while provisioning manager is running. Various options may be provided by the scheme for setting this field. Use WIFI_PROV_EVENT_HANDLER_NONE when not used. When using scheme wifi_prov_scheme_ble, the following options are available:

- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM
- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE
- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT

wifi_prov_event_handler_t **app_event_handler**

Event handler that can be set for the purpose of incorporating application specific behavior. Use WIFI_PROV_EVENT_HANDLER_NONE when not used.

Macros

WIFI_PROV_EVENT_HANDLER_NONE

Event handler can be set to none if not used.

Type Definitions

typedef void (***wifi_prov_cb_func_t**)(void *user_data, *wifi_prov_cb_event_t* event, void *event_data)

typedef struct *wifi_prov_scheme* **wifi_prov_scheme_t**

Structure for specifying the provisioning scheme to be followed by the manager.

备注： Ready to use schemes are available:

- wifi_prov_scheme_ble : for provisioning over BLE transport + GATT server
- wifi_prov_scheme_softap : for provisioning over SoftAP transport + HTTP server
- wifi_prov_scheme_console : for provisioning over Serial UART transport + Console (for debugging)

---

typedef enum *wifi_prov_security* **wifi_prov_security_t**

Security modes supported by the Provisioning Manager.

These are same as the security modes provided by protocomm

typedef *protocomm_security2_params_t* **wifi_prov_security2_params_t**

Security 2 params structure This needs to be passed when using WIFI_PROV_SECURITY_2.

Enumerations

enum **wifi_prov_cb_event_t**

Events generated by manager.

These events are generated in order of declaration and, for the stretch of time between initialization and de-initialization of the manager, each event is signaled only once

Values:

enumerator **WIFI_PROV_INIT**

Emitted when the manager is initialized

enumerator **WIFI_PROV_START**

Indicates that provisioning has started

enumerator **WIFI_PROV_CRED_RECV**

Emitted when Wi-Fi AP credentials are received via `protocomm` endpoint `wifi_config`. The event data in this case is a pointer to the corresponding *`wifi_sta_config_t`* structure

enumerator **WIFI_PROV_CRED_FAIL**

Emitted when device fails to connect to the AP of which the credentials were received earlier on event `WIFI_PROV_CRED_RECV`. The event data in this case is a pointer to the disconnection reason code with type `wifi_prov_sta_fail_reason_t`

enumerator **WIFI_PROV_CRED_SUCCESS**

Emitted when device successfully connects to the AP of which the credentials were received earlier on event `WIFI_PROV_CRED_RECV`

enumerator **WIFI_PROV_END**

Signals that provisioning service has stopped

enumerator **WIFI_PROV_DEINIT**

Signals that manager has been de-initialized

enum **wifi_prov_security**

Security modes supported by the Provisioning Manager.

These are same as the security modes provided by protocomm

Values:

enumerator **WIFI_PROV_SECURITY_0**

No security (plain-text communication)

enumerator **WIFI_PROV_SECURITY_1**

This secure communication mode consists of X25519 key exchange

- proof of possession (pop) based authentication
- AES-CTR encryption

enumerator **WIFI_PROV_SECURITY_2**

This secure communication mode consists of SRP6a based authentication and key exchange

- AES-GCM encryption/decryption

Header File

- components/wifi_provisioning/include/wifi_provisioning/scheme_ble.h

Functions

void **wifi_prov_scheme_ble_event_cb_free_btdm**(void *user_data, *wifi_prov_cb_event_t* event, void *event_data)

void **wifi_prov_scheme_ble_event_cb_free_ble**(void *user_data, *wifi_prov_cb_event_t* event, void *event_data)

void **wifi_prov_scheme_ble_event_cb_free_bt**(void *user_data, *wifi_prov_cb_event_t* event, void *event_data)

esp_err_t **wifi_prov_scheme_ble_set_service_uuid**(uint8_t *uuid128)

Set the 128 bit GATT service UUID used for provisioning.

This API is used to override the default 128 bit provisioning service UUID, which is 0000ffff-0000-1000-8000-00805f9b34fb.

This must be called before starting provisioning, i.e. before making a call to wifi_prov_mgr_start_provisioning(), otherwise the default UUID will be used.

备注: The data being pointed to by the argument must be valid atleast till provisioning is started. Upon start, the manager will store an internal copy of this UUID, and this data can be freed or invalidated afterwords.

参数 **uuid128** –**[in]** A custom 128 bit UUID

返回

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : Null argument

esp_err_t **wifi_prov_scheme_ble_set_mfg_data**(uint8_t *mfg_data, ssize_t mfg_data_len)

Set manufacturer specific data in scan response.

This must be called before starting provisioning, i.e. before making a call to wifi_prov_mgr_start_provisioning().

备注: It is important to understand that length of custom manufacturer data should be within limits. The manufacturer data goes into scan response along with BLE device name. By default, BLE device name length is of 11 Bytes, however it can vary as per application use case. So, one has to honour the scan response data size limits i.e. (mfg_data_len + 2) < 31 - (device_name_length + 2). If the mfg_data length exceeds this limit, the length will be truncated.

参数

- **mfg_data** –**[in]** Custom manufacturer data
- **mfg_data_len** –**[in]** Manufacturer data length

返回

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : Null argument

Macros

WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM

WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE

WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT

Header File

- components/wifi_provisioning/include/wifi_provisioning/scheme_softap.h

Functions

void **wifi_prov_scheme_softap_set_httpd_handle**(void *handle)

Provide HTTPD Server handle externally.

Useful in cases wherein applications need the webserver for some different operations, and do not want the wifi provisioning component to start/stop a new instance.

备注: This API should be called before wifi_prov_mgr_start_provisioning()

参数 **handle** –**[in]** Handle to HTTPD server instance

Header File

- components/wifi_provisioning/include/wifi_provisioning/scheme_console.h

Header File

- components/wifi_provisioning/include/wifi_provisioning/wifi_config.h

Functions

esp_err_t **wifi_prov_config_data_handler**(uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen, void *priv_data)

Handler for receiving and responding to requests from master.

This is to be registered as the `wifi_config` endpoint handler (protocomm `protocomm_req_handler_t`) using `protocomm_add_endpoint()`

Structures

struct **wifi_prov_sta_conn_info_t**

WiFi STA connected status information.

Public Members

char **ip_addr**[IP4ADDR_STRLEN_MAX]

IP Address received by station

char **bssid**[6]

BSSID of the AP to which connection was estalished

char **ssid**[33]

SSID of the to which connection was estalished

uint8_t **channel**

Channel of the AP

uint8_t **auth_mode**

Authorization mode of the AP

struct **wifi_prov_config_get_data_t**

WiFi status data to be sent in response to `get_status` request from master.

Public Members

wifi_prov_sta_state_t **wifi_state**

WiFi state of the station

wifi_prov_sta_fail_reason_t **fail_reason**

Reason for disconnection (valid only when `wifi_state` is `WIFI_STATION_DISCONNECTED`)

wifi_prov_sta_conn_info_t **conn_info**

Connection information (valid only when `wifi_state` is `WIFI_STATION_CONNECTED`)

struct **wifi_prov_config_set_data_t**

WiFi config data received by slave during `set_config` request from master.

Public Members

char **ssid**[33]

SSID of the AP to which the slave is to be connected

char **password**[64]

Password of the AP

char **bssid**[6]

BSSID of the AP

uint8_t **channel**

Channel of the AP

struct **wifi_prov_config_handlers**

Internal handlers for receiving and responding to protocomm requests from master.

This is to be passed as priv_data for protocomm request handler (refer to `wifi_prov_config_data_handler()`) when calling `protocomm_add_endpoint()`.

Public Members

esp_err_t (***get_status_handler**)(*wifi_prov_config_get_data_t* *resp_data, *wifi_prov_ctx_t* **ctx)

Handler function called when connection status of the slave (in WiFi station mode) is requested

esp_err_t (***set_config_handler**)(const *wifi_prov_config_set_data_t* *req_data, *wifi_prov_ctx_t* **ctx)

Handler function called when WiFi connection configuration (eg. AP SSID, password, etc.) of the slave (in WiFi station mode) is to be set to user provided values

esp_err_t (***apply_config_handler**)(*wifi_prov_ctx_t* **ctx)

Handler function for applying the configuration that was set in `set_config_handler`. After applying the station may get connected to the AP or may fail to connect. The slave must be ready to convey the updated connection status information when `get_status_handler` is invoked again by the master.

wifi_prov_ctx_t ***ctx**

Context pointer to be passed to above handler functions upon invocation

Type Definitions

typedef struct wifi_prov_ctx **wifi_prov_ctx_t**

Type of context data passed to each get/set/apply handler function set in *`wifi_prov_config_handlers`* structure.

This is passed as an opaque pointer, thereby allowing it be defined later in application code as per requirements.

typedef struct *wifi_prov_config_handlers* **wifi_prov_config_handlers_t**

Internal handlers for receiving and responding to protocomm requests from master.

This is to be passed as priv_data for protocomm request handler (refer to `wifi_prov_config_data_handler()`) when calling `protocomm_add_endpoint()`.

Enumerations

enum **wifi_prov_sta_state_t**

WiFi STA status for conveying back to the provisioning master.

Values:

enumerator **WIFI_PROV_STA_CONNECTING**

enumerator **WIFI_PROV_STA_CONNECTED**

enumerator **WIFI_PROV_STA_DISCONNECTED**

enum **wifi_prov_sta_fail_reason_t**

WiFi STA connection fail reason.

Values:

enumerator **WIFI_PROV_STA_AUTH_ERROR**

enumerator **WIFI_PROV_STA_AP_NOT_FOUND**

本部分的 API 示例代码存放在 ESP-IDF 示例项目的 provisioning 目录下。

本部分的 API 示例代码存放在 wifi/smart_config 目录下。

本部分的 API 示例代码存放在 wifi/wifi_easy_connect/dpp-enrollee 目录下。

2.9 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 Documents* 。

2.9.1 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，您可以直接使用，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. 调用 FatFs 函数 `f_mount`，随后调用 `f_fdisk` 或 `f_mkfs`，并使用与传递到*`esp_vfs_fat_register()`* 相同的驱动编号挂载文件系统。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)。若您需要使用长文件名 (LFN)，启用*CONFIG_FATFS_LONG_FILENAMES* 选项。请参考 here，查看更多信息；
5. 您可以选择启用*CONFIG_FATFS_USE_FASTSEEK* 选项，使用 POSIX lseek 来快速执行。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；
6. 您也可以选择直接调用 FatFs 库函数，但需要使用没有 VFS 前缀的路径（例如，`"/hello.txt"`）；
7. 关闭所有打开的文件；
8. 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；
9. 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；
10. 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_sdmmc_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。

esp_err_t **esp_vfs_fat_register**(const char *base_path, const char *fat_drive, size_t max_files, FATFS **out_fs)

Register FATFS with VFS component.

This function registers given FAT drive in VFS, at the specified base path. 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

- **base_path** –path prefix where FATFS should be registered
- **fat_drive** –FATFS drive specification; if only one drive is used, can be an empty string
- **max_files** –maximum number of files which can be open at the same time
- **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

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。

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_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_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

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.

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

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。

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

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，也可以初始化磨损均衡。

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。

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磨损均衡，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。

如果您决定使用 `fatfs_create_rawflash_image` （不支持磨损均衡），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）。

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。

您可以使用:

```
./fatfsparse.py [-h] [--wl-layer {detect,enabled,disabled}] fatfs_image.img
```

2.9.2 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）

2.9.3 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。为现有的键写入新的值可能产生如下结果：

- 如果新旧值数据类型相同，则更新值；
- 如果新旧值数据类型不同，则返回错误。

读取值时也会执行数据类型检查。如果读取操作的数据类型与该值的数据类型不匹配，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。

您可以使用以下函数，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-C2 flash 加密系统不直接兼容。但如果 NVS 加密与 ESP32-C2 flash 加密一起使用时，数据仍可以加密形式存储。详情请参阅*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

NVS 分区内存储的数据可使用 AES-XTS 进行加密，类似于 IEEE P1619 磁盘加密标准中提到的加密方式。为了实现加密，每个条目被均视为一个扇区，并将条目相对地址（相对于分区开头）传递给加密算法，用作扇区号。可通过*CONFIG_NVS_ENCRYPTION* 启用 NVS 加密。NVS 加密所需的密钥存储于其他分区，并且被*Flash 加密* 保护。因此，在使用 NVS 加密前应先启用*Flash 加密*。

启用*Flash 加密* 时，默认启用 NVS 加密。这是因为 Wi-Fi 驱动在默认的 NVS 分区中存储了凭证（如 SSID 和密码）。如已启用平台级加密，那么同时默认启用 NVS 加密有其必要性。

使用 NVS 加密，分区表必须包含*NVS 密钥分区*。在分区表选项 (menuconfig > Partition Table) 下，为 NVS 加密提供了两个包含*NVS 密钥分区* 的分区表，您可以通过工程配置菜单 (idf.py menuconfig) 进行选择。请参考 security/flash_encryption 中的例子，了解如何配置和使用 NVS 加密功能。

NVS 密钥分区 应用程序如果想使用 NVS 加密，则需要编译进一个类型为 *data*，子类型为 *key* 的密钥分区。该分区应标记为 已加密且最小为 4096 字节。如需了解更多详细信息，请参考分区表。在分区表选项 (menuconfig > Partition Table) 下提供了两个包含*NVS 密钥分区* 的额外分区表，可以直接用于*NVS 加密*。这些分区的具体结构见下表：

```
+-----------+-------------+-------------+----+
|              XTS encryption key (32)        |
+---------------------------------------------+
|              XTS tweak key (32)             |
+---------------------------------------------+
|                  CRC32 (4)                  |
+---------------------------------------------+
```

可以通过以下两种方式生成*NVS 密钥分区* 中的 XTS 加密密钥：

1. 在 ESP 芯片上生成密钥：
   启用 NVS 加密时，可用*nvs_flash_init()* API 函数来初始化加密的默认 NVS 分区，在内部生成 ESP 芯片上的 XTS 加密密钥。在找到*NVS 密钥分区* 后，API 函数利用 nvs_flash/include/nvs_flash.h 提供的*nvs_flash_generate_keys()* 函数，自动生成并存储该分区中的 NVS 密钥。只有当各自的密钥分区为空时，才会生成并存储新的密钥。可以借助*nvs_flash_secure_init_partition()* 用同一个密钥分区来读取安全配置，以初始化一个定制的加密 NVS 分区。
   API 函数*nvs_flash_secure_init()* 和*nvs_flash_secure_init_partition()* 不在内部产生密钥。当这些 API 函数用于初始化加密的 NVS 分区时，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，以加密的形式把密钥写到密钥分区上。

   备注： 请注意，使用该方法启动应用前，必须先完全擦除 *nvs_keys* 分区，否则该应用可能会认为 *nvs_keys* 分区不为空，并且包含数据格式错误，从而导致*ESP_ERR_NVS_CORRUPT_KEY_PART* 报错。如果遇到这种情况，可以使用以下命令：

   ```
   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
   ```

2. 使用预先生成的密钥分区：
   若*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 分区生成工具* 生成包含 XTS 加密密钥的*NVS 密钥分区*。用户可以借助以下两个命令，将预先生成的密钥分区储存在 flash 上：
   i) 建立并烧录分区表

   ```
   idf.py partition-table partition-table-flash
   ```

   ii) 调用 parttool.py，将密钥存储在 flash 上的*NVS 密钥分区* 中。详见:doc:‘ 分区表 </api-guides/partition-tables>‘ 的分区工具部分。

   ```
   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： 如需在设备处于 flash 加密开发模式时更新 NVS 密钥分区，请调用 parttool.py 对 NVS 密钥分区进行加密。同时，由于设备上的分区表也已加密，您还需要在构建目录（build/partition_table）中提供一个指向未加密分区表的指针。您可以使用如下命令：

   ```
   parttool.py --esptool-write-args encrypt --port PORT --partition-table-
   ↪file=PARTITION_TABLE_FILE --partition-table-offset PARTITION_TABLE_
   ↪OFFSET write_partition --partition-name="name of nvs_key partition" -
   ↪-input NVS_KEY_PARTITION_FILE
   ```

由于分区已标记为 *已加密*，而且启用了*Flash 加密*，引导程序在首次启动时将使用 flash 加密对密钥分区进行加密。

应用程序可以使用不同的密钥对不同的 NVS 分区进行加密，这样就会需要多个加密密钥分区。应用程序应为加解密操作提供正确的密钥或密钥分区。

加密读取/写入 `nvs_get_*` 和 `nvs_set_*` 等 NVS API 函数同样可以对 NVS 加密分区执行读写操作。

加密默认的 NVS 分区： 无需额外步骤即可启用默认 NVS 分区的加密。启用*CONFIG_NVS_ENCRYPTION* 时，*nvs_flash_init()* API 函数会在内部使用找到的第一个*NVS 密钥分区* 执行额外步骤，以启用默认 NVS 分区的加密（详情请参考 API 文档）。另外，*nvs_flash_secure_init()* API 函数也可以用来启用默认 NVS 分区的加密。

加密一个自定义的 NVS 分区： 使用*nvs_flash_secure_init_partition()* API 函数启用自定义 NVS 分区的加密，而非*nvs_flash_init_partition()*。

使用*nvs_flash_secure_init()* 和*nvs_flash_secure_init_partition()* API 函数时，应用程序如需在加密状态下执行 NVS 读写操作，应遵循以下步骤：

1. 使用 `esp_partition_find*` API 查找密钥分区和 NVS 数据分区；
2. 使用 `nvs_flash_read_security_cfg` 或 `nvs_flash_generate_keys` API 填充 `nvs_sec_cfg_t` 结构；
3. 使用 `nvs_flash_secure_init` 或 `nvs_flash_secure_init_partition` API 初始化 NVS flash 分区；
4. 使用 `nvs_open` 或 `nvs_open_from_partition` API 打开命名空间；
5. 使用 `nvs_get_*` 或 `nvs_set_*` API 执行 NVS 读取/写入操作；
6. 使用 `nvs_flash_deinit` API 释放已初始化的 NVS 分区。

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*。

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-C2 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，帮助您追踪程序流程，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-C2 模组重启也不会消失。

- value - 记录 ESP32-C2 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。

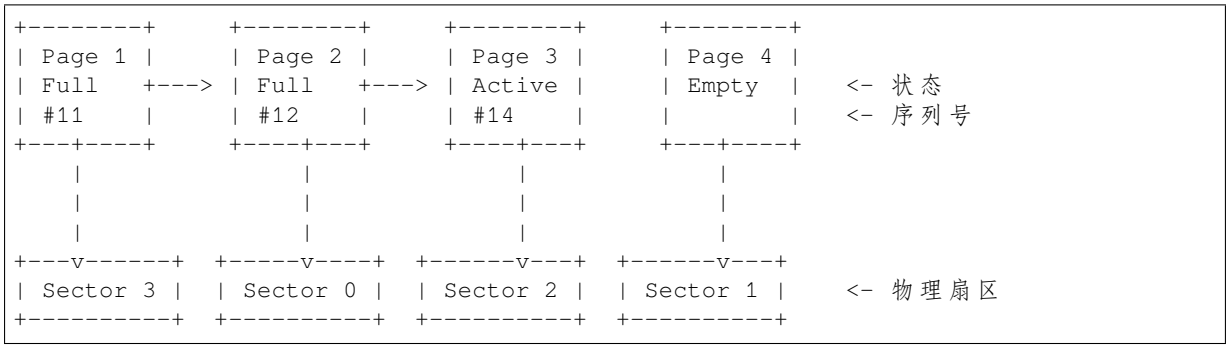
活跃状态 此时 flash 已完成初始化，页头部写入 flash，页面已具备有效序列号。页面中存在一些空条目，可写入数据。任意时刻，至多有一个页面处于活跃状态。

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。

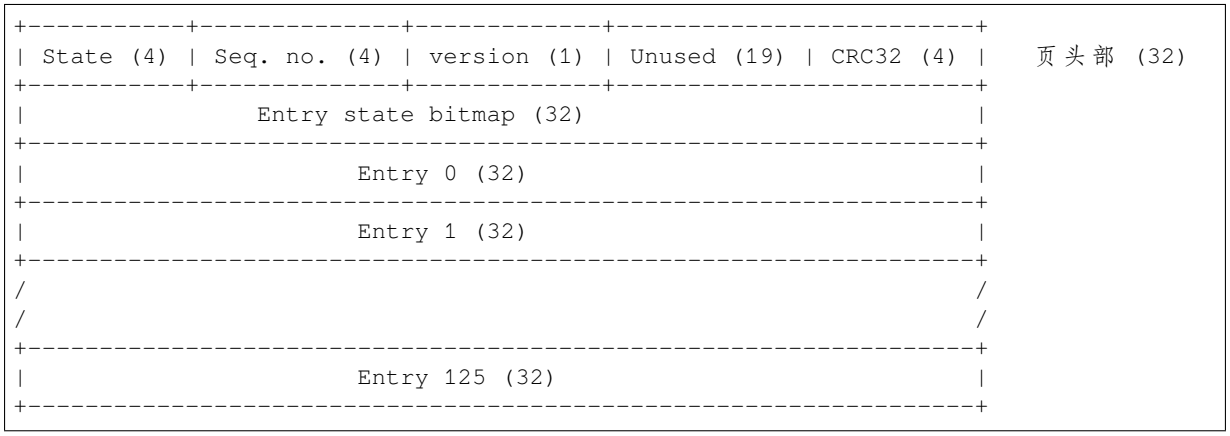
Flash 扇区映射至逻辑页面并没有特定的顺序，NVS 库会检查存储在 flash 扇区的页面序列号，并根据序列号组织页面。



页面结构 当前，我们假设 flash 扇区大小为 4096 字节，并且 ESP32-C2 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-C2 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）。



头部和条目状态位图写入 flash 时不加密。如果启用了 ESP32-C2 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）。

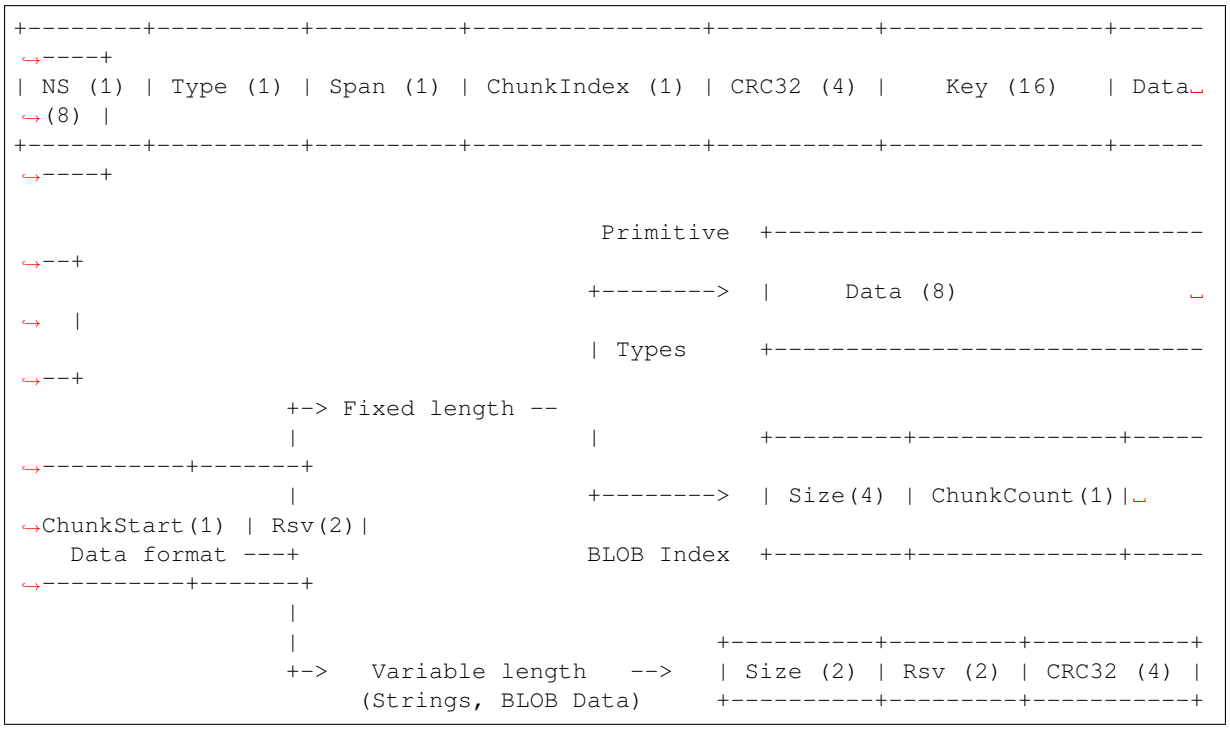
头部中 CRC32 值是由不包含状态值的条目计算所得（4 到 28 字节）。当前未使用的条目用 `0xff` 字节填充。

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"
+----------------------------------------+
| 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

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 partiton obtained in the previous step is empty, generate and store new keys in that NVS key partiton.
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_INVALID_ARG, if partition or cfg; -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_INVALID_ARG, if partition or cfg; -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.

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

Macros

NVS_KEY_SIZE

Header File

- components/nvs_flash/include/nvs.h

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.

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

Set value for the key, given its name. Note that the actual 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)
- 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)
- 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

This family of functions set value for the key, given its name. Note that actual storage will not be updated until nvs_commit function 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_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 used memory the partition.

This function calculates to runtime the number of used entries, free entries, total entries, and amount namespace in partition.

```
// Example of nvs_get_stats() to get the number of used entries and free␣
↪entries:
nvs_stats_t nvs_stats;
nvs_get_stats(NULL, &nvs_stats);
printf("Count: UsedEntries = (%d), FreeEntries = (%d), AllEntries = (%d)\n",
       nvs_stats.used_entries, nvs_stats.free_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 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 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);
```

(下页继续)

(续上页)

```
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_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 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 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 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**

Amount of used entries.

size_t **free_entries**

Amount of free entries.

size_t **total_entries**

Amount all available entries.

size_t **namespace_count**

Amount name space.

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)

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

2.9.4 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器 *(NVS)* 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 package

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。

支持加密

NVS 分区生成程序还可使用 AES-XTS 加密生成二进制加密文件。更多信息详见*NVS 加密*。

支持解密

如果 NVS 二进制文件采用了 AES-XTS 加密，该程序还可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。 • 版本 1：禁用多页 Blob； • 版本 2：启用多页 Blob； • 默认版本：版本 2。
–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
```

仅生成加密密钥分区　使用方法:

```
python nvs_partition_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）

运行以下命令仅生成加密密钥分区:

```
python nvs_partition_gen.py generate-key
```

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]
                                    input output size
```

位置参数:

参数	描述
input	待解析 CSV 文件的路径
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。 • 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
–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）

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`。

- 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/keys-<timestamp>.bin`。

备注： 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
```

您可以自定义格式版本号：- 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。

备注： 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。

2.9.5 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。

2.9.6 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（driver/sdmmc/include/driver/sdmmc_host.h 和 driver/spi/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 SD SPI 主机，请参阅*SD SPI 主机 API*

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。

API 参考

Header File

- components/sdmmc/include/sdmmc_cmd.h

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.

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
- **size** –number of bytes to read

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.

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
- **size** –number of bytes to write

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.

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
- **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.

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
- **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.

Header File

- components/driver/sdmmc/include/driver/sdmmc_types.h

Structures

struct **sdmmc_csd_t**

Decoded values from SD card Card Specific Data register

Public Members

int **csd_ver**

CSD structure format

int **mmc_ver**

MMC version (for CID format)

int **capacity**

total number of sectors

int **sector_size**

sector size in bytes

int **read_block_len**

block length for reads

int **card_command_class**

Card Command Class for SD

int **tr_speed**

Max transfer speed

struct **sdmmc_cid_t**

Decoded values from SD card Card IDentification register

Public Members

int **mfg_id**

manufacturer identification number

int **oem_id**

OEM/product identification number

char **name**[8]

product name (MMC v1 has the longest)

int **revision**

product revision

int **serial**

product serial number

int **date**

manufacturing date

struct **sdmmc_scr_t**

Decoded values from SD Configuration Register Note: When new member is added, update reserved bits accordingly

Public Members

uint32_t **sd_spec**

SD Physical layer specification version, reported by card

uint32_t **erase_mem_state**

data state on card after erase whether 0 or 1 (card vendor dependent)

uint32_t **bus_width**

bus widths supported by card: BIT(0) —1-bit bus, BIT(2) —4-bit bus

uint32_t **reserved**

reserved for future expansion

uint32_t **rsvd_mnf**

reserved for manufacturer usage

struct **sdmmc_ssr_t**

Decoded values from SD Status Register Note: When new member is added, update reserved bits accordingly

Public Members

uint32_t **alloc_unit_kb**

Allocation unit of the card, in multiples of kB (1024 bytes)

uint32_t **erase_size_au**

Erase size for the purpose of timeout calculation, in multiples of allocation unit

uint32_t **cur_bus_width**

SD current bus width

uint32_t **discard_support**

SD discard feature support

uint32_t **fule_support**

SD FULE (Full User Area Logical Erase) feature support

uint32_t **erase_timeout**

Timeout (in seconds) for erase of a single allocation unit

uint32_t **erase_offset**

Constant timeout offset (in seconds) for any erase operation

uint32_t **reserved**

reserved for future expansion

struct **sdmmc_ext_csd_t**

Decoded values of Extended Card Specific Data

Public Members

uint8_t **rev**

Extended CSD Revision

uint8_t **power_class**

Power class used by the card

uint8_t **erase_mem_state**

data state on card after erase whether 0 or 1 (card vendor dependent)

uint8_t **sec_feature**

secure data management features supported by the card

struct **sdmmc_switch_func_rsp_t**

SD SWITCH_FUNC response buffer

Public Members

uint32_t **data**[512 / 8 / sizeof(uint32_t)]

response data

struct **sdmmc_command_t**

SD/MMC command information

Public Members

uint32_t **opcode**

SD or MMC command index

uint32_t **arg**

SD/MMC command argument

sdmmc_response_t **response**

response buffer

void ***data**

buffer to send or read into

size_t **datalen**

length of data buffer

size_t **blklen**

block length

int **flags**

see below

esp_err_t **error**

error returned from transfer

uint32_t **timeout_ms**

response timeout, in milliseconds

struct **sdmmc_host_t**

SD/MMC Host description

This structure defines properties of SD/MMC host and functions of SD/MMC host which can be used by upper layers.

Public Members

uint32_t **flags**

flags defining host properties

int **slot**

slot number, to be passed to host functions

int **max_freq_khz**

max frequency supported by the host

float **io_voltage**

I/O voltage used by the controller (voltage switching is not supported)

esp_err_t (***init**)(void)

Host function to initialize the driver

esp_err_t (***set_bus_width**)(int slot, size_t width)

host function to set bus width

size_t (***get_bus_width**)(int slot)

host function to get bus width

esp_err_t (***set_bus_ddr_mode**)(int slot, bool ddr_enable)

host function to set DDR mode

esp_err_t (***set_card_clk**)(int slot, uint32_t freq_khz)

host function to set card clock frequency

esp_err_t (***set_cclk_always_on**)(int slot, bool cclk_always_on)

host function to set whether the clock is always enabled

esp_err_t (***do_transaction**)(int slot, *sdmmc_command_t* *cmdinfo)

host function to do a transaction

esp_err_t (***deinit**)(void)

host function to deinitialize the driver

esp_err_t (*__deinit_p__)(int slot)

host function to deinitialize the driver, called with the `slot`

esp_err_t (*__io_int_enable__)(int slot)

Host function to enable SDIO interrupt line

esp_err_t (*__io_int_wait__)(int slot, TickType_t timeout_ticks)

Host function to wait for SDIO interrupt line to be active

int __command_timeout_ms__

timeout, in milliseconds, of a single command. Set to 0 to use the default value.

esp_err_t (*__get_real_freq__)(int slot, int *real_freq)

Host function to provide real working freq, based on SDMMC controller setup

struct __sdmmc_card_t__

SD/MMC card information structure

Public Members

sdmmc_host_t __host__

Host with which the card is associated

uint32_t __ocr__

OCR (Operation Conditions Register) value

sdmmc_cid_t __cid__

decoded CID (Card IDentification) register value

sdmmc_response_t __raw_cid__

raw CID of MMC card to be decoded after the CSD is fetched in the data transfer mode

sdmmc_csd_t __csd__

decoded CSD (Card-Specific Data) register value

sdmmc_scr_t __scr__

decoded SCR (SD card Configuration Register) value

sdmmc_ssr_t __ssr__

decoded SSR (SD Status Register) value

sdmmc_ext_csd_t __ext_csd__

decoded EXT_CSD (Extended Card Specific Data) register value

uint16_t __rca__

RCA (Relative Card Address)

uint16_t **max_freq_khz**

Maximum frequency, in kHz, supported by the card

int **real_freq_khz**

Real working frequency, in kHz, configured on the host controller

uint32_t **is_mem**

Bit indicates if the card is a memory card

uint32_t **is_sdio**

Bit indicates if the card is an IO card

uint32_t **is_mmc**

Bit indicates if the card is MMC

uint32_t **num_io_functions**

If is_sdio is 1, contains the number of IO functions on the card

uint32_t **log_bus_width**

log2(bus width supported by card)

uint32_t **is_ddr**

Card supports DDR mode

uint32_t **reserved**

Reserved for future expansion

Macros

SDMMC_HOST_FLAG_1BIT

host supports 1-line SD and MMC protocol

SDMMC_HOST_FLAG_4BIT

host supports 4-line SD and MMC protocol

SDMMC_HOST_FLAG_8BIT

host supports 8-line MMC protocol

SDMMC_HOST_FLAG_SPI

host supports SPI protocol

SDMMC_HOST_FLAG_DDR

host supports DDR mode for SD/MMC

SDMMC_HOST_FLAG_DEINIT_ARG

host `deinit` function called with the slot argument

SDMMC_FREQ_DEFAULT

SD/MMC Default speed (limited by clock divider)

SDMMC_FREQ_HIGHSPEED

SD High speed (limited by clock divider)

SDMMC_FREQ_PROBING

SD/MMC probing speed

SDMMC_FREQ_52M

MMC 52MHz speed

SDMMC_FREQ_26M

MMC 26MHz speed

Type Definitions

typedef uint32_t **sdmmc_response_t**[4]

SD/MMC command response buffer

Enumerations

enum **sdmmc_erase_arg_t**

SD/MMC erase command(38) arguments SD: ERASE: Erase the write blocks, physical/hard erase.

DISCARD: Card may deallocate the discarded blocks partially or completely. After discard operation the previously written data may be partially or fully read by the host depending on card implementation.

MMC: ERASE: Does TRIM, applies erase operation to write blocks instead of Erase Group.

DISCARD: The Discard function allows the host to identify data that is no longer required so that the device can erase the data if necessary during background erase events. Applies to write blocks instead of Erase Group After discard operation, the original data may be remained partially or fully accessible to the host dependent on device.

Values:

enumerator **SDMMC_ERASE_ARG**

Erase operation on SD, Trim operation on MMC

enumerator **SDMMC_DISCARD_ARG**

Discard operation for SD/MMC

2.9.7 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 驱动* 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
- *OTA API* 提供了高层 API 用于更新存储在 flash 中的 app 固件。
- *NVS API* 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

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; 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; 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; 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.

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

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_ANY**
Used to search for partitions with any subtype.

2.9.8 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。
- 被删除文件通常不会被完全清除，会在文件系统中遗留下无法使用的部分。
- 如果 ESP32-C2 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:

```
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-C2（偏移量：0x110000）：

```
python esptool.py --chip esp32c2 --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

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.

2.9.9 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，您可以使用 /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
// (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 vfs/vfs_uart.c 中 UART 外设的 VFS 驱动，尤其是函数*esp_vfs_dev_uart_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，您可以禁用*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` 在 newlib `sys/types.h` 中定义。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。您可以在 menuconfig 中更改上述参数。

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 `esp_vfs_dev_uart_use_driver` 函数来指导 VFS 使用驱动中断、读写阻塞功能等。您也可以调用 `esp_vfs_dev_uart_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 `esp_vfs_dev_uart_port_set_rx_line_endings` 和 `esp_vfs_dev_uart_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)。

Event fds

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

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.

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.

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 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.

Type Definitions

typedef int **esp_vfs_id_t**

Header File

- components/vfs/include/esp_vfs_dev.h

Functions

void **esp_vfs_dev_uart_register**(void)

add /dev/uart virtual filesystem driver

This function is called from startup code to enable serial output

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

int **esp_vfs_dev_uart_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 successed, or -1 when an error (specified by errno) have occurred.

int **esp_vfs_dev_uart_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 successed, or -1 when an error (specified by errno) have occurred.

void **esp_vfs_dev_uart_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 **esp_vfs_dev_uart_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

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/vfs/include/esp_vfs_eventfd.h

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 maxinum number of eventfds supported

Macros

EFD_SUPPORT_ISR

ESP_VFS_EVENTD_CONFIG_DEFAULT()

2.9.10 磨损均衡 API

概述

ESP32-C2 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

函数

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

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.

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

中层 API 参考

Header File

- components/wear_levelling/include/wear_levelling.h

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 allocation was successfully;
- ESP_ERR_INVALID_ARG, if WL allocation was unsuccessful;
- ESP_ERR_NO_MEM, if there was no memory to allocate WL components;

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 completed successfully;
- 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 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 result of function wl_sector_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;
- 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 that are related to the 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. Pointer must be non-NULL and 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 size of the WL storage.

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。

2.10 System API

2.10.1 App Image Format

An application image consists of the following structures:

1. The *esp_image_header_t* structure describes the mode of SPI flash and the count of memory segments.
2. The *esp_image_segment_header_t* structure describes each segment, its length, and its location in ESP32-C2's memory, followed by the data with a length of `data_len`. The data offset for each segment in the image is calculated in the following way:

- offset for 0 Segment = sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*).
- offset for 1 Segment = offset for 0 Segment + length of 0 Segment + sizeof(*esp_image_segment_header_t*).
- offset for 2 Segment = offset for 1 Segment + length of 1 Segment + sizeof(*esp_image_segment_header_t*).
- ...

The count of each segment is defined in the `segment_count` field that is stored in *esp_image_header_t*. The count cannot be more than *ESP_IMAGE_MAX_SEGMENTS*.

To get the list of your image segments, please run the following command:

```
esptool.py --chip esp32c2 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)
```

You can also see the information on segments in the ESP-IDF logs while your application is booting:

```
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
```

(下页继续)

(续上页)

```
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
```

For more details on the type of memory segments and their address ranges, see *ESP32-C2 Technical Reference Manual* > *System and Memory* > *Internal Memory* [PDF].

3. The image has a single checksum byte after the last segment. This byte is written on a sixteen byte padded boundary, so the application image might need padding.
4. If the `hash_appended` field from *`esp_image_header_t`* is set then a SHA256 checksum will be appended. The value of the SHA256 hash is calculated on the range from the first byte and up to this field. The length of this field is 32 bytes.
5. If the option *CONFIG_SECURE_SIGNED_APPS_SCHEME* is set to ECDSA then the application image will have an additional 68 bytes for an ECDSA signature, which includes:

- version word (4 bytes),
- signature data (64 bytes).

6. If the option *CONFIG_SECURE_SIGNED_APPS_SCHEME* is set to RSA or ECDSA (V2) then the application image will have an additional signature sector of 4K size. For more details on the format of this signature sector, please refer to *Signature Block Format*.

Application Description

The `DROM` segment of the application binary starts with the *`esp_app_desc_t`* structure which carries specific fields describing the application:

- `magic_word` - the magic word for the esp_app_desc structure.
- `secure_version` - see *Anti-rollback*.
- `version` - see *App version*. *
- `project_name` is filled from `PROJECT_NAME`. *
- `time` and `date` - compile time and date.
- `idf_ver` - version of ESP-IDF. *
- `app_elf_sha256` - contains sha256 hash for the application ELF file.

* - The maximum length is 32 characters, including null-termination character. For example, if the length of `PROJECT_NAME` exceeds 31 characters, the excess characters will be disregarded.

This structure is useful for identification of images uploaded via Over-the-Air (OTA) updates because it has a fixed offset = sizeof(*`esp_image_header_t`*) + sizeof(*`esp_image_segment_header_t`*). As soon as a device receives the first fragment containing this structure, it has all the information to determine whether the update should be continued with or not.

To obtain the *`esp_app_desc_t`* structure for the currently running application, use *`esp_app_get_description()`*.

To obtain the *`esp_app_desc_t`* structure for another OTA partition, use *`esp_ota_get_partition_description()`*.

Adding a Custom Structure to an Application

Users also have the opportunity to have similar structure with a fixed offset relative to the beginning of the image. The following pattern can be used to add a custom structure to your image:

```
const __attribute__((section(".rodata_custom_desc"))) esp_custom_app_desc_t custom_
↪app_desc = { ... }
```

Offset for custom structure is sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*) + sizeof(*esp_app_desc_t*).

To guarantee that the custom structure is located in the image even if it is not used, you need to add `target_link_libraries(${COMPONENT_TARGET} "-u custom_app_desc")` into `CMakeLists.txt`.

API Reference

Header File

- components/bootloader_support/include/esp_app_format.h

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_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

2.10.2 Application Level Tracing

Overview

IDF provides a useful feature for program behavior analysis called **Application Level Tracing**. The feature can be enabled in menuconfig and allows transfer of arbitrary data between the host and ESP32-C2 via JTAG interface with minimal overhead on program execution. Developers can use this library to send application specific state of execution to the host and receive commands or other type of information in the opposite direction at runtime. The main use cases of this library are:

1. Collecting application specific data, see 特定应用程序的跟踪
2. Lightweight logging to the host, see 记录日志到主机
3. System behaviour analysis, see 基于 *SEGGER SystemView* 的系统行为分析

API Reference

Header File

- components/app_trace/include/esp_app_trace.h

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

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

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.

2.10.3 Call function with external stack

Overview

A given function can be executed with a user allocated stack space which is independent of current task stack, this mechanism can be used to save stack space wasted by tasks which call a common function with intensive stack usage such as *printf*. The given function can be called inside the shared stack space which is a callback function deferred by calling `esp_execute_shared_stack_function()`, passing that function as parameter.

Usage

`esp_execute_shared_stack_function()` takes four arguments:

- a mutex object allocated by the caller, which is used to protect if the same function shares its allocated stack
- a pointer to the top of stack used for that function
- the size of stack in bytes
- a pointer to the shared stack function

The user defined function will be deferred as a callback and can be called using the user allocated space without taking space from current task stack.

The usage may look like the code below:

```
void external_stack_function(void)
{
    printf("Executing this printf from external stack! \n");
}

//Let's suppose we want to call printf using a separated stack space
//allowing the app to reduce its stack size.
void app_main()
{
    //Allocate a stack buffer, from heap or as a static form:
    portSTACK_TYPE *shared_stack = malloc(8192 * sizeof(portSTACK_TYPE));
    assert(shared_stack != NULL);

    //Allocate a mutex to protect its usage:
    SemaphoreHandle_t printf_lock = xSemaphoreCreateMutex();
    assert(printf_lock != NULL);

    //Call the desired function using the macro helper:
    esp_execute_shared_stack_function(printf_lock,
                                      shared_stack,
                                      8192,
                                      external_stack_function);

    vSemaphoreDelete(printf_lock);
    free(shared_stack);
}
```

API Reference

Header File

- components/esp_system/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 user defined shared stack space function.

备注： if either lock, stack or stack size is invalid, the expression will be called using the current stack.

参数

- **lock** –Mutex object to protect in case of shared stack
- **stack** –Pointer to user alocated 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)

2.10.4 Chip Revision

Overview

A new chip versioning logic was introduced in new chips. Chips have several eFuse version fields:

- Major wafer version (`WAFER_VERSION_MAJOR` eFuse)
- Minor wafer version (`WAFER_VERSION_MINOR` eFuse)
- Ignore maximal revision (`DISABLE_WAFER_VERSION_MAJOR` eFuse)

The new versioning logic is being introduced to distinguish changes in chips as breaking changes and non-breaking changes. Chips with non-breaking changes can run the same software as the previous chip. The previous chip means that the major version is the same.

If the newly released chip does not have breaking changes, that means it can run the same software as the previous chip, then in that chip we keep the same major version and increment the minor version by 1. Otherwise, if there is a breaking change in the newly released chip, meaning it can not run the same software as the previous chip, then in that chip we increase the major version and set the minor version to 0.

The software supports a number of revisions, from the minimum to the maximum (the min/max configs are defined in Kconfig). If the software is unaware of a new chip (when the chip version is out of range), it will refuse to run on it unless the Ignore maximum revision restrictions bit is set. This bit removes the upper revision limit.

Minimum versions limits the software to only run on a chip revision that is high enough to support some features. Maximum version is the maximum version that is well-supported by current software. When chip version is above the maximum version, software will reject to boot, because it may not work on, or work with risk on the chip.

Adding the major and minor wafer revision make the versioning logic is branchable.

备注： The previous versioning logic was based on a single eFuse version field (`WAFER_VERSION`). This approach makes it impossible to mark chips as breaking or non-breaking changes, and the versioning logic becomes linear.

Using the branched versioning scheme allows us to support more chips in the software without updating the software when a new released compatible chip is used. Thus, the software will be compatible with as many new chip revisions as possible. If the software is no longer compatible with a new chip with breaking changes, the software will abort.

Revisions

ECO	Revision (Major.Minor)
ECO0	v0.0
ECO1	v1.0

Chip Revision `vX.Y`, where:

- `X` means Major wafer version. If it is changed, it means that the current software version is not compatible with this released chip and the software must be updated to use this chip.
- `Y` means Minor wafer version. If it is changed that means the current software version is compatible with the released chip, and there is no need to update the software.

The `vX.Y` chip version format will be used further instead of the ECO number.

Representing Revision Requirement Of A Binary Image

The 2nd stage bootloader and the application binary images have the *`esp_image_header_t`* header, which stores the revision numbers of the chip on which the software can be run. This header has 3 fields related to revisions:

- `min_chip_rev` - Minimal chip MAJOR revision required by image (but for ESP32-C3 it is MINOR revision). Its value is determined by *CONFIG_ESP32C2_REV_MIN*.
- `min_chip_rev_full` - Minimal chip MINOR revision required by image in format: `major * 100 + minor`. Its value is determined by *CONFIG_ESP32C2_REV_MIN*.
- `max_chip_rev_full` - Maximal chip revision required by image in format: `major * 100 + minor`. Its value is determined by `CONFIG_ESP32C2_REV_MAX_FULL`. It can not be changed by user. Only Espressif can change it when a new version will be supported in IDF.

Chip Revision APIs

These APIs helps to get chip revision from eFuses:

- *`efuse_hal_chip_revision()`*. It returns revision in the `major * 100 + minor` format.
- *`efuse_hal_get_major_chip_version()`*. It returns Major revision.
- *`efuse_hal_get_minor_chip_version()`*. It returns Minor revision.

The following Kconfig definitions (in `major * 100 + minor` format) that can help add the chip revision dependency to the code:

- `CONFIG_ESP32C2_REV_MIN_FULL`
- `CONFIG_ESP_REV_MIN_FULL`
- `CONFIG_ESP32C2_REV_MAX_FULL`
- `CONFIG_ESP_REV_MAX_FULL`

Maximal And Minimal Revision Restrictions

The order for checking the minimum and maximum revisions:

1. The 1st stage bootloader (ROM bootloader) does not check minimal and maximal revision fields from *`esp_image_header_t`* before running the 2nd stage bootloader.
2. The 2nd stage bootloader checks at the initialization phase that bootloader itself can be launched on the chip of this revision. It extracts the minimum revision from the header of the bootloader image and checks against the chip revision from eFuses. If the chip revision is less than the minimum revision, the bootloader refuses to boot up and aborts. The maximum revision is not checked at this phase.
3. Then the 2nd stage bootloader checks the revision requirements of the application. It extracts the minimum and maximum revisions from the header of the application image and checks against the chip revision from eFuses. If the chip revision is less than the minimum revision or higher than the maximum revision, the bootloader refuses to boot up and aborts. However, if the Ignore maximal revision bit is set, the maximum revision constraint can be ignored. The ignore bit is set by the customer themself when there is confirmation that the software is able to work with this chip revision.
4. Further, at the OTA update stage, the running application checks if the new software matches the chip revision. It extracts the minimum and maximum revisions from the header of the new application image and checks against the chip revision from eFuses. It checks for revision matching in the same way that the bootloader does, so that the chip revision is between the min and max revisions (logic of ignoring max revision also applies).

Issues

1. If the 2nd stage bootloader is run on the chip revision < minimum revision shown in the image, a reboot occurs. The following message will be printed:

```
Image requires chip rev >= v3.0, but chip is v1.0
```

To resolve this issue:

- make sure the chip you are using is suitable for the software, or use a chip with the required minimum revision or higher.
- update the software with *CONFIG_ESP32C2_REV_MIN* to get it <= the revision of chip being used

2. If application does not match minimal and maximal chip revisions, a reboot occurs. The following message will be printed:

```
Image requires chip rev <= v2.99, but chip is v3.0
```

To resolve this issue, update the IDF to a newer version that supports the used chip (`CONFIG_ESP32C2_REV_MAX_FULL`). Another way to fix this is to set the `Ignore maximal revision` bit in eFuse or use a chip that is suitable for the software.

Backward Compatible With Bootloaders Built By Older ESP-IDF Versions

ESP32-C2 chip support was added in IDF 5.0. The bootloader is able to detect any chip versions in range v0.0 - v3.15.

Please check the chip version using `esptool chip_id` command.

API Reference

Header File

- components/hal/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

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.

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.

2.10.5 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。

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

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。

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

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.

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_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' ]
*
```

(下页继续)

(续上页)

```
* - 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.

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_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

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

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_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.

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.

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()

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 struct *esp_console_repl_s* **esp_console_repl_t**

Type defined for console REPL.

2.10.6 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-C2 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-C2 Technical Reference Manual* > *eFuse Controller (eFuse)* [PDF]. Some eFuse bits are available for user applications.

ESP32-C2 has 4 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_KEY0) can be used as key (for secure_boot or flash_encryption) or for user 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 will check with the *common* CSV file.

CSV files:

- common (*esp_efuse_table.csv*) - contains eFuse fields which are used inside the 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_BLK3), 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_* will be added to the name, and this field name will be available in the code. This name will be 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 will be placed for this field. Available EFUSE_BLK0..EFUSE_BLK3.

bit_start Start bit number (0..255). The bit_start field can be omitted. In this case, it will be 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 can not be omitted. This field also may be `MAX_BLK_LEN` in this case, the field length will have 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, this will indicate 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 will 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 can not 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 IDF names for structured efuse fields should be unique. The `efuse_table_gen` tool will generate the final names where the dot will be replaced by _. The names for using in 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 FEILD, EFUSE_BLK3, 0, 50  out of range  FEILD.MAJOR_NUMBER, EFUSE_BLK3,␣
↪60, 32
```

Solution: Change `bit_start` for `FIELD.MAJOR_NUMBER` from 60 to 0, so `MAJOR_NUMBER` is in the `FEILD` 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 esp32c2 esp32c2/esp_efuse_table.csv
```

After generation in the folder $IDF_PATH/components/efuse/*esp32c2* 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 esp32c2 esp32c2/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-C2 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_BLK3 use Reed-Solomon coding scheme that supports up to 5 bytes of automatic error correction. Software will encode 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.

For frequently used fields, special functions are made, like this *esp_efuse_get_pkg_ver()*.

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 esp32c2/esp_efuse_table.csv --info

Parsing efuse CSV input file $IDF_PATH/components/efuse/esp32c2/esp_efuse_table.
↪csv ...
Verifying efuse table...
Max number of bits in BLK 256
Sorted efuse table:
#       field_name                              efuse_block     bit_start       ␣
↪bit_count
```

(下页继续)

(续上页)

```
1       WR_DIS                                  EFUSE_BLK0          0                ␣
↪8
2       WR_DIS.RD_DIS                           EFUSE_BLK0          0                ␣
↪1
3       WR_DIS.WDT_DELAY_SEL                    EFUSE_BLK0          1                ␣
↪1
4       WR_DIS.DIS_PAD_JTAG                     EFUSE_BLK0          1                ␣
↪1
5       WR_DIS.DIS_DOWNLOAD_ICACHE              EFUSE_BLK0          1                ␣
↪1
6       WR_DIS.DIS_DOWNLOAD_MANUAL_ENCRYPT      EFUSE_BLK0          2                ␣
↪1
7       WR_DIS.SPI_BOOT_CRYPT_CNT               EFUSE_BLK0          2                ␣
↪1
8       WR_DIS.XTS_KEY_LENGTH_256               EFUSE_BLK0          2                ␣
↪1
9       WR_DIS.SECURE_BOOT_EN                   EFUSE_BLK0          2                ␣
↪1
10      WR_DIS.UART_PRINT_CONTROL               EFUSE_BLK0          3                ␣
↪1
11      WR_DIS.FORCE_SEND_RESUME                EFUSE_BLK0          3                ␣
↪1
12      WR_DIS.DIS_DOWNLOAD_MODE                EFUSE_BLK0          3                ␣
↪1
13      WR_DIS.DIS_DIRECT_BOOT                  EFUSE_BLK0          3                ␣
↪1
14      WR_DIS.ENABLE_SECURITY_DOWNLOAD         EFUSE_BLK0          3                ␣
↪1
15      WR_DIS.FLASH_TPUW                       EFUSE_BLK0          3                ␣
↪1
16      WR_DIS.SECURE_VERSION                   EFUSE_BLK0          4                ␣
↪1
17      WR_DIS.CUSTOM_MAC_USED                  EFUSE_BLK0          4                ␣
↪1
18      WR_DIS.DISABLE_WAFER_VERSION_MAJOR      EFUSE_BLK0          4                ␣
↪1
19      WR_DIS.DISABLE_BLK_VERSION_MAJOR        EFUSE_BLK0          4                ␣
↪1
20      WR_DIS.CUSTOM_MAC                       EFUSE_BLK0          5                ␣
↪1
21      WR_DIS.MAC                              EFUSE_BLK0          6                ␣
↪1
22      WR_DIS.WAFER_VERSION_MINOR              EFUSE_BLK0          6                ␣
↪1
23      WR_DIS.WAFER_VERSION_MAJOR              EFUSE_BLK0          6                ␣
↪1
24      WR_DIS.PKG_VERSION                      EFUSE_BLK0          6                ␣
↪1
25      WR_DIS.BLK_VERSION_MINOR                EFUSE_BLK0          6                ␣
↪1
26      WR_DIS.BLK_VERSION_MAJOR                EFUSE_BLK0          6                ␣
↪1
27      WR_DIS.OCODE                            EFUSE_BLK0          6                ␣
↪1
28      WR_DIS.TEMP_CALIB                       EFUSE_BLK0          6                ␣
↪1
29      WR_DIS.ADC1_INIT_CODE_ATTEN0            EFUSE_BLK0          6                ␣
↪1
30      WR_DIS.ADC1_INIT_CODE_ATTEN3            EFUSE_BLK0          6                ␣
↪1
31      WR_DIS.ADC1_CAL_VOL_ATTEN0              EFUSE_BLK0          6                ␣
↪1
```

(下页继续)

(续上页)

```
32      WR_DIS.ADC1_CAL_VOL_ATTEN3              EFUSE_BLK0          6                ␣
↪1
33      WR_DIS.DIG_DBIAS_HVT                    EFUSE_BLK0          6                ␣
↪1
34      WR_DIS.DIG_LDO_SLP_DBIAS2               EFUSE_BLK0          6                ␣
↪1
35      WR_DIS.DIG_LDO_SLP_DBIAS26              EFUSE_BLK0          6                ␣
↪1
36      WR_DIS.DIG_LDO_ACT_DBIAS26              EFUSE_BLK0          6                ␣
↪1
37      WR_DIS.DIG_LDO_ACT_STEPD10              EFUSE_BLK0          6                ␣
↪1
38      WR_DIS.RTC_LDO_SLP_DBIAS13              EFUSE_BLK0          6                ␣
↪1
39      WR_DIS.RTC_LDO_SLP_DBIAS29              EFUSE_BLK0          6                ␣
↪1
40      WR_DIS.RTC_LDO_SLP_DBIAS31              EFUSE_BLK0          6                ␣
↪1
41      WR_DIS.RTC_LDO_ACT_DBIAS31              EFUSE_BLK0          6                ␣
↪1
42      WR_DIS.RTC_LDO_ACT_DBIAS13              EFUSE_BLK0          6                ␣
↪1
43      WR_DIS.ADC_CALIBRATION_3                EFUSE_BLK0          6                ␣
↪1
44      WR_DIS.BLOCK_KEY0                       EFUSE_BLK0          7                ␣
↪1
45      RD_DIS                                  EFUSE_BLK0          32               ␣
↪2
46      RD_DIS.KEY0                             EFUSE_BLK0          32               ␣
↪2
47      RD_DIS.KEY0.LOW                         EFUSE_BLK0          32               ␣
↪1
48      RD_DIS.KEY0.HI                          EFUSE_BLK0          33               ␣
↪1
49      WDT_DELAY_SEL                           EFUSE_BLK0          34               ␣
↪2
50      DIS_PAD_JTAG                            EFUSE_BLK0          36               ␣
↪1
51      DIS_DOWNLOAD_ICACHE                     EFUSE_BLK0          37               ␣
↪1
52      DIS_DOWNLOAD_MANUAL_ENCRYPT             EFUSE_BLK0          38               ␣
↪1
53      SPI_BOOT_CRYPT_CNT                      EFUSE_BLK0          39               ␣
↪3
54      XTS_KEY_LENGTH_256                      EFUSE_BLK0          42               ␣
↪1
55      UART_PRINT_CONTROL                      EFUSE_BLK0          43               ␣
↪2
56      FORCE_SEND_RESUME                       EFUSE_BLK0          45               ␣
↪1
57      DIS_DOWNLOAD_MODE                       EFUSE_BLK0          46               ␣
↪1
58      DIS_DIRECT_BOOT                         EFUSE_BLK0          47               ␣
↪1
59      ENABLE_SECURITY_DOWNLOAD                EFUSE_BLK0          48               ␣
↪1
60      FLASH_TPUW                              EFUSE_BLK0          49               ␣
↪4
61      SECURE_BOOT_EN                          EFUSE_BLK0          53               ␣
↪1
62      SECURE_VERSION                          EFUSE_BLK0          54               ␣
↪4
```

(下页继续)

(续上页)

```
63      CUSTOM_MAC_USED                         EFUSE_BLK0          58          ␣
↪1
64      DISABLE_WAFER_VERSION_MAJOR             EFUSE_BLK0          59          ␣
↪1
65      DISABLE_BLK_VERSION_MAJOR               EFUSE_BLK0          60          ␣
↪1
66      USER_DATA                               EFUSE_BLK1          0           ␣
↪88
67      USER_DATA.MAC_CUSTOM                    EFUSE_BLK1          0           ␣
↪48
68      MAC                                     EFUSE_BLK2          0           ␣
↪8
69      MAC                                     EFUSE_BLK2          8           ␣
↪8
70      MAC                                     EFUSE_BLK2          16          ␣
↪8
71      MAC                                     EFUSE_BLK2          24          ␣
↪8
72      MAC                                     EFUSE_BLK2          32          ␣
↪8
73      MAC                                     EFUSE_BLK2          40          ␣
↪8
74      WAFER_VERSION_MINOR                     EFUSE_BLK2          48          ␣
↪4
75      WAFER_VERSION_MAJOR                     EFUSE_BLK2          52          ␣
↪2
76      PKG_VERSION                             EFUSE_BLK2          54          ␣
↪3
77      BLK_VERSION_MINOR                       EFUSE_BLK2          57          ␣
↪3
78      BLK_VERSION_MAJOR                       EFUSE_BLK2          60          ␣
↪2
79      OCODE                                   EFUSE_BLK2          62          ␣
↪7
80      TEMP_CALIB                              EFUSE_BLK2          69          ␣
↪9
81      ADC1_INIT_CODE_ATTEN0                   EFUSE_BLK2          78          ␣
↪8
82      ADC1_INIT_CODE_ATTEN3                   EFUSE_BLK2          86          ␣
↪5
83      ADC1_CAL_VOL_ATTEN0                     EFUSE_BLK2          91          ␣
↪8
84      ADC1_CAL_VOL_ATTEN3                     EFUSE_BLK2          99          ␣
↪6
85      DIG_DBIAS_HVT                           EFUSE_BLK2          105         ␣
↪5
86      DIG_LDO_SLP_DBIAS2                      EFUSE_BLK2          110         ␣
↪7
87      DIG_LDO_SLP_DBIAS26                     EFUSE_BLK2          117         ␣
↪8
88      DIG_LDO_ACT_DBIAS26                     EFUSE_BLK2          125         ␣
↪6
89      DIG_LDO_ACT_STEPD10                     EFUSE_BLK2          131         ␣
↪4
90      RTC_LDO_SLP_DBIAS13                     EFUSE_BLK2          135         ␣
↪7
91      RTC_LDO_SLP_DBIAS29                     EFUSE_BLK2          142         ␣
↪9
92      RTC_LDO_SLP_DBIAS31                     EFUSE_BLK2          151         ␣
↪6
93      RTC_LDO_ACT_DBIAS31                     EFUSE_BLK2          157         ␣
↪6
```

(下页继续)

(续上页)

```
94      RTC_LDO_ACT_DBIAS13                     EFUSE_BLK2        163              ␣
↪8
95      ADC_CALIBRATION_3                       EFUSE_BLK2        192              ␣
↪11
96      KEY0                                    EFUSE_BLK3         0               ␣
↪256
97      KEY0.FE_256BIT                          EFUSE_BLK3         0               ␣
↪256
98      KEY0.FE_128BIT                          EFUSE_BLK3         0               ␣
↪128
99      KEY0.SB_128BIT                          EFUSE_BLK3        128              ␣
↪128

Used bits in efuse table:
EFUSE_BLK0
[0 7] [0 1] [1 1] [1 2] [2 2] ... [6 6] [6 6] [6 6] [6 6] [6 6] [6 6] [6 6] [6 6]␣
↪[6 6] [6 7] [32 33] [32 33] [32 60]
EFUSE_BLK1
[0 87] [0 47]
EFUSE_BLK2
[0 170] [192 202]
EFUSE_BLK3
[0 255] [0 255] [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
```

(下页继续)

(续上页)

```
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
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; // can NOT 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,
        "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* will virtualize 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-C2 eFuse bits - espefuse.py.

```
espefuse.py -p PORT summary

espefuse.py v4.6-dev
Connecting....
Detecting chip type... ESP32-C2

=== Run "summary" command ===
EFUSE_NAME (Block) Description  = [Meaningful Value] [Readable/Writeable] (Hex␣
↪Value)
----------------------------------------------------------------------------␣
↪-----
Calibration fuses:
OCODE (BLOCK2)                                     OCode                      ␣
↪                    = 78 R/W (0b1001110)
TEMP_CALIB (BLOCK2)                                Temperature calibration data  ␣
↪                    = -7.4 R/W (0b101001010)
ADC1_INIT_CODE_ATTEN0 (BLOCK2)                     ADC1 init code at atten0      ␣
↪                    = 28 R/W (0x07)
ADC1_INIT_CODE_ATTEN3 (BLOCK2)                     ADC1 init code at atten3      ␣
↪                    = 0 R/W (0b10000)
ADC1_CAL_VOL_ATTEN0 (BLOCK2)                       ADC1 calibration voltage at␣
↪atten0                  = -44 R/W (0x8b)
ADC1_CAL_VOL_ATTEN3 (BLOCK2)                       ADC1 calibration voltage at␣
↪atten3                  = 16 R/W (0b000100)
DIG_DBIAS_HVT (BLOCK2)                             BLOCK2 digital dbias when hvt  ␣
↪                    = -16 R/W (0b10100)
DIG_LDO_SLP_DBIAS2 (BLOCK2)                        BLOCK2 DIG_LDO_DBG0_DBIAS2    ␣
↪                    = -8 R/W (0b1000010)
DIG_LDO_SLP_DBIAS26 (BLOCK2)                       BLOCK2 DIG_LDO_DBG0_DBIAS26   ␣
↪                    = 24 R/W (0x06)
DIG_LDO_ACT_DBIAS26 (BLOCK2)                       BLOCK2 DIG_LDO_ACT_DBIAS26    ␣
↪                    = 16 R/W (0b000100)
DIG_LDO_ACT_STEPD10 (BLOCK2)                       BLOCK2 DIG_LDO_ACT_STEPD10    ␣
↪                    = 12 R/W (0x3)
RTC_LDO_SLP_DBIAS13 (BLOCK2)                       BLOCK2 DIG_LDO_SLP_DBIAS13    ␣
↪                    = 88 R/W (0b0010110)
RTC_LDO_SLP_DBIAS29 (BLOCK2)                       BLOCK2 DIG_LDO_SLP_DBIAS29    ␣
↪                    = 96 R/W (0b000011000)
RTC_LDO_SLP_DBIAS31 (BLOCK2)                       BLOCK2 DIG_LDO_SLP_DBIAS31    ␣
↪                    = 4 R/W (0b000001)
RTC_LDO_ACT_DBIAS31 (BLOCK2)                       BLOCK2 DIG_LDO_ACT_DBIAS31    ␣
↪                    = 24 R/W (0b000110)
RTC_LDO_ACT_DBIAS13 (BLOCK2)                       BLOCK2 DIG_LDO_ACT_DBIAS13    ␣
↪                    = 72 R/W (0x12)

Config fuses:
WR_DIS (BLOCK0)                                    Disable programming of␣
↪individual eFuses           = 0 R/W (0x00)
RD_DIS (BLOCK0)                                    Disable reading from BlOCK3   ␣
↪                    = 0 R/W (0b00)
UART_PRINT_CONTROL (BLOCK0)                        Set the default UARTboot␣
↪message output mode       = Enable R/W (0b00)
DIS_DIRECT_BOOT (BLOCK0)                           This bit set means disable␣
↪direct_boot mode        = False R/W (0b0)

Flash fuses:
FORCE_SEND_RESUME (BLOCK0)                         Set this bit to force ROM code␣
↪to send a resume co = False R/W (0b0)
                                                   mmand during SPI boot
FLASH_TPUW (BLOCK0)                                Configures flash waiting time␣
↪after power-up; in u = 0 R/W (0x0)
                                                   nit of ms. If the value is less␣
↪than 15; the waiti
```

(下页继续)

(续上页)

```
                                                    ng time is the configurable␣
↪value.  Otherwise; the
                                                     waiting time is twice the␣
↪configurable value

Identity fuses:
DISABLE_WAFER_VERSION_MAJOR (BLOCK0)                Disables check of wafer version␣
↪major               = False R/W (0b0)
DISABLE_BLK_VERSION_MAJOR (BLOCK0)                  Disables check of blk version␣
↪major                 = False R/W (0b0)
WAFER_VERSION_MINOR (BLOCK2)                        WAFER_VERSION_MINOR            ␣
↪                    = 2 R/W (0x2)
WAFER_VERSION_MAJOR (BLOCK2)                        WAFER_VERSION_MAJOR            ␣
↪                    = 1 R/W (0b01)
PKG_VERSION (BLOCK2)                                EFUSE_PKG_VERSION              ␣
↪                    = 1 R/W (0b001)
BLK_VERSION_MINOR (BLOCK2)                          Minor version of BLOCK2        ␣
↪                    = With calib R/W (0b001)
BLK_VERSION_MAJOR (BLOCK2)                          Major version of BLOCK2        ␣
↪                    = 0 R/W (0b00)

Jtag fuses:
DIS_PAD_JTAG (BLOCK0)                               Set this bit to disable pad␣
↪jtag                    = False R/W (0b0)

Mac fuses:
CUSTOM_MAC_USED (BLOCK0)                            True if MAC_CUSTOM is burned   ␣
↪                    = False R/W (0b0)
CUSTOM_MAC (BLOCK1)                                 Custom MAC address
   = 00:00:00:00:00:00 (OK) R/W
MAC (BLOCK2)                                        MAC address
   = 08:3a:8d:5c:4b:94 (OK) R/W

Security fuses:
DIS_DOWNLOAD_ICACHE (BLOCK0)                        The bit be set to disable␣
↪icache in download mode  = False R/W (0b0)
DIS_DOWNLOAD_MANUAL_ENCRYPT (BLOCK0)                The bit be set to disable␣
↪manual encryption        = False R/W (0b0)
SPI_BOOT_CRYPT_CNT (BLOCK0)                         Enables flash encryption when 1␣
↪or 3 bits are set  = Disable R/W (0b000)
                                                    and disables otherwise
XTS_KEY_LENGTH_256 (BLOCK0)                         Flash encryption key length    ␣
↪                    = 128 bits key R/W (0b0)
DIS_DOWNLOAD_MODE (BLOCK0)                          Set this bit to disable␣
↪download mode (boot_mode[3 = False R/W (0b0)
                                                    :0] = 0; 1; 2; 4; 5; 6; 7)
ENABLE_SECURITY_DOWNLOAD (BLOCK0)                   Set this bit to enable secure␣
↪UART download mode    = False R/W (0b0)
SECURE_BOOT_EN (BLOCK0)                             The bit be set to enable secure␣
↪boot                = False R/W (0b0)
SECURE_VERSION (BLOCK0)                             Secure version for anti-
↪rollback                    = 0 R/W (0x0)
BLOCK_KEY0 (BLOCK3)                                 BLOCK_KEY0 - 256-bits. 256-bit␣
↪key of Flash Encryp
   = 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
                                                    tion
BLOCK_KEY0_LOW_128 (BLOCK3)                         BLOCK_KEY0 - lower 128-bits.␣
↪128-bit key of Flash
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
                                                    Encryption
```

(下页继续)

(续上页)

```
BLOCK_KEY0_HI_128 (BLOCK3)                                BLOCK_KEY0 - higher 128-bits.␣
↪128-bits key of Secu
   = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
                                                          re Boot

Wdt fuses:
WDT_DELAY_SEL (BLOCK0)                                    RTC watchdog timeout threshold;␣
↪in unit of slow cl = 40000 R/W (0b00)
                                                          ock cycle
```

To get a dump for all eFuse registers.

```
espefuse.py -p PORT dump

espefuse.py v4.6-dev
Connecting....
Detecting chip type... ESP32-C2
BLOCK0          (BLOCK0          ) [0 ] read_regs: 00000000 00000000
BLOCK1          (BLOCK1          ) [1 ] read_regs: 00000000 00000000 00000000
BLOCK2          (BLOCK2          ) [2 ] read_regs: 8d5c4b94 8252083a 5c01e953␣
↪80d0a824 c0860b18 00006890 00000000 4b000000
BLOCK_KEY0      (BLOCK3          ) [3 ] read_regs: 00000000 00000000 00000000␣
↪00000000 00000000 00000000 00000000 00000000

BLOCK0          (BLOCK0          ) [0 ] err__regs: 00000000 00000000
EFUSE_RD_RS_ERR_REG         0x00000000

=== Run "dump" command ===
```

Header File

- components/efuse/esp32c2/include/esp_efuse_chip.h

Enumerations

enum **esp_efuse_block_t**

Type of eFuse blocks.

Values:

enumerator **EFUSE_BLK0**

Number of eFuse BLOCK0. REPEAT_DATA

enumerator **EFUSE_BLK1**

Number of eFuse BLOCK1. SYS_DATA_PART0

enumerator **EFUSE_BLK_SYS_DATA_PART0**

Number of eFuse BLOCK2. SYS_DATA_PART0

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. KEY0. whole block

enumerator **EFUSE_BLK_KEY0**

Number of eFuse BLOCK3. KEY0. whole block

enumerator **EFUSE_BLK_SECURE_BOOT**

enumerator **EFUSE_BLK_KEY_MAX**

enumerator **EFUSE_BLK_MAX**

Number of eFuse blocks

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 purposes (they are virtual because this chip has only fixed purposes for block)

Values:

enumerator **ESP_EFUSE_KEY_PURPOSE_USER**

whole BLOCK3

enumerator **ESP_EFUSE_KEY_PURPOSE_XTS_AES_128_KEY**

FE uses the whole BLOCK3 (key is 256-bits)

enumerator **ESP_EFUSE_KEY_PURPOSE_XTS_AES_128_KEY_DERIVED_FROM_128_EFUSE_BITS**

FE uses lower 128-bits of BLOCK3 (key is 128-bits)

enumerator **ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_V2**

SB uses higher 128-bits of BLOCK3 (key is 128-bits)

enumerator **ESP_EFUSE_KEY_PURPOSE_MAX**

MAX PURPOSE

Header File

- components/efuse/include/esp_efuse.h

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.

Note: 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.
...

// 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.

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.

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.

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.

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**[(1U)]

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

2.10.7 Error Codes and Helper Functions

This section lists definitions of common ESP-IDF error codes and several helper functions related to error handling.

For general information about error codes in ESP-IDF, see *Error Handling*.

For the full list of error codes defined in ESP-IDF, see *Error Code Reference*.

API Reference

Header File

- components/esp_common/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_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_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

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

There are items remained to retrieve

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**

2.10.8 ESP HTTPS OTA

Overview

`esp_https_ota` provides simplified APIs to perform firmware upgrades over HTTPS. It's an abstraction layer over existing OTA APIs.

Application Example

```
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) {
```

(下页继续)

(续上页)

```
        esp_restart();
    } else {
        return ESP_FAIL;
    }
    return ESP_OK;
}
```

Server Verification

Please refer to *ESP-TLS: TLS Server Verification* for more information on server verification. The root certificate (in PEM format) needs to be provided to the `esp_http_client_config_t::cert_pem` member.

备 注: The server-endpoint **root** certificate should be used for verification instead of any intermediate ones from the certificate chain. The reason being that the root certificate has the maximum validity and usually remains the same for a long period of time. Users can also use the `ESP x509 Certificate Bundle` feature for verification, which covers most of the trusted root certificates (using the `esp_http_client_config_t::crt_bundle_attach` member).

Partial Image Download over HTTPS

To use partial image download feature, enable `partial_http_download` configuration in `esp_https_ota_config_t`. When this configuration is enabled, firmware image will be downloaded in multiple HTTP requests of specified size. Maximum content length of each request can be specified by setting `max_http_request_size` to required value.

This option is useful while fetching image from a service like AWS S3, where mbedTLS Rx buffer size (*CONFIG_MBEDTLS_SSL_IN_CONTENT_LEN*) can be set to lower value which is not possible without enabling this configuration.

Default value of mbedTLS Rx buffer size is set to 16K. By using partial_http_download with max_http_request_size of 4K, size of mbedTLS Rx buffer can be reduced to 4K. With this configuration, memory saving of around 12K is expected.

Signature Verification

For additional security, signature of OTA firmware images can be verified. For that, refer *没有安全启动的安全 OTA 升级*

Advanced APIs

`esp_https_ota` also provides advanced APIs which can be used if more information and control is needed during the OTA process.

Example that uses advanced ESP_HTTPS_OTA APIs: system/ota/advanced_https_ota.

OTA Upgrades with Pre-Encrypted Firmware

To perform OTA upgrades with Pre-Encrypted Firmware, please enable *CONFIG_ESP_HTTPS_OTA_DECRYPT_CB* in component menuconfig.

Example that performs OTA upgrade with Pre-Encrypted Firmware: system/ota/pre_encrypted_ota.

OTA System Events

ESP HTTPS OTA has various events for which a handler can be triggered by *the Event Loop library* when the particular event occurs. The handler has to be registered using *esp_event_handler_register()*. This helps in event handling for ESP HTTPS OTA. *esp_https_ota_event_t* has all the events which can happen when performing OTA upgrade using ESP HTTPS OTA.

Event Handler Example

```
/* Event handler for catching system events */
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;
        }
    }
}
```

Expected data type for different ESP HTTPS OTA events in the system event loop:

- 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 Reference

Header File

- components/esp_https_ota/include/esp_https_ota.h

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 atleast 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_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

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

2.10.9 Event Loop Library

Overview

The event loop library allows components to declare events to which other components can register handlers –code which will execute when those events occur. This allows loosely coupled components to attach desired behavior to state changes of other components without application involvement. This also simplifies event processing by serializing and deferring code execution to another context.

One common use case is if a high level library is using the WiFi library: it may subscribe to *events produced by the Wi-Fi subsystem* directly and act on those events.

备注： Various modules of the Bluetooth stack deliver events to applications via dedicated callback functions instead of via the Event Loop Library.

Using `esp_event` APIs

There are two objects of concern for users of this library: events and event loops.

Events are occurrences of note. For example, for Wi-Fi, a successful connection to the access point may be an event. Events are referenced using a two part identifier which are discussed more *here*. Event loops are the vehicle by which events get posted by event sources and handled by event handler functions. These two appear prominently in the event loop library APIs.

Using this library roughly entails the following flow:

1. A user defines a function that should run when an event is posted to a loop. This function is referred to as the event handler. It should have the same signature as `esp_event_handler_t`.
2. An event loop is created using `esp_event_loop_create()`, which outputs a handle to the loop of type `esp_event_loop_handle_t`. Event loops created using this API are referred to as user event loops. There is, however, a special type of event loop called the default event loop which are discussed *here*.
3. Components register event handlers to the loop using `esp_event_handler_register_with()`. Handlers can be registered with multiple loops, more on that *here*.
4. Event sources post an event to the loop using `esp_event_post_to()`.
5. Components wanting to remove their handlers from being called can do so by unregistering from the loop using `esp_event_handler_unregister_with()`.
6. Event loops which are no longer needed can be deleted using `esp_event_loop_delete()`.

In code, the flow above may look like as follows:

```
// 1. Define the event handler
void run_on_event(void* handler_arg, esp_event_base_t base, int32_t id, void*␣
↪event_data)
{
    // Event handler logic
}

void app_main()
{
    // 2. A configuration structure of type esp_event_loop_args_t is needed to␣
↪specify the properties of the loop to be
    // created. A handle of type esp_event_loop_handle_t is obtained, which is␣
↪needed by the other APIs to reference the loop
    // to perform their operations on.
    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. Register event handler defined in (1). MY_EVENT_BASE and MY_EVENT_ID␣
↪specifies a hypothetical
    // event that handler run_on_event should execute on when it gets posted to␣
↪the loop.
    esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_
↪on_event, ...);

    ...

    // 4. Post events to the loop. This queues the event on the event loop. At␣
↪some point in time
    // the event loop executes the event handler registered to the posted event,␣
↪in this case run_on_event.
    // For simplicity sake this example calls esp_event_post_to from app_main, but␣
↪posting can be done from
    // any other tasks (which is the more interesting use case).
    esp_event_post_to(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, ...);

    ...

    // 5. Unregistering an unneeded handler
    esp_event_handler_unregister_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_
↪on_event);

    ...

    // 6. Deleting an unneeded event loop
    esp_event_loop_delete(loop_handle);
}
```

Declaring and defining events

As mentioned previously, events consists of two-part identifiers: the event base and the event ID. The event base identifies an independent group of events; the event ID identifies the event within that group. Think of the event base and event ID as a person's last name and first name, respectively. A last name identifies a family, and the first name

identifies a person within that family.

The event loop library provides macros to declare and define the event base easily.

Event base declaration:

```
ESP_EVENT_DECLARE_BASE(EVENT_BASE)
```

Event base definition:

```
ESP_EVENT_DEFINE_BASE(EVENT_BASE)
```

备注： In IDF, the base identifiers for system events are uppercase and are postfixed with `_EVENT`. For example, the base for Wi-Fi events is declared and defined as `WIFI_EVENT`, the Ethernet event base `ETHERNET_EVENT`, and so on. The purpose is to have event bases look like constants (although they are global variables considering the definitions of macros `ESP_EVENT_DECLARE_BASE` and `ESP_EVENT_DEFINE_BASE`).

For event ID's, declaring them as enumerations is recommended. Once again, for visibility, these are typically placed in public header files.

Event ID:

```
enum {
    EVENT_ID_1,
    EVENT_ID_2,
    EVENT_ID_3,
    ...
}
```

Default Event Loop

The default event loop is a special type of loop used for system events (Wi-Fi events, for example). The handle for this loop is hidden from the user. The creation, deletion, handler registration/unregistration and posting of events is done through a variant of the APIs for user event loops. The table below enumerates those variants, and the user event loops equivalent.

User Event Loops	Default Event Loops
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()*

If you compare the signatures for both, they are mostly similar except the for the lack of loop handle specification for the default event loop APIs.

Other than the API difference and the special designation to which system events are posted to, there is no difference to how default event loops and user event loops behave. It is even possible for users to post their own events to the default event loop, should the user opt to not create their own loops to save memory.

Notes on Handler Registration

It is possible to register a single handler to multiple events individually, i.e. using multiple calls to *esp_event_handler_register_with()*. For those multiple calls, the specific event base and event ID can be specified with which the handler should execute.

However, in some cases it is desirable for a handler to execute on (1) all events that get posted to a loop or (2) all events of a particular base identifier. This is possible using the special event base identifier `ESP_EVENT_ANY_BASE` and

special event ID ESP_EVENT_ANY_ID. These special identifiers may be passed as the event base and event ID arguments for *esp_event_handler_register_with()*.

Therefore, the valid arguments to *esp_event_handler_register_with()* are:

1. <event base>, <event ID> - handler executes when the event with base <event base> and event ID <event ID> gets posted to the loop
2. <event base>, ESP_EVENT_ANY_ID - handler executes when any event with base <event base> gets posted to the loop
3. ESP_EVENT_ANY_BASE, ESP_EVENT_ANY_ID - handler executes when any event gets posted to the loop

As an example, suppose the following handler registrations were performed:

```
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, ...);
```

If the hypothetical event MY_EVENT_BASE, MY_EVENT_ID is posted, all three handlers run_on_event_1, run_on_event_2, and run_on_event_3 would execute.

If the hypothetical event MY_EVENT_BASE, MY_OTHER_EVENT_ID is posted, only run_on_event_2 and run_on_event_3 would execute.

If the hypothetical event MY_OTHER_EVENT_BASE, MY_OTHER_EVENT_ID is posted, only run_on_event_3 would execute.

Handler Un-registering Itself In general, an event handler run by an event loop is *not allowed to do any (un)registering activity on that event loop*. There is one exception, though: un-registering itself is allowed for the handler. E.g., it is possible to do the following:

```
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);
    // ... post event MY_EVENT_BASE, MY_EVENT_ID and run loop at some point
}
```

Handler Registration and Handler Dispatch Order The general rule is that for handlers that match a certain posted event during dispatch, those which are registered first also gets executed first. The user can then control which handlers get executed first by registering them before other handlers, provided that all registrations are performed using a single task. If the user plans to take advantage of this behavior, caution must be exercised if there are multiple tasks registering handlers. While the 'first registered, first executed' behavior still holds true, the task which gets executed first will also get their handlers registered first. Handlers registered one after the other by a single task will still be dispatched in the order relative to each other, but if that task gets pre-empted in between registration by another task which also registers handlers; then during dispatch those handlers will also get executed in between.

Event loop profiling

A configuration option *CONFIG_ESP_EVENT_LOOP_PROFILING* can be enabled in order to activate statistics collection for all event loops created. The function `esp_event_dump()` can be used to output the collected statistics to a file stream. More details on the information included in the dump can be found in the `esp_event_dump()` API Reference.

Application Example

Examples on using the `esp_event` library can be found in system/esp_event. The examples cover event declaration, loop creation, handler registration and unregistration and event posting.

Other examples which also adopt esp_event library:

- NMEA Parser , which will decode the statements received from GPS.

API Reference

Header File

- components/esp_event/include/esp_event.h

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_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

---

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

---

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_to, 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
          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

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

Related Documents

2.10.10 FreeRTOS (Overview)

Overview

FreeRTOS is an open source real-time operating system kernel that acts as the operating system for ESP-IDF applications and is integrated into ESP-IDF as a component. The FreeRTOS component in ESP-IDF contains ports of the FreeRTOS kernel for all the CPU architectures used by ESP targets (i.e., Xtensa and RISC-V). Furthermore,

ESP-IDF provides different implementations of FreeRTOS in order to support SMP (Symmetric Multiprocessing) on multi-core ESP targets. This document provides an overview of the FreeRTOS component, the FreeRTOS implementations offered by ESP-IDF, and the common aspects across all implementations.

Implementations

The official FreeRTOS (henceforth referred to as Vanilla FreeRTOS) is a single-core RTOS. In order to support the various multi-core ESP targets, ESP-IDF supports different FreeRTOS implementations, namely **ESP-IDF FreeRTOS** and **Amazon SMP FreeRTOS**.

ESP-IDF FreeRTOS ESP-IDF FreeRTOS is a FreeRTOS implementation based on Vanilla FreeRTOS v10.4.3, but contains significant modifications to support SMP. ESP-IDF FreeRTOS only supports two cores at most (i.e., dual core SMP), but is more optimized for this scenario by design. For more details regarding ESP-IDF FreeRTOS and its modifications, please refer to the *FreeRTOS (ESP-IDF)* document.

备注： ESP-IDF FreeRTOS is currently the default FreeRTOS implementation for ESP-IDF.

Amazon SMP FreeRTOS Amazon SMP FreeRTOS is an SMP implementation of FreeRTOS that is officially supported by Amazon. Amazon SMP FreeRTOS is able to support N-cores (i.e., more than two cores). Amazon SMP FreeRTOS can be enabled via the *CONFIG_FREERTOS_SMP* option. For more details regarding Amazon SMP FreeRTOS, please refer to the official Amazon SMP FreeRTOS documentation.

警告： The Amazon SMP FreeRTOS implementation (and its port in ESP-IDF) are currently in experimental/beta state. Therefore, significant behavioral changes and breaking API changes can occur.

Configuration

Kernel Configuration Vanilla FreeRTOS requires that ports and applications configure the kernel by adding various `#define config...` macros to `FreeRTOSConfig.h`. Vanilla FreeRTOS supports a list of kernel configuration options which allow various kernel behaviors and features to be enabled or disabled.

However, for all FreeRTOS ports in ESP-IDF, the "FreeRTOSConfig.h" file is considered private and must not be modified by users. A large number of kernel configuration options in `FreeRTOSConfig.h` are hard coded as they are either required or not supported in ESP-IDF. All kernel configuration options that are configurable by the user will be exposed via menuconfig under `Component Config/FreeRTOS/Kernel`.

For the full list of user configurable kernel options, see 项目配置. The list below highlights some commonly used kernel configuration options:

- *CONFIG_FREERTOS_UNICORE* will run FreeRTOS only on CPU0. Note that this is **not equivalent to running Vanilla FreeRTOS**. Furthermore, this option may affect behavior of components other than freertos. For more details regarding the effects of running FreeRTOS on a single core, refer to *ESP-IDF FreeRTOS Single Core* (if using ESP-IDF FreeRTOS) or the official Amazon SMP FreeRTOS documentation. Alternatively, users can also search for occurrences of `CONFIG_FREERTOS_UNICORE` in the ESP-IDF components.

备注： As ESP32-C2 is a single core SoC, the *CONFIG_FREERTOS_UNICORE* configuration is always set.

- *CONFIG_FREERTOS_ENABLE_BACKWARD_COMPATIBILITY* enables backward compatibility with some FreeRTOS macros/types/functions that were deprecated from v8.0 onwards.

Port Configuration All other FreeRTOS related configuration options that are not part of the kernel configuration are exposed via menuconfig under `Component Config/FreeRTOS/Port`. These options configure aspects such as:

- The FreeRTOS ports themselves (e.g., tick timer selection, ISR stack size)
- Additional features added to the FreeRTOS implementation or ports

Using FreeRTOS

Application Entry Point Unlike Vanilla FreeRTOS, users of FreeRTOS in ESP-IDF **must never call** *`vTaskStartScheduler()`* and *`vTaskEndScheduler()`*. Instead, ESP-IDF will start FreeRTOS automatically. Users must define a `void app_main(void)` function which acts as the entry point for user's application and is automatically called on ESP-IDF startup.

- Typically, users would spawn the rest of their application's task from `app_main`.
- The `app_main` function is allowed to return at any point (i.e., before the application terminates).
- The `app_main` function is called from the `main` task.

Background Tasks During startup, ESP-IDF and FreeRTOS will automatically create multiple tasks that run in the background (listed in the the table below).

表 6: List of Tasks Created During Startup

Task Name	Description	Stack Size	Affinity	Priority
Idle Tasks (`IDLEx`)	An idle task (`IDLEx`) is created for (and pinned to) each CPU, where `x` is the CPU's number.	*CONFIG_FREERTOS_IDLE_TASK_ST*	CPU`x`	0
FreeRTOS Timer Task (`Tmr Svc`)	FreeRTOS will create the Timer Service/Daemon Task if any FreeRTOS Timer APIs are called by the application.	*CONFIG_FREERTOS_TIMER_TASK_S*	CPU0	*CONFIG_FREERTOS_TIMER_TASK_PR*
Main Task (`main`)	Task that simply calls `app_main`. This task will self delete when `app_main` returns	*CONFIG_ESP_MAIN_TASK_STACK_SI*	*CONFIG_ESP_MAIN_TASK_AFFINITY*	1
IPC Tasks (`ipcx`)	When *CONFIG_FREERTOS_UNICORE* is false, an IPC task (`ipcx`) is created for (and pinned to) each CPU. IPC tasks are used to implement the Inter-processor Call (IPC) feature.	*CONFIG_ESP_IPC_TASK_STACK_SIZ*	CPU`x`	24
ESP Timer Task (`esp_timer`)	ESP-IDF will create the ESP Timer Task used to process ESP Timer callbacks.	*CONFIG_ESP_TIMER_TASK_STACK_*	CPU0	22

备注： Note that if an application uses other ESP-IDF features (e.g., WiFi or Bluetooth), those features may create their own background tasks in addition to the tasks listed in the table above.

FreeRTOS Additions

ESP-IDF provides some supplemental features to FreeRTOS such as Ring Buffers, ESP-IDF style Tick and Idle Hooks, and TLSP deletion callbacks. See *FreeRTOS (Supplemental Features)* for more details.

FreeRTOS Heap

Vanilla FreeRTOS provides its own selection of heap implementations. However, ESP-IDF already implements its own heap (see *Heap Memory Allocation*), thus ESP-IDF does not make use of the heap implementations provided by Vanilla FreeRTOS. All FreeRTOS ports in ESP-IDF map FreeRTOS memory allocation/free calls (e.g., `pvPortMalloc()` and `pvPortFree()`) to ESP-IDF heap API (i.e., `heap_caps_malloc()` and `heap_caps_free()`). However, the FreeRTOS ports ensure that all dynamic memory allocated by FreeRTOS is placed in internal memory.

备注： If users wish to place FreeRTOS tasks/objects in external memory, users can use the following methods:

- Allocate the task/object using one of the `...CreateWithCaps()` API such as `xTaskCreateWithCaps()` and `xQueueCreateWithCaps()` (see *IDF Additional API* for more details).
- Manually allocate external memory for those objects using `heap_caps_malloc()`, then create the objects from the allocated memory using on of the `...CreateStatic()` FreeRTOS functions.

2.10.11 FreeRTOS (ESP-IDF)

Overview

The original FreeRTOS (hereinafter referred to as Vanilla FreeRTOS) is a small and efficient Real Time Operating System supported on many single-core MCUs and SoCs. However, to support numerous dual core ESP targets (such as the ESP32 and ESP32-S3), ESP-IDF provides a dual core SMP (Symmetric Multiprocessing) capable implementation of FreeRTOS, (hereinafter referred to as ESP-IDF FreeRTOS).

ESP-IDF FreeRTOS is based on Vanilla FreeRTOS v10.4.3, but contains significant modifications to both API and kernel behavior in order to support dual core SMP. This document describes the API and behavioral differences between Vanilla FreeRTOS and ESP-IDF FreeRTOS.

备注： This document assumes that the reader has a requisite understanding of Vanilla FreeRTOS (its features, behavior, and API usage). Refer to the Vanilla FreeRTOS documentation for more details.

备注： ESP-IDF FreeRTOS can be built for single core by enabling the *CONFIG_FREERTOS_UNICORE* configuration option. ESP targets that are single core will always have the *CONFIG_FREERTOS_UNICORE* option enabled. However, note that building with *CONFIG_FREERTOS_UNICORE* enabled does not equate to building with Vanilla FreeRTOS (i.e., some of the behavioral and API changes of ESP-IDF will still be present). For more details, see *ESP-IDF FreeRTOS Single Core* for more details.

This document is split into the following parts.

Contents

Symmetric Multiprocessing

Basic Concepts SMP (Symmetric Multiprocessing) is a computing architecture where two or more identical CPUs (cores) are connected to a single shared main memory and controlled by a single operating system. In general, an SMP system···

- has multiple cores running independently. Each core has its own register file, interrupts, and interrupt handling.
- presents an identical view of memory to each core. Thus a piece of code that accesses a particular memory address will have the same effect regardless of which core it runs on.

The main advantages of an SMP system compared to single core or Asymmetric Multiprocessing systems are that···

- the presence of multiple CPUs allows for multiple hardware threads, thus increases overall processing throughput.
- having symmetric memory means that threads can switch cores during execution. This in general can lead to better CPU utilization.

Although an SMP system allows threads to switch cores, there are scenarios where a thread must/should only run on a particular core. Therefore, threads in an SMP systems will also have a core affinity that specifies which particular core the thread is allowed to run on.

- A thread that is pinned to a particular core will only be able to run on that core
- A thread that is unpinned will be allowed to switch between cores during execution instead of being pinned to a particular core.

SMP on an ESP Target ESP targets (such as the ESP32, ESP32-S3) are dual core SMP SoCs. These targets have the following hardware features that make them SMP capable:

- Two identical cores known as CPU0 (i.e., Protocol CPU or PRO_CPU) and CPU1 (i.e., Application CPU or APP_CPU). This means that the execution of a piece of code is identical regardless of which core it runs on.
- Symmetric memory (with some small exceptions).
  - If multiple cores access the same memory address, their access will be serialized at the memory bus level.
  - True atomic access to the same memory address is achieved via an atomic compare-and-swap instruction provided by the ISA.
- Cross-core interrupts that allow one CPU to trigger and interrupt on another CPU. This allows cores to signal each other.

备注： The "PRO_CPU" and "APP_CPU" aliases for CPU0 and CPU1 exist in ESP-IDF as they reflect how typical IDF applications will utilize the two CPUs. Typically, the tasks responsible for handling wireless networking (e.g., WiFi or Bluetooth) will be pinned to CPU0 (thus the name PRO_CPU), whereas the tasks handling the remainder of the application will be pinned to CPU1 (thus the name APP_CPU).

Tasks

Creation Vanilla FreeRTOS provides the following functions to create a task:

- *`xTaskCreate()`* creates a task. The task's memory is dynamically allocated
- *`xTaskCreateStatic()`* creates a task. The task's memory is statically allocated (i.e., provided by the user)

However, in an SMP system, tasks need to be assigned a particular affinity. Therefore, ESP-IDF provides a `PinnedToCore` version of Vanilla FreeRTOS's task creation functions:

- *`xTaskCreatePinnedToCore()`* creates a task with a particular core affinity. The task's memory is dynamically allocated.
- *`xTaskCreateStaticPinnedToCore()`* creates a task with a particular core affinity. The task's memory is statically allocated (i.e., provided by the user)

The `PinnedToCore` versions of the task creation functions API differ from their vanilla counter parts by having an extra `xCoreID` parameter that is used to specify the created task's core affinity. The valid values for core affinity are:

- `0` which pins the created task to CPU0
- `1` which pins the created task to CPU1
- `tskNO_AFFINITY` which allows the task to be run on both CPUs

Note that ESP-IDF FreeRTOS still supports the vanilla versions of the task creation functions. However, they have been modified to simply call their `PinnedToCore` counterparts with `tskNO_AFFINITY`.

备注： ESP-IDF FreeRTOS also changes the units of `ulStackDepth` in the task creation functions. Task stack sizes in Vanilla FreeRTOS are specified in number of words, whereas in ESP-IDF FreeRTOS, the task stack sizes are specified in bytes.

Execution The anatomy of a task in ESP-IDF FreeRTOS is the same as Vanilla FreeRTOS. More specifically, ESP-IDF FreeRTOS tasks:

- Can only be in one of following states: Running, Ready, Blocked, or Suspended.
- Task functions are typically implemented as an infinite loop
- Task functions should never return

Deletion Task deletion in Vanilla FreeRTOS is called via *`vTaskDelete()`*. The function allows deletion of another task or the currently running task (if the provided task handle is `NULL`). The actual freeing of the task's memory is sometimes delegated to the idle task (if the task being deleted is the currently running task).

ESP-IDF FreeRTOS provides the same *`vTaskDelete()`* function. However, due to the dual core nature, there are some behavioral differences when calling *`vTaskDelete()`* in ESP-IDF FreeRTOS:

- When deleting a task that is pinned to the other core, that task's memory is always freed by the idle task of the other core (due to the need to clear FPU registers).
- When deleting a task that is currently running on the other core, a yield is triggered on the other core and the task's memory is freed by one of the idle tasks (depending on the task's core affinity)
- A deleted task's memory is freed immediately if…
  - The tasks is currently running on this core and is also pinned to this core
  - The task is not currently running and is not pinned to any core

Users should avoid calling *`vTaskDelete()`* on a task that is currently running on the other core. This is due to the fact that it is difficult to know what the task currently running on the other core is executing, thus can lead to unpredictable behavior such as…

- Deleting a task that is holding a mutex
- Deleting a task that has yet to free memory it previously allocated

Where possible, users should design their application such that *`vTaskDelete()`* is only ever called on tasks in a known state. For example:

- Tasks self deleting (via `vTaskDelete(NULL)`) when their execution is complete and have also cleaned up all resources used within the task.
- Tasks placing themselves in the suspend state (via *`vTaskSuspend()`*) before being deleted by another task.

SMP Scheduler

The Vanilla FreeRTOS scheduler is best described as a **Fixed Priority Preemptive scheduler with Time Slicing** meaning that:

- Each tasks is given a constant priority upon creation. The scheduler executes highest priority ready state task
- The scheduler can switch execution to another task without the cooperation of the currently running task

- The scheduler will periodically switch execution between ready state tasks of the same priority (in a round robin fashion). Time slicing is governed by a tick interrupt.

The ESP-IDF FreeRTOS scheduler supports the same scheduling features (i.e., Fixed Priority, Preemption, and Time Slicing) albeit with some small behavioral differences.

Fixed Priority In Vanilla FreeRTOS, when scheduler selects a new task to run, it will always select the current highest priority ready state task. In ESP-IDF FreeRTOS, each core will independently schedule tasks to run. When a particular core selects a task, the core will select the highest priority ready state task that can be run by the core. A task can be run by the core if:

- The task has a compatible affinity (i.e., is either pinned to that core or is unpinned)
- The task is not currently being run by another core

However, users should not assume that the two highest priority ready state tasks are always run by the scheduler as a task's core affinity must also be accounted for. For example, given the following tasks:

- Task A of priority 10 pinned to CPU0
- Task B of priority 9 pinned to CPU0
- Task C of priority 8 pinned to CPU1

The resulting schedule will have Task A running on CPU0 and Task C running on CPU1. Task B is not run even though it is the second highest priority task.

Preemption In Vanilla FreeRTOS, the scheduler can preempt the currently running task if a higher priority task becomes ready to execute. Likewise in ESP-IDF FreeRTOS, each core can be individually preempted by the scheduler if the scheduler determines that a higher priority task can run on that core.

However, there are some instances where a higher priority task that becomes ready can be run on multiple cores. In this case, the scheduler will only preempt one core. The scheduler always gives preference to the current core when multiple cores can be preempted. In other words, if the higher priority ready task is unpinned and has a higher priority than the current priority of both cores, the scheduler will always choose to preempt the current core. For example, given the following tasks:

- Task A of priority 8 currently running on CPU0
- Task B of priority 9 currently running on CPU1
- Task C of priority 10 that is unpinned and was unblocked by Task B

The resulting schedule will have Task A running on CPU0 and Task C preempting Task B given that the scheduler always gives preference to the current core.

Time Slicing The Vanilla FreeRTOS scheduler implements time slicing meaning that if current highest ready priority contains multiple ready tasks, the scheduler will switch between those tasks periodically in a round robin fashion.

However, in ESP-IDF FreeRTOS, it is not possible to implement perfect Round Robin time slicing due to the fact that a particular task may not be able to run on a particular core due to the following reasons:

- The task is pinned to the another core.
- For unpinned tasks, the task is already being run by another core.

Therefore, when a core searches the ready state task list for a task to run, the core may need to skip over a few tasks in the same priority list or drop to a lower priority in order to find a ready state task that the core can run.

The ESP-IDF FreeRTOS scheduler implements a Best Effort Round Robin time slicing for ready state tasks of the same priority by ensuring that tasks that have been selected to run will be placed at the back of the list, thus giving unselected tasks a higher priority on the next scheduling iteration (i.e., the next tick interrupt or yield)

The following example demonstrates the Best Effort Round Robin time slicing in action. Assume that:

- There are four ready state tasks of the same priority `AX, B0, C1, D1` where: - The priority is the current highest priority with ready state tasks - The first character represents the task's names (i.e., `A, B, C, D`) - And the second character represents the tasks core pinning (and `X` means unpinned)

- The task list is always searched from the head

```
------------------------------------------------------------------------------

1. Starting state. None of the ready state tasks have been selected to run

Head [ AX , B0 , C1 , D0 ] Tail

------------------------------------------------------------------------------

2. Core 0 has tick interrupt and searches for a task to run.
  Task A is selected and is moved to the back of the list

Core0--|
Head [ AX , B0 , C1 , D0 ] Tail

                       0
Head [ B0 , C1 , D0 , AX ] Tail

------------------------------------------------------------------------------

3. Core 1 has a tick interrupt and searches for a task to run.
  Task B cannot be run due to incompatible affinity, so core 1 skips to Task C.
  Task C is selected and is moved to the back of the list

Core1-------|          0
Head [ B0 , C1 , D0 , AX ] Tail

                  0    1
Head [ B0 , D0 , AX , C1 ] Tail

------------------------------------------------------------------------------

4. Core 0 has another tick interrupt and searches for a task to run.
  Task B is selected and moved to the back of the list


Core0--|              1
Head [ B0 , D0 , AX , C1 ] Tail

                  1    0
Head [ D0 , AX , C1 , B0 ] Tail

------------------------------------------------------------------------------

5. Core 1 has another tick and searches for a task to run.
  Task D cannot be run due to incompatible affinity, so core 1 skips to Task A
  Task A is selected and moved to the back of the list

Core1-------|          0
Head [ D0 , AX , C1 , B0 ] Tail

                  0    1
Head [ D0 , C1 , B0 , AX ] Tail
```

The implications to users regarding the Best Effort Round Robin time slicing:

- Users cannot expect multiple ready state tasks of the same priority to run sequentially (as is the case in Vanilla FreeRTOS). As demonstrated in the example above, a core may need to skip over tasks.
- However, given enough ticks, a task will eventually be given some processing time.
- If a core cannot find a task runnable task at the highest ready state priority, it will drop to a lower priority to search for tasks.
- To achieve ideal round robin time slicing, users should ensure that all tasks of a particular priority are pinned

to the same core.

Tick Interrupts Vanilla FreeRTOS requires that a periodic tick interrupt occurs. The tick interrupt is responsible for:

- Incrementing the scheduler's tick count
- Unblocking any blocked tasks that have timed out
- Checking if time slicing is required (i.e., triggering a context switch)
- Executing the application tick hook

In ESP-IDF FreeRTOS, each core will receive a periodic interrupt and independently run the tick interrupt. The tick interrupts on each core are of the same period but can be out of phase. However, the tick responsibilities listed above are not run by all cores:

- CPU0 will execute all of the tick interrupt responsibilities listed above
- CPU1 will only check for time slicing and execute the application tick hook

---

备注： CPU0 is solely responsible for keeping time in ESP-IDF FreeRTOS. Therefore anything that prevents CPU0 from incrementing the tick count (such as suspending the scheduler on CPU0) will cause the entire schedulers time keeping to lag behind.

Idle Tasks Vanilla FreeRTOS will implicitly create an idle task of priority 0 when the scheduler is started. The idle task runs when no other task is ready to run, and it has the following responsibilities:

- Freeing the memory of deleted tasks
- Executing the application idle hook

In ESP-IDF FreeRTOS, a separate pinned idle task is created for each core. The idle tasks on each core have the same responsibilities as their vanilla counterparts.

Scheduler Suspension Vanilla FreeRTOS allows the scheduler to be suspended/resumed by calling `vTaskSuspendAll()` and `xTaskResumeAll()` respectively. While the scheduler is suspended:

- Task switching is disabled but interrupts are left enabled.
- Calling any blocking/yielding function is forbidden, and time slicing is disabled.
- The tick count is frozen (but the tick interrupt will still occur to execute the application tick hook)

On scheduler resumption, `xTaskResumeAll()` will catch up all of the lost ticks and unblock any timed out tasks.

In ESP-IDF FreeRTOS, suspending the scheduler across multiple cores is not possible. Therefore when `vTaskSuspendAll()` is called on a particular core (e.g., core A):

- Task switching is disabled only on core A but interrupts for core A are left enabled
- Calling any blocking/yielding function on core A is forbidden. Time slicing is disabled on core A.
- If an interrupt on core A unblocks any tasks, tasks with affinity to core A will go into core A's own pending ready task list. Unpinned tasks or tasks with affinity to other cores can be scheduled on cores with the scheduler running.
- In case the scheduler is suspended on all cores, tasks unblocked by an interrupt will go to the pending ready task lists of their pinned cores or to the pending ready list of the core on which the interrupt is called if the tasks are unpinned.
- If core A is CPU0, the tick count is frozen and a pended tick count is incremented instead. However, the tick interrupt will still occur in order to execute the application tick hook.

When `xTaskResumeAll()` is called on a particular core (e.g., core A):

- Any tasks added to core A's pending ready task list will be resumed
- If core A is CPU0, the pended tick count is unwound to catch up the lost ticks.

警告： Given that scheduler suspension on ESP-IDF FreeRTOS will only suspend scheduling on a particular core, scheduler suspension is **NOT** a valid method ensuring mutual exclusion between tasks when accessing shared data. Users should use proper locking primitives such as mutexes or spinlocks if they require mutual exclusion.

Disabling Interrupts Vanilla FreeRTOS allows interrupts to be disabled and enabled by calling *taskDISABLE_INTERRUPTS* and *taskENABLE_INTERRUPTS* respectively.

ESP-IDF FreeRTOS provides the same API, however interrupts will only disabled or enabled on the current core.

警告： Disabling interrupts is a valid method of achieve mutual exclusion in Vanilla FreeRTOS (and single core systems in general). However, in an SMP system, disabling interrupts is **NOT** a valid method ensuring mutual exclusion. Refer to Critical Sections for more details.

Critical Sections

API Changes Vanilla FreeRTOS implements critical sections by disabling interrupts, This prevents preemptive context switches and the servicing of ISRs during a critical section. Thus a task/ISR that enters a critical section is guaranteed to be the sole entity to access a shared resource. Critical sections in Vanilla FreeRTOS have the following API:

- `taskENTER_CRITICAL()` enters a critical section by disabling interrupts
- `taskEXIT_CRITICAL()` exits a critical section by reenabling interrupts
- `taskENTER_CRITICAL_FROM_ISR()` enters a critical section from an ISR by disabling interrupt nesting
- `taskEXIT_CRITICAL_FROM_ISR()` exits a critical section from an ISR by reenabling interrupt nesting

However, in an SMP system, merely disabling interrupts does not constitute a critical section as the presence of other cores means that a shared resource can still be concurrently accessed. Therefore, critical sections in ESP-IDF FreeRTOS are implemented using spinlocks. To accommodate the spinlocks, the ESP-IDF FreeRTOS critical section APIs contain an additional spinlock parameter as shown below:

- Spinlocks are of `portMUX_TYPE` (**not to be confused to FreeRTOS mutexes**)
- `taskENTER_CRITICAL(&spinlock)` enters a critical from a task context
- `taskEXIT_CRITICAL(&spinlock)` exits a critical section from a task context
- `taskENTER_CRITICAL_ISR(&spinlock)` enters a critical section from an interrupt context
- `taskEXIT_CRITICAL_ISR(&spinlock)` exits a critical section from an interrupt context

备注： The critical section API can be called recursively (i.e., nested critical sections). Entering a critical section multiple times recursively is valid so long as the critical section is exited the same number of times it was entered. However, given that critical sections can target different spinlocks, users should take care to avoid dead locking when entering critical sections recursively.

Spinlocks can be allocated statically or dynamically. As such, macros are provided for both static and dynamic initialization of spinlocks, as demonstrated by the following code snippets.

- Allocating a static spinlock and initializing it using `portMUX_INITIALIZER_UNLOCKED`

```
// Statically allocate and initialize the spinlock
static portMUX_TYPE my_spinlock = portMUX_INITIALIZER_UNLOCKED;

void some_function(void)
{
    taskENTER_CRITICAL(&my_spinlock);
    // We are now in a critical section
    taskEXIT_CRITICAL(&my_spinlock);
}
```

- Allocating a dynamic spinlock and initializing it using `portMUX_INITIALIZE()`

```
// Allocate the spinlock dynamically
portMUX_TYPE *my_spinlock = malloc(sizeof(portMUX_TYPE));
// Initialize the spinlock dynamically
portMUX_INITIALIZE(my_spinlock);

...

taskENTER_CRITICAL(my_spinlock);
// Access the resource
taskEXIT_CRITICAL(my_spinlock);
```

Implementation In ESP-IDF FreeRTOS, the process of a particular core entering and exiting a critical section is as follows:

- For `taskENTER_CRITICAL(&spinlock)` (or `taskENTER_CRITICAL_ISR(&spinlock)`)
  1. The core disables its interrupts (or interrupt nesting) up to `configMAX_SYSCALL_INTERRUPT_PRIORITY`
  2. The core then spins on the spinlock using an atomic compare-and-set instruction until it acquires the lock. A lock is acquired when the core is able to set the lock's owner value to the core's ID.
  3. Once the spinlock is acquired, the function returns. The remainder of the critical section runs with interrupts (or interrupt nesting) disabled.
- For `taskEXIT_CRITICAL(&spinlock)` (or `taskEXIT_CRITICAL_ISR(&spinlock)`)
  1. The core releases the spinlock by clearing the spinlock's owner value
  2. The core re-enables interrupts (or interrupt nesting)

Restrictions and Considerations Given that interrupts (or interrupt nesting) are disabled during a critical section, there are multiple restrictions regarding what can be done within a critical sections. During a critical section, users should keep the following restrictions and considerations in mind:

- Critical sections should be as kept as short as possible
  - The longer the critical section lasts, the longer a pending interrupt can be delayed.
  - A typical critical section should only access a few data structures and/or hardware registers
  - If possible, defer as much processing and/or event handling to the outside of critical sections.
- FreeRTOS API should not be called from within a critical section
- Users should never call any blocking or yielding functions within a critical section

Misc

Floating Point Usage Usually, when a context switch occurs:

- the current state of a CPU's registers are saved to the stack of task being switch out
- the previously saved state of the CPU's registers are loaded from the stack of the task being switched in

However, ESP-IDF FreeRTOS implements Lazy Context Switching for the FPU (Floating Point Unit) registers of a CPU. In other words, when a context switch occurs on a particular core (e.g., CPU0), the state of the core's FPU registers are not immediately saved to the stack of the task getting switched out (e.g., Task A). The FPU's registers are left untouched until:

- A different task (e.g., Task B) runs on the same core and uses the FPU. This will trigger an exception that will save the FPU registers to Task A's stack.
- Task A get's scheduled to the same core and continues execution. Saving and restoring the FPU's registers is not necessary in this case.

However, given that tasks can be unpinned thus can be scheduled on different cores (e.g., Task A switches to CPU1), it is unfeasible to copy and restore the FPU's registers across cores. Therefore, when a task utilizes the FPU (by using a `float` type in its call flow), ESP-IDF FreeRTOS will automatically pin the task to the current core it is running on. This ensures that all tasks that uses the FPU are always pinned to a particular core.

Furthermore, ESP-IDF FreeRTOS by default does not support the usage of the FPU within an interrupt context given that the FPU's register state is tied to a particular task.

备注: ESP targets that contain an FPU do not support hardware acceleration for double precision floating point arithmetic (`double`). Instead `double` is implemented via software hence the behavioral restrictions regarding the `float` type do not apply to `double`. Note that due to the lack of hardware acceleration, `double` operations may consume significantly more CPU time in comparison to `float`.

ESP-IDF FreeRTOS Single Core Although ESP-IDF FreeRTOS is an SMP scheduler, some ESP targets are single core (such as the ESP32-S2 and ESP32-C3). When building ESP-IDF applications for these targets, ESP-IDF FreeRTOS is still used but the number of cores will be set to *1* (i.e., the *CONFIG_FREERTOS_UNICORE* will always be enabled for single core targets).

For multicore targets (such as the ESP32 and ESP32-S3), *CONFIG_FREERTOS_UNICORE* can also be set. This will result in ESP-IDF FreeRTOS only running on CPU0, and all other cores will be inactive.

备注: Users should bear in mind that enabling *CONFIG_FREERTOS_UNICORE* **is NOT equivalent to running Vanilla FreeRTOS**. The additional API of ESP-IDF FreeRTOS can still be called, and the behavior changes of ESP-IDF FreeRTOS will incur a small amount of overhead even when compiled for only a single core.

API Reference

This section contains documentation of FreeRTOS types, functions, and macros. It is automatically generated from FreeRTOS header files.

Task API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/task.h

Functions

BaseType_t **xTaskCreatePinnedToCore**(TaskFunction_t pxTaskCode, const char *const pcName, const configSTACK_DEPTH_TYPE usStackDepth, void *const pvParameters, UBaseType_t uxPriority, *TaskHandle_t* *const pvCreatedTask, const BaseType_t xCoreID)

Create a new task with a specified affinity and add it to the list of tasks that are ready to run.

This function is similar to xTaskCreate, but allows setting task affinity in SMP system.

Example usage:

```
// Task to be created.
void vTaskCode( void * pvParameters )
{
  for( ;; )
  {
      // Task code goes here.
  }
}

// Function that creates a task.
```

(下页继续)

(续上页)

```
void vOtherFunction( void )
{
static uint8_t ucParameterToPass;
TaskHandle_t xHandle = NULL;

  // Create the task pinned to core 0, 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.
  xTaskCreatePinnedToCore( vTaskCode, "NAME", STACK_SIZE, &ucParameterToPass,␣
↪tskIDLE_PRIORITY, &xHandle, 0 );
  configASSERT( xHandle );

  // Use the handle to delete the task.
  if( xHandle != NULL )
  {
     vTaskDelete( xHandle );
  }
}
```

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), or should be terminated using vTaskDelete function.
- **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).
- **pvCreatedTask** –**[out]** Used to pass back a handle by which the created task can be referenced.
- **xCoreID** –If the value is tskNO_AFFINITY, the created task is not pinned to any CPU, and the scheduler can run it on any core available. Values 0 or 1 indicate the index number of the CPU which the task should be pinned to. Specifying values larger than (configNUM_CORES - 1) will cause the function to fail.

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 pxStackBuffer, StaticTask_t *const pxTaskBuffer, const BaseType_t xCoreID)

Create a new task with a specified affinity and add it to the list of tasks that are ready to run.

This function is similar to xTaskCreateStatic, but allows specifying task affinity in an SMP system.

Example usage:

```
// Dimensions the buffer that the task being created will use as its stack.
// NOTE:  This is the number of words the stack will hold, not the number of
// bytes.  For example, if each stack item is 32-bits, and this is set to 100,
// then 400 bytes (100 * 32-bits) will be allocated.
#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 ];

// 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 xTaskCreateStaticPinnedToCore().
    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 pinned to core 0 without using any dynamic memory␣
↪allocation.
    xHandle = xTaskCreateStaticPinnedToCore(
                  vTaskCode,       // Function that implements the task.
                  "NAME",          // Text name for the task.
                  STACK_SIZE,      // Stack size in bytes, not words.
                  ( 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.
                  0 );             // Specify the task's core affinity

    // 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 );
}
```

参数

- **pxTaskCode** –Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop), or should be terminated using vTaskDelete function.
- **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.
- **pxStackBuffer** –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.
- **xCoreID** –If the value is tskNO_AFFINITY, the created task is not pinned to any CPU, and the scheduler can run it on any core available. Values 0 or 1 indicate the index number of the CPU which the task should be pinned to. Specifying values larger than (configNUM_CORES - 1) will cause the function to fail.

返回 If neither pxStackBuffer or pxTaskBuffer are NULL, then the task will be created and pdPASS is returned. If either pxStackBuffer or pxTaskBuffer are NULL then the task will not be created and errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY is returned.

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 );
```

(下页继续)

(续上页)

```
  configASSERT( xHandle );

  // Use the handle to delete the task.
  if( xHandle != NULL )
  {
     vTaskDelete( xHandle );
  }
}
```

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), or should be terminated using vTaskDelete function.
- **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.
```

(下页继续)

(续上页)

```
StackType_t xStack[ STACK_SIZE ];

// 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, not words.
                  ( 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 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), or should be terminated using vTaskDelete function.
- **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 pxStackBuffer or pxTaskBuffer are NULL, then the task will be created and pdPASS is returned. If either pxStackBuffer or pxTaskBuffer are NULL then the task will

not be created and errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY is returned.

void **vTaskAllocateMPURegions**(*TaskHandle_t* xTask, const MemoryRegion_t *const pxRegions)

Only available when configSUPPORT_DYNAMIC_ALLOCATION is set to 1.

xTaskCreateRestricted() should only be used in systems that include an MPU implementation.

Create a new task and add it to the list of tasks that are ready to run. The function parameters define the memory regions and associated access permissions allocated to the task.

See xTaskCreateRestrictedStatic() for a version that does not use any dynamic memory allocation.

Example usage:

```
// Create an TaskParameters_t structure that defines the task to be created.
static const TaskParameters_t xCheckTaskParameters =
{
 vATask,     // pvTaskCode - the function that implements the task.
 "ATask",    // pcName - just a text name for the task to assist debugging.
 100,        // usStackDepth - the stack size DEFINED IN WORDS.
 NULL,       // pvParameters - passed into the task function as the function␣
↪parameters.
 ( 1UL | portPRIVILEGE_BIT ),// uxPriority - task priority, set the␣
↪portPRIVILEGE_BIT if the task should run in a privileged state.
 cStackBuffer,// puxStackBuffer - the buffer to be used as the task stack.

 // xRegions - Allocate up to three separate memory regions for access by
 // the task, with appropriate access permissions.  Different processors have
 // different memory alignment requirements - refer to the FreeRTOS␣
↪documentation
 // for full information.
 {
     // Base address                 Length  Parameters
     { cReadWriteArray,              32,     portMPU_REGION_READ_WRITE },
     { cReadOnlyArray,               32,     portMPU_REGION_READ_ONLY },
     { cPrivilegedOnlyAccessArray,   128,    portMPU_REGION_PRIVILEGED_READ_
↪WRITE }
 }
};

int main( void )
{
TaskHandle_t xHandle;

 // Create a task from the const structure defined above.  The task handle
 // is requested (the second parameter is not NULL) but in this case just for
 // demonstration purposes as its not actually used.
 xTaskCreateRestricted( &xRegTest1Parameters, &xHandle );

 // Start the scheduler.
 vTaskStartScheduler();

 // Will only get here if there was insufficient memory to create the idle
 // and/or timer task.
 for( ;; );
}
```

Only available when configSUPPORT_STATIC_ALLOCATION is set to 1.

xTaskCreateRestrictedStatic() should only be used in systems that include an MPU implementation.

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

xTaskCreateRestricted() then the stack is provided by the application writer, and the memory used to hold the task's data structure is automatically dynamically allocated inside the xTaskCreateRestricted() function. If a task is created using xTaskCreateRestrictedStatic() then the application writer must provide the memory used to hold the task's data structures too. xTaskCreateRestrictedStatic() therefore allows a memory protected task to be created without using any dynamic memory allocation.

Example usage:

```
// Create an TaskParameters_t structure that defines the task to be created.
// The StaticTask_t variable is only included in the structure when
// configSUPPORT_STATIC_ALLOCATION is set to 1.  The PRIVILEGED_DATA macro can
// be used to force the variable into the RTOS kernel's privileged data area.
static PRIVILEGED_DATA StaticTask_t xTaskBuffer;
static const TaskParameters_t xCheckTaskParameters =
{
 vATask,      // pvTaskCode - the function that implements the task.
 "ATask",     // pcName - just a text name for the task to assist debugging.
 100,         // usStackDepth - the stack size DEFINED IN BYTES.
 NULL,        // pvParameters - passed into the task function as the function
↪parameters.
 ( 1UL | portPRIVILEGE_BIT ),// uxPriority - task priority, set the
↪portPRIVILEGE_BIT if the task should run in a privileged state.
 cStackBuffer,// puxStackBuffer - the buffer to be used as the task stack.

 // xRegions - Allocate up to three separate memory regions for access by
 // the task, with appropriate access permissions.  Different processors have
 // different memory alignment requirements - refer to the FreeRTOS
↪documentation
 // for full information.
 {
     // Base address                 Length  Parameters
     { cReadWriteArray,              32,     portMPU_REGION_READ_WRITE },
     { cReadOnlyArray,               32,     portMPU_REGION_READ_ONLY },
     { cPrivilegedOnlyAccessArray,   128,    portMPU_REGION_PRIVILEGED_READ_
↪WRITE }
 }

 &xTaskBuffer; // Holds the task's data structure.
};

int main( void )
{
TaskHandle_t xHandle;

 // Create a task from the const structure defined above.  The task handle
 // is requested (the second parameter is not NULL) but in this case just for
 // demonstration purposes as its not actually used.
 xTaskCreateRestricted( &xRegTest1Parameters, &xHandle );

 // Start the scheduler.
 vTaskStartScheduler();

 // Will only get here if there was insufficient memory to create the idle
 // and/or timer task.
 for( ;; );
}
```

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

- **pxTaskDefinition** –Pointer to a structure that contains a member for each of the normal xTaskCreate() parameters (see the xTaskCreate() API documentation) plus an optional stack buffer and the memory region definitions.
- **pxCreatedTask** –Used to pass back a handle by which the created task can be referenced.
- **pxTaskDefinition** –Pointer to a structure that contains a member for each of the normal xTaskCreate() parameters (see the xTaskCreate() API documentation) plus an optional stack buffer and the memory region definitions. If configSUPPORT_STATIC_ALLOCATION is set to 1 the structure contains an additional member, which is used to point to a variable of type StaticTask_t - which is then used to hold the task's data structure.
- **pxCreatedTask** –Used to pass back a handle by which the created task can be referenced.
- **xTask** –The handle of the task being updated.
- **pxRegions** –A pointer to an MemoryRegion_t structure that contains the new memory region definitions.

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

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 **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 effect 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␣
↪);
```

(下页继续)

(续上页)

```
  // ...

  // Use the handle to raise the priority of the created task.
  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 **vTaskStartScheduler**(void)

Starts the real time kernel tick processing. After calling the kernel has control over which tasks are executed and when.

See the demo application file main.c for an example of creating tasks and starting the kernel.

Example usage:

```
void vAFunction( void )
{
  // Create at least one task before starting the kernel.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );

  // Start the real time kernel with preemption.
  vTaskStartScheduler ();

  // Will not get here unless a task calls vTaskEndScheduler ()
}
```

备注： : In ESP-IDF the scheduler is started automatically during application startup, vTaskStartScheduler() should not be called from ESP-IDF applications.

---

void **vTaskEndScheduler** (void)

NOTE: At the time of writing only the x86 real mode port, which runs on a PC in place of DOS, implements this function.

Stops the real time kernel tick. All created tasks will be automatically deleted and multitasking (either preemptive or cooperative) will stop. Execution then resumes from the point where vTaskStartScheduler () was called, as if vTaskStartScheduler () had just returned.

See the demo application file main. c in the demo/PC directory for an example that uses vTaskEndScheduler ().

vTaskEndScheduler () requires an exit function to be defined within the portable layer (see vPortEndScheduler () in port. c for the PC port). This performs hardware specific operations such as stopping the kernel tick.

vTaskEndScheduler () will cause all of the resources allocated by the kernel to be freed - but will not free resources allocated by application tasks.

Example usage:

```
void vTaskCode( void * pvParameters )
{
  for( ;; )
  {
      // Task code goes here.

      // At some point we want to end the real time kernel processing
      // so call ...
      vTaskEndScheduler ();
  }
}

void vAFunction( void )
{
  // Create at least one task before starting the kernel.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );

  // Start the real time kernel with preemption.
  vTaskStartScheduler ();

  // Will only get here when the vTaskCode () task has called
  // vTaskEndScheduler ().  When we get here we are back to single task
  // execution.
}
```

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, vTaskDelayUntil(), 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.
```

(下页继续)

(续上页)

```
        // 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.

        // ...

        // 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)

UBaseType_t **uxTaskGetStackHighWaterMark**(*TaskHandle_t* xTask)

Returns the high water mark of the stack associated with 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 bytes not words, unlike vanilla FreeRTOS) 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 bytes not words, unlike vanilla FreeRTOS) since the task referenced by xTask was created.

configSTACK_DEPTH_TYPE **uxTaskGetStackHighWaterMark2**(*TaskHandle_t* xTask)

Returns the start of the stack associated with 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 bytes not words, unlike vanilla FreeRTOS) 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 bytes not words, unlike vanilla FreeRTOS) since the task referenced by xTask was created.

uint8_t *__pxTaskGetStackStart__(*TaskHandle_t* xTask)

Returns the start of the stack associated with xTask.

INCLUDE_pxTaskGetStackStart must be set to 1 in FreeRTOSConfig.h for this function to be available.

Returns the lowest stack memory address, regardless of whether the stack grows up or down.

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 **vTaskSetApplicationTaskTag**(*TaskHandle_t* xTask, *TaskHookFunction_t* pxHookFunction)

Sets pxHookFunction to be the task hook function used by the task xTask.

参数

- **xTask** –Handle of the task to set the hook function for Passing xTask as NULL has the effect of setting the calling tasks hook function.
- **pxHookFunction** –Pointer to the 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)

Set local storage pointer specific to the given task.

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.

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.

void ***pvTaskGetThreadLocalStoragePointer**(*TaskHandle_t* xTaskToQuery, BaseType_t xIndex)

Get local storage pointer specific to the given task.

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.

参数

- **xTaskToQuery** –Task to get thread local storage pointer for
- **xIndex** –The index of the pointer to get, from 0 to configNUM_THREAD_LOCAL_STORAGE_POINTERS - 1.

返回 Pointer value

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.

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 thie 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.

参数

- **xTask** –Handle of the task to call the hook for.
- **pvParameter** –Parameter 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. It is not valid to call xTaskGetIdleTaskHandle() before the scheduler has been started.

UBaseType_t **uxTaskGetSystemState**(TaskStatus_t *const pxTaskStatusArray, const UBaseType_t uxArraySize, uint32_t *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.

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;
uint32_t 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;

        // 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++ )
            {
```

(下页继续)

(续上页)

```
                // 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 );
    }
}
```

备注： This function is intended for debugging use only as its use results in the scheduler remaining suspended for an extended period.

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)

List all the current tasks.

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 and stack usage.

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)

Get the state of running tasks as a string

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.

uint32_t **`ulTaskGetIdleRunTimeCounter`**(void)

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.

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.

返回 The total run time of 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.

BaseType_t **xTaskGenericNotify**(*TaskHandle_t* xTaskToNotify, UBaseType_t uxIndexToNotify, uint32_t ulValue, *eNotifyAction* eAction, uint32_t *pulPreviousNotificationValue)

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() 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. 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.

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:
- **pulPreviousNotificationValue** –- Can be used to pass out the subject task's notification value before any bits are modified by the notify function.

返回 Dependent on the value of eAction. See the description of the eAction parameter.

BaseType_t **xTaskGenericNotifyFromISR**(*TaskHandle_t* xTaskToNotify, UBaseType_t uxIndexToNotify, uint32_t ulValue, *eNotifyAction* eAction, uint32_t *pulPreviousNotificationValue, BaseType_t *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:
- **pulPreviousNotificationValue** –- Can be used to pass out the subject task's notification value before any bits are modified by the notify function.
- **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.

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 is marked as waiting for a new notification (provided a notification is not already pending). 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 ulTaskNotificationTakeIndexed() 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.

uint32_t **`ulTaskGenericNotifyTake`**(UBaseType_t uxIndexToWaitOn, BaseType_t xClearCountOnExit, TickType_t 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 the task's notification value to be non-zero. 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).

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)

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
// 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.
```

(下页继续)

(续上页)

```
    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)

Macros

tskKERNEL_VERSION_NUMBER

tskKERNEL_VERSION_MAJOR

tskKERNEL_VERSION_MINOR

tskKERNEL_VERSION_BUILD

tskMPU_REGION_READ_ONLY

tskMPU_REGION_READ_WRITE

tskMPU_REGION_EXECUTE_NEVER

tskMPU_REGION_NORMAL_MEMORY

tskMPU_REGION_DEVICE_MEMORY

tskDEFAULT_INDEX_TO_NOTIFY

tskNO_AFFINITY

tskIDLE_PRIORITY

Defines the priority used by the idle task. This must not be modified.

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.

> **备注:** 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.

> **备注:** 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

taskSCHEDULER_NOT_STARTED

taskSCHEDULER_RUNNING

vTaskDelayUntil(pxPreviousWakeTime, xTimeIncrement)

xTaskNotify(xTaskToNotify, ulValue, eAction)

xTaskNotifyIndexed(xTaskToNotify, uxIndexToNotify, ulValue, eAction)

xTaskNotifyAndQuery(xTaskToNotify, 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.

xTaskNotifyAndQueryIndexed(xTaskToNotify, uxIndexToNotify, ulValue, eAction, pulPreviousNotifyValue)

xTaskNotifyFromISR(xTaskToNotify, ulValue, eAction, pxHigherPriorityTaskWoken)

xTaskNotifyIndexedFromISR(xTaskToNotify, uxIndexToNotify, ulValue, eAction, pxHigherPriorityTaskWoken)

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.

xTaskNotifyAndQueryFromISR(xTaskToNotify, ulValue, eAction, pulPreviousNotificationValue, pxHigherPriorityTaskWoken)

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 ulTaskNotificationTakeIndexed() 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.

xTaskNotifyGive(xTaskToNotify)

vTaskNotifyGiveFromISR(xTaskToNotify, pxHigherPriorityTaskWoken)

vTaskNotifyGiveIndexedFromISR(xTaskToNotify, uxIndexToNotify, pxHigherPriorityTaskWoken)

ulTaskNotifyTake(xClearCountOnExit, xTicksToWait)

ulTaskNotifyTakeIndexed(uxIndexToWaitOn, xClearCountOnExit, xTicksToWait)

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*)

typedef void (***TlsDeleteCallbackFunction_t**)(int, void*)

Prototype of local storage pointer deletion callback.

Enumerations

enum **eTaskState**

Task states returned by eTaskGetState.

Values:

enumerator **eRunning**

enumerator **eReady**

enumerator **eBlocked**

enumerator **eSuspended**

enumerator **eDeleted**

enumerator **eInvalid**

enum **eNotifyAction**

Values:

enumerator **eNoAction**

enumerator **eSetBits**

enumerator **eIncrement**

enumerator **eSetValueWithOverwrite**

enumerator **eSetValueWithoutOverwrite**

enum **eSleepModeStatus**

Possible return values for eTaskConfirmSleepModeStatus().

Values:

enumerator **eAbortSleep**

enumerator **eStandardSleep**

enumerator **eNoTasksWaitingTimeout**

Queue API

Header File

- components/freertos/FreeRTOS-Kernel/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;
```

(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( 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;

// 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 )
```

(下页继续)

(续上页)

```
{
    // 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.
    }
}

// ... 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;
```

(下页继续)

(续上页)

```
// 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 );

    // 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 )
{
    taskYIELD_YIELD_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** –**[out]** 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 ) );
```

(下页继续)

(续上页)

```
 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.
 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 cTaskWokenByReceive will have been set to
     // pdTRUE.  No matter how many times this loop iterates only one
     // task will be woken.
 }

 if( cTaskWokenByPost != ( 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** –**[out]** 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)

BaseType_t **xQueueIsQueueFullFromISR**(const *QueueHandle_t* xQueue)

UBaseType_t **uxQueueMessagesWaitingFromISR**(const *QueueHandle_t* xQueue)

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 effect the number of queues, semaphores and mutexes that can be created - just the number that the registry can hold.

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.

QueueHandle_t **xQueueGenericCreate**(const UBaseType_t uxQueueLength, const UBaseType_t uxItemSize, const uint8_t ucQueueType)

Generic version of the function used to create a queue using dynamic memory allocation. This is called by other functions and macros that create other RTOS objects that use the queue structure as their base.

QueueHandle_t **xQueueGenericCreateStatic**(const UBaseType_t uxQueueLength, const UBaseType_t uxItemSize, uint8_t *pucQueueStorage, StaticQueue_t *pxStaticQueue, const uint8_t ucQueueType)

Generic version of the function used to create a queue using dynamic memory allocation. This is called by other functions and macros that create other RTOS objects that use the queue structure as their base.

BaseType_t **xQueueGenericGetStaticBuffers**(*QueueHandle_t* xQueue, uint8_t **ppucQueueStorage, StaticQueue_t **ppxStaticQueue)

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.
```

(下页继续)

(续上页)

```
 // 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.
 }

 // ... 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.
```

(下页继续)

(续上页)

```
 // 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 pucQueueStorageBuffer 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 pucQueueStorageBuffer 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）

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;

 // Create a queue capable of containing 10 uint32_t values.
 xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );
```

(下页继续)

(续上页)

```
// 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;

 // Create a queue to hold one uint32_t value.  It is strongly
 // recommended *not* to use xQueueOverwrite() on queues that can
```

(下页继续)

(续上页)

```
// 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 xHigherPrioritTaskWoken;

 // 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 )
 {
     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** –**[out]** 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 )
{
    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** –**[out]** 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( xHigherPrioritytaskWoken == 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** –**[out]** 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** –**[out]** 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.

Semaphore API

Header File

- components/freertos/FreeRTOS-Kernel/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 xSemaphoreCreateBinaryStatic().
 // The semaphore's data structures will be placed in the xSemaphoreBuffer
 // variable, the address of which is passed into the function.  The
```

(下页继续)

(续上页)

```
 // 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 = xSemaphoreCreateBinaryStatic( &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** –**[out]** 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.

xSemaphoreCreateCounting(uxMaxCount, uxInitialCount)

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 mutexs 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.
 }
}
```

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 mutexs 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;

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
```

(下页继续)

(续上页)

```
 // 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.
}
```

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 )
{
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.
 }
}
```

返回 xSemaphore Handle to the created mutex semaphore. Should be of type SemaphoreHandle_t.

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.
- **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 If the recursive mutex was successfully created then a handle to the created recursive mutex is returned. If pxMutexBuffer was NULL then NULL is returned.

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 )
{
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.
```

(下页继续)

(续上页)

```
 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.

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**

Timer API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/timers.h

Functions

TimerHandle_t **xTimerCreate**(const char *const pcTimerName, const TickType_t xTimerPeriodInTicks, const UBaseType_t uxAutoReload, void *const pvTimerID, *TimerCallbackFunction_t* pxCallbackFunction)

TimerHandle_t xTimerCreate(const char * const pcTimerName, TickType_t xTimerPeriodInTicks, UBaseType_t uxAutoReload, void * 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 )
* {
* 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 );
*     }
```

(下页继续)

(续上页)

```
* }
*
* 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 ),   // 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.
*             }
*         }
*     }
*
*     // ...
*     // 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.

- **uxAutoReload** –If uxAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the xTimerPeriodInTicks parameter. If uxAutoReload 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 UBaseType_t uxAutoReload, void *const pvTimerID, *TimerCallbackFunction_t* pxCallbackFunction, StaticTimer_t *pxTimerBuffer)

TimerHandle_t xTimerCreateStatic(const char * const pcTimerName, TickType_t xTimerPeriodInTicks, UBaseType_t uxAutoReload, void * 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 );
```

(下页继续)

(续上页)

```
*
*     // 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.
*     // ...
*
*     // 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.

- **uxAutoReload** –If uxAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the xTimerPeriodInTicks parameter. If uxAutoReload 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)

void *pvTimerGetTimerID(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)

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)

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)

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)

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
```

(下页继续)

(续上页)

```
*       // 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)

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)

const char * const 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 UBaseType_t uxAutoReload)

void vTimerSetReloadMode(TimerHandle_t xTimer, const UBaseType_t uxAutoReload);

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.
- **uxAutoReload** –If uxAutoReload 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 uxAutoReload is set to pdFALSE then the timer will be a one-shot timer and enter the dormant state after it expires.

UBaseType_t **uxTimerGetReloadMode** (*TimerHandle_t* xTimer)

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)

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)

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)

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 thie idle task
- **pulTimerTaskStackSize** –A pointer to the number of elements that will fit in the allocated stack buffer

Macros

tmrCOMMAND_EXECUTE_CALLBACK_FROM_ISR

tmrCOMMAND_EXECUTE_CALLBACK

tmrCOMMAND_START_DONT_TRACE

tmrCOMMAND_START

tmrCOMMAND_RESET

tmrCOMMAND_STOP

tmrCOMMAND_CHANGE_PERIOD

tmrCOMMAND_DELETE

tmrFIRST_FROM_ISR_COMMAND

tmrCOMMAND_START_FROM_ISR

tmrCOMMAND_RESET_FROM_ISR

tmrCOMMAND_STOP_FROM_ISR

tmrCOMMAND_CHANGE_PERIOD_FROM_ISR

xTimerStart（xTimer, xTicksToWait）

BaseType_t xTimerStart(TimerHandle_t xTimer, TickType_t 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）

BaseType_t xTimerStop(TimerHandle_t xTimer, TickType_t 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)

BaseType_t xTimerChangePeriod(TimerHandle_t xTimer, TickType_t xNewPeriod, TickType_t 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 ) )"
*     {
*         // 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)

BaseType_t xTimerDelete(TimerHandle_t xTimer, TickType_t 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)

BaseType_t xTimerReset(TimerHandle_t xTimer, TickType_t 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 )
*     {
*         // 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.
*                                   );
```

(下页继续)

(续上页)

```
*
*     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)

BaseType_t xTimerStartFromISR(TimerHandle_t xTimer, BaseType_t *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
```

(下页继续)

(续上页)

```
* // 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).
*     }
* }
*
```

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 ser-

vice/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)

BaseType_t xTimerStopFromISR(TimerHandle_t xTimer, BaseType_t *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)

BaseType_t xTimerChangePeriodFromISR(TimerHandle_t xTimer, TickType_t xNewPeriod, BaseType_t *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 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.
- **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)

BaseType_t xTimerResetFromISR(TimerHandle_t xTimer, BaseType_t *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 )
*     {
*         // 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).
*     }
```

(下页继续)

(续上页)

```
* }
*
```

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)

typedef void (***PendedFunction_t**)(void*, uint32_t)

Event Group API

Header File

- components/freertos/FreeRTOS-Kernel/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 )
     {
         // 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.

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 ) )
    {
        // 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 )
    {
        // 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( ;; )
  {
     // 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 );
```

(下页继续)

(续上页)

```
        // 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.
                         BIT_0 | BIT_4 ); // The bits being set.

     if( xResult == pdPASS )
     {
          // The message was posted successfully.
     }
}
```

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
          // 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**

Stream Buffer API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/stream_buffer.h

Functions

BaseType_t **xStreamBufferGetStaticBuffers**(*StreamBufferHandle_t* xStreamBuffer, uint8_t **ppucStreamBufferStorageArea, StaticStreamBuffer_t **ppxStaticStreamBuffer)

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.

: 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.

: 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
```

(下页继续)

(续上页)

```
 // 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 taskYIELD_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.
 taskYIELD_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.

: 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
```

(下页继续)

(续上页)

```
 // available.
 xReceivedBytes = xStreamBufferReceive( xStreamBuffer,
                                        ( void * ) ucRxData,
                                        sizeof( ucRxData ),
                                        xBlockTime );

 if( xReceivedBytes > 0 )
 {
     // A ucRxData contains another xRecievedBytes 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 )
```

(下页继续)

(续上页)

```
{
    // ucRxData contains xReceivedBytes read from the stream buffer.
    // Process the stream here....
}

// 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 taskYIELD_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.
taskYIELD_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

xStreamBufferCreate(xBufferSizeBytes, xTriggerLevelBytes)

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.
```

(下页继续)

(续上页)

```
 }
}
```

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.

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.

xStreamBufferCreateStatic(xBufferSizeBytes, xTriggerLevelBytes, pucStreamBufferStorageArea, pxStaticStreamBuffer)

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( ucBufferStorage ),
                                            xTriggerLevel,
                                            ucBufferStorage,
                                            &xStreamBufferStruct );

 // 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.
```

(下页继续)

(续上页)

```
 // 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 + 1 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.

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**

Message Buffer API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/message_buffer.h

Macros

xMessageBufferCreate(xBufferSizeBytes)

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
```

(下页继续)

(续上页)

```
// dynamically.  Each message added to the buffer consumes an additional 4
// bytes which are used to hold the lengh 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.

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.

xMessageBufferCreateStatic(xBufferSizeBytes, pucMessageBufferStorageArea, pxStaticMessageBuffer)

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( ucBufferStorage ),
                                              ucBufferStorage,
                                              &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 + 1 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.

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)

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.

: 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 );
```

(下页继续)

(续上页)

```
 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.

: 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;
```

(下页继续)

(续上页)

```
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.
 }

 // 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 *pxHigherPriority-TaskWoken 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. *px-HigherPriorityTaskWoken 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.

: 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.

: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implemen-

tation, 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:

```
// 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 xMessageBufferCreateStatic(). 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)

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 void ***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.

2.10.12 FreeRTOS (Supplemental Features)

ESP-IDF provides multiple features to supplement the features offered by FreeRTOS. These supplemental features are available on all FreeRTOS implementations supported by ESP-IDF (i.e., ESP-IDF FreeRTOS and Amazon SMP FreeRTOS). This document describes these supplemental features and is split into the following sections:

Contents

Overview

ESP-IDF adds various new features to supplement the capabilities of FreeRTOS as follows:

- **Ring buffers**: Ring buffers provide a FIFO buffer that can accept entries of arbitrary lengths.
- **ESP-IDF Tick and Idle Hooks**: ESP-IDF provides multiple custom tick interrupt hooks and idle task hooks that are more numerous and more flexible when compared to FreeRTOS tick and idle hooks.
- **Thread Local Storage Pointer (TLSP) Deletion Callbacks**: TLSP Deletion callbacks are run automatically when a task is deleted, thus allowing users to clean up their TLSPs automatically.
- **Component Specific Properties**: Currently added only one component specific property `ORIG_INCLUDE_PATH`.

Ring Buffers

FreeRTOS provides stream buffers and message buffers as the primary mechanisms to send arbitrarily sized data between tasks and ISRs. However, FreeRTOS stream buffers and message buffers have the following limitations:

- Strictly single sender and single receiver
- Data is passed by copy
- Unable to reserve buffer space for a deferred send (i.e., send acquire)

Therefore, ESP-IDF provides a separate ring buffer implementation to address the issues above. ESP-IDF ring buffers are strictly FIFO buffers that supports arbitrarily sized items. Ring buffers are a more memory efficient alternative to FreeRTOS queues in situations where the size of items is variable. The capacity of a ring buffer is not measured by the number of items it can store, but rather by the amount of memory used for storing items. The ring buffer provides APIs to send an item, or to allocate space for an item in the ring buffer to be filled manually by the user. For efficiency reasons, **items are always retrieved from the ring buffer by reference**. As a result, all retrieved items *must also be returned* to the ring buffer by using `vRingbufferReturnItem()` or `vRingbufferReturnItemFromISR()`, in order for them to be removed from the ring buffer completely. The ring buffers are split into the three following types:

No-Split buffers will guarantee that an item is stored in contiguous memory and will not attempt to split an item under any circumstances. Use No-Split buffers when items must occupy contiguous memory. *Only this buffer type allows you to get the data item address and write to the item by yourself.* Refer the documentation of the functions `xRingbufferSendAcquire()` and `xRingbufferSendComplete()` for more details.

Allow-Split buffers will allow an item to be split in two parts when wrapping around the end of the buffer if there is enough space at the tail and the head of the buffer combined to store the item. Allow-Split buffers are more memory efficient than No-Split buffers but can return an item in two parts when retrieving.

Byte buffers do not store data as separate items. All data is stored as a sequence of bytes, and any number of bytes can be sent or retrieved each time. Use byte buffers when separate items do not need to be maintained (e.g. a byte stream).

备注： No-Split buffers and Allow-Split buffers will always store items at 32-bit aligned addresses. Therefore, when retrieving an item, the item pointer is guaranteed to be 32-bit aligned. This is useful especially when you need to send some data to the DMA.

备注： Each item stored in No-Split or Allow-Split buffers will **require an additional 8 bytes for a header**. Item sizes will also be rounded up to a 32-bit aligned size (multiple of 4 bytes), however the true item size is recorded within the header. The sizes of No-Split and Allow-Split buffers will also be rounded up when created.

Usage The following example demonstrates the usage of *xRingbufferCreate()* and *xRingbufferSend()* to create a ring buffer and then send an item to it.

```
#include "freertos/ringbuf.h"
static char tx_item[] = "test_item";

...

    //Create ring buffer
    RingbufHandle_t buf_handle;
    buf_handle = xRingbufferCreate(1028, RINGBUF_TYPE_NOSPLIT);
    if (buf_handle == NULL) {
        printf("Failed to create ring buffer\n");
    }

    //Send an item
    UBaseType_t res =  xRingbufferSend(buf_handle, tx_item, sizeof(tx_item), pdMS_
↪TO_TICKS(1000));
    if (res != pdTRUE) {
        printf("Failed to send item\n");
    }
```

The following example demonstrates the usage of *xRingbufferSendAcquire()* and *xRingbufferSendComplete()* instead of *xRingbufferSend()* to acquire memory on the ring buffer (of type *RINGBUF_TYPE_NOSPLIT*) and then send an item to it. This adds one more step, but allows getting the address of the memory to write to, and writing to the memory yourself.

```
#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)))

...

    //Retrieve space for DMA descriptor and corresponding data buffer
    //This has to be done with SendAcquire, or the address may be different when␣
↪we copy
    dma_item_t item;
    UBaseType_t res =  xRingbufferSendAcquire(buf_handle,
                       &item, DMA_ITEM_SIZE(buffer_size), pdMS_TO_TICKS(1000));
```

(下页继续)

(续上页)

```
    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,
    };
    //Actually send to the ring buffer for consumer to use
    res = xRingbufferSendComplete(buf_handle, &item);
    if (res != pdTRUE) {
        printf("Failed to send item\n");
    }
```

The following example demonstrates retrieving and returning an item from a **No-Split ring buffer** using *xRingbufferReceive()* and *vRingbufferReturnItem()*

```
...

    //Receive an item from no-split ring buffer
    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");
        //Return Item
        vRingbufferReturnItem(buf_handle, (void *)item);
    } else {
        //Failed to receive item
        printf("Failed to receive item\n");
    }
```

The following example demonstrates retrieving and returning an item from an **Allow-Split ring buffer** using *xRingbufferReceiveSplit()* and *vRingbufferReturnItem()*

```
...

    //Receive an item from allow-split ring buffer
    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));

    //Check received item
    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]);
```

(下页继续)

(续上页)

```
            }
            vRingbufferReturnItem(buf_handle, (void *)item2);
        }
        printf("\n");
    } else {
        //Failed to receive item
        printf("Failed to receive item\n");
    }
```

The following example demonstrates retrieving and returning an item from a **byte buffer** using *xRingbufferReceiveUpTo()* and *vRingbufferReturnItem()*

```
...

    //Receive data from byte buffer
    size_t item_size;
    char *item = (char *)xRingbufferReceiveUpTo(buf_handle, &item_size, pdMS_TO_
↪TICKS(1000), sizeof(tx_item));

    //Check received data
    if (item != NULL) {
        //Print item
        for (int i = 0; i < item_size; i++) {
            printf("%c", item[i]);
        }
        printf("\n");
        //Return Item
        vRingbufferReturnItem(buf_handle, (void *)item);
    } else {
        //Failed to receive item
        printf("Failed to receive item\n");
    }
```

For ISR safe versions of the functions used above, call *xRingbufferSendFromISR()*, *xRingbufferReceiveFromISR()*, *xRingbufferReceiveSplitFromISR()*, *xRingbufferReceiveUpToFromISR()*, and *vRingbufferReturnItemFromISR()*

备注: Two calls to RingbufferReceive[UpTo][FromISR]() are required if the bytes wraps around the end of the ring buffer.

Sending to Ring Buffer The following diagrams illustrate the differences between No-Split and Allow-Split buffers as compared to byte buffers with regard to sending items/data. The diagrams assume that three items of sizes **18, 3, and 27 bytes** are sent respectively to a **buffer of 128 bytes**.

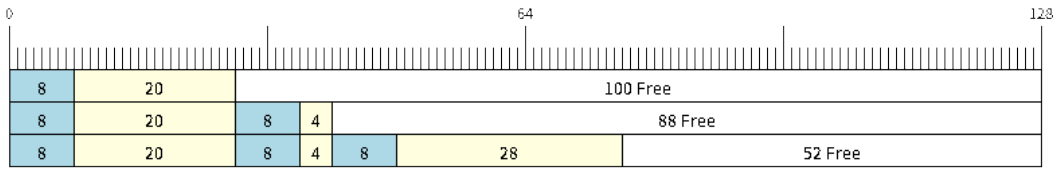


图 10: Sending items to No-Split or Allow-Split ring buffers

For No-Split and Allow-Split buffers, a header of 8 bytes precedes every data item. Furthermore, the space occupied by each item is **rounded up to the nearest 32-bit aligned size** in order to maintain overall 32-bit alignment.

However, the true size of the item is recorded inside the header which will be returned when the item is retrieved.

Referring to the diagram above, the 18, 3, and 27 byte items are **rounded up to 20, 4, and 28 bytes** respectively. An 8 byte header is then added in front of each item.

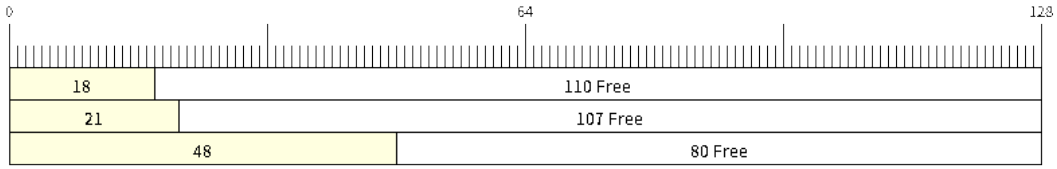


图 11: Sending items to byte buffers

Byte buffers treat data as a sequence of bytes and does not incur any overhead (no headers). As a result, all data sent to a byte buffer is merged into a single item.

Referring to the diagram above, the 18, 3, and 27 byte items are sequentially written to the byte buffer and **merged into a single item of 48 bytes**.

Using SendAcquire and SendComplete Items in No-Split buffers are acquired (by `SendAcquire`) in strict FIFO order and must be sent to the buffer by `SendComplete` for the data to be accessible by the consumer. Multiple items can be sent or acquired without calling `SendComplete`, and the items do not necessarily need to be completed in the order they were acquired. However, the receiving of data items must occur in FIFO order, therefore not calling `SendComplete` for the earliest acquired item will prevent the subsequent items from being received.

The following diagrams illustrate what will happen when `SendAcquire` and `SendComplete` don't happen in the same order. At the beginning, there is already a data item of 16 bytes sent to the ring buffer. Then `SendAcquire` is called to acquire space of 20, 8, 24 bytes on the ring buffer.

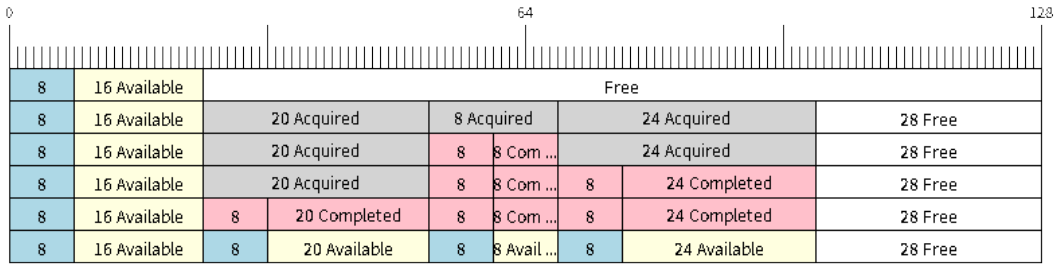


图 12: SendAcquire/SendComplete items in No-Split ring buffers

After that, we fill (use) the buffers, and send them to the ring buffer by `SendComplete` in the order of 8, 24, 20. When 8 bytes and 24 bytes data are sent, the consumer still can only get the 16 bytes data item. Hence, if `SendComplete` is not called for the 20 bytes, it will not be available, nor will the data items following the 20 bytes item.

When the 20 bytes item is finally completed, all the 3 data items can be received now, in the order of 20, 8, 24 bytes, right after the 16 bytes item existing in the buffer at the beginning.

Allow-Split buffers and byte buffers do not allow using `SendAcquire` or `SendComplete` since acquired buffers are required to be complete (not wrapped).

Wrap around The following diagrams illustrate the differences between No-Split, Allow-Split, and byte buffers when a sent item requires a wrap around. The diagrams assume a buffer of **128 bytes** with **56 bytes of free space that wraps around** and a sent item of **28 bytes**.

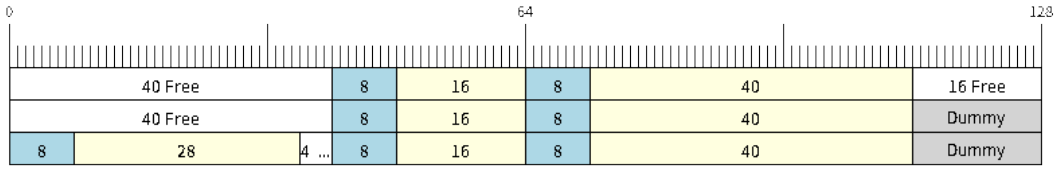


图 13: Wrap around in No-Split buffers

No-Split buffers will **only store an item in continuous free space and will not split an item under any circumstances**. When the free space at the tail of the buffer is insufficient to completely store the item and its header, the free space at the tail will be **marked as dummy data**. The buffer will then wrap around and store the item in the free space at the head of the buffer.

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to store the 28 byte item. Therefore, the 16 bytes is marked as dummy data and the item is written to the free space at the head of the buffer instead.

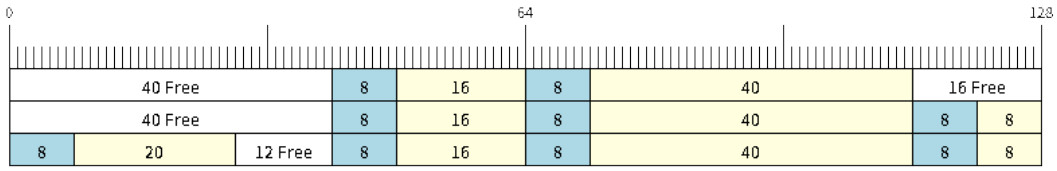


图 14: Wrap around in Allow-Split buffers

Allow-Split buffers will attempt to **split the item into two parts** when the free space at the tail of the buffer is insufficient to store the item data and its header. Both parts of the split item will have their own headers (therefore incurring an extra 8 bytes of overhead).

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to store the 28 byte item. Therefore, the item is split into two parts (8 and 20 bytes) and written as two parts to the buffer.

备 注: Allow-Split buffers treat both parts of the split item as two separate items, therefore call `xRingbufferReceiveSplit()` instead of `xRingbufferReceive()` to receive both parts of a split item in a thread safe manner.

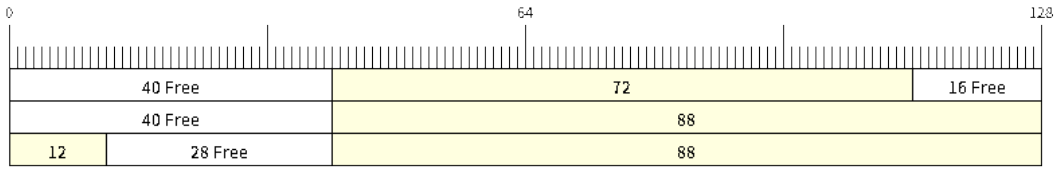


图 15: Wrap around in byte buffers

Byte buffers will **store as much data as possible into the free space at the tail of buffer**. The remaining data will then be stored in the free space at the head of the buffer. No overhead is incurred when wrapping around in byte buffers.

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to completely store the 28 bytes of data. Therefore, the 16 bytes of free space is filled with data, and the remaining 12 bytes are written to the free space at the head of the buffer. The buffer now contains data in two separate continuous parts, and each continuous part will be treated as a separate item by the byte buffer.

Retrieving/Returning The following diagrams illustrate the differences between No-Split and Allow-Split buffers as compared to byte buffers in retrieving and returning data.

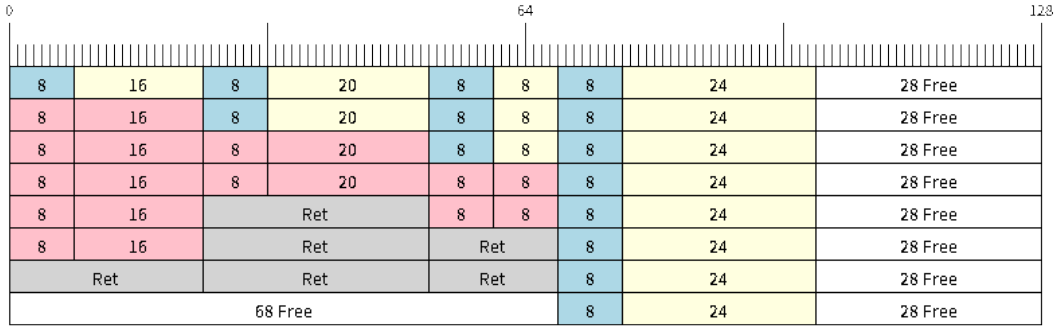


图 16: Retrieving/Returning items in No-Split and Allow-Split ring buffers

Items in No-Split buffers and Allow-Split buffers are **retrieved in strict FIFO order** and **must be returned** for the occupied space to be freed. Multiple items can be retrieved before returning, and the items do not necessarily need to be returned in the order they were retrieved. However, the freeing of space must occur in FIFO order, therefore not returning the earliest retrieved item will prevent the space of subsequent items from being freed.

Referring to the diagram above, the **16, 20, and 8 byte items are retrieved in FIFO order**. However, the items are not returned in the order they were retrieved. First, the 20 byte item is returned followed by the 8 byte and the 16 byte items. The space is not freed until the first item, i.e., the 16 byte item is returned.

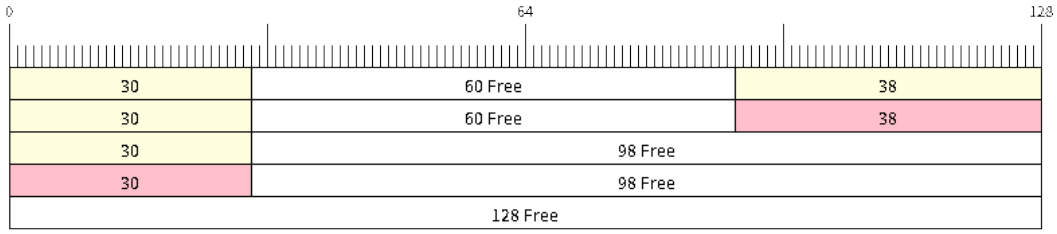


图 17: Retrieving/Returning data in byte buffers

Byte buffers **do not allow multiple retrievals before returning** (every retrieval must be followed by a return before another retrieval is permitted). When using `xRingbufferReceive()` or `xRingbufferReceiveFromISR()`, all continuous stored data will be retrieved. `xRingbufferReceiveUpTo()` or `xRingbufferReceiveUpToFromISR()` can be used to restrict the maximum number of bytes retrieved. Since every retrieval must be followed by a return, the space will be freed as soon as the data is returned.

Referring to the diagram above, the 38 bytes of continuous stored data at the tail of the buffer is retrieved, returned, and freed. The next call to `xRingbufferReceive()` or `xRingbufferReceiveFromISR()` then wraps around and does the same to the 30 bytes of continuous stored data at the head of the buffer.

Ring Buffers with Queue Sets Ring buffers can be added to FreeRTOS queue sets using *xRingbufferAddToQueueSetRead()* such that every time a ring buffer receives an item or data, the queue set is notified. Once added to a queue set, every attempt to retrieve an item from a ring buffer should be preceded by a call to *xQueueSelectFromSet()*. To check whether the selected queue set member is the ring buffer, call *xRingbufferCanRead()*.

The following example demonstrates queue set usage with ring buffers.

```
#include "freertos/queue.h"
#include "freertos/ringbuf.h"

...

    //Create ring buffer and queue set
    RingbufHandle_t buf_handle = xRingbufferCreate(1028, RINGBUF_TYPE_NOSPLIT);
    QueueSetHandle_t queue_set = xQueueCreateSet(3);

    //Add ring buffer to queue set
    if (xRingbufferAddToQueueSetRead(buf_handle, queue_set) != pdTRUE) {
        printf("Failed to add to queue set\n");
    }

...

    //Block on queue set
    QueueSetMemberHandle_t member = xQueueSelectFromSet(queue_set, pdMS_TO_
↪TICKS(1000));

    //Check if member is ring buffer
    if (member != NULL && xRingbufferCanRead(buf_handle, member) == pdTRUE) {
        //Member is ring buffer, receive item from ring buffer
        size_t item_size;
        char *item = (char *)xRingbufferReceive(buf_handle, &item_size, 0);

        //Handle item
        ...

    } else {
        ...
    }
```

Ring Buffers with Static Allocation The *xRingbufferCreateStatic()* can be used to create ring buffers with specific memory requirements (such as a ring buffer being allocated in external RAM). All blocks of memory used by a ring buffer must be manually allocated beforehand then passed to the *xRingbufferCreateStatic()* to be initialized as a ring buffer. These blocks include the following:

- The ring buffer's data structure of type *StaticRingbuffer_t*
- The ring buffer's storage area of size `xBufferSize`. Note that `xBufferSize` must be 32-bit aligned for No-Split and Allow-Split buffers.

The manner in which these blocks are allocated will depend on the users requirements (e.g. all blocks being statically declared, or dynamically allocated with specific capabilities such as external RAM).

备 注: When deleting a ring buffer created via *xRingbufferCreateStatic()*, the function *vRingbufferDelete()* will not free any of the memory blocks. This must be done manually by the user after *vRingbufferDelete()* is called.

The code snippet below demonstrates a ring buffer being allocated entirely in external RAM.

```
#include "freertos/ringbuf.h"
#include "freertos/semphr.h"
#include "esp_heap_caps.h"

#define BUFFER_SIZE     400      //32-bit aligned size
#define BUFFER_TYPE     RINGBUF_TYPE_NOSPLIT
...

//Allocate ring buffer data structure and storage area into external RAM
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);

//Create a ring buffer with manually allocated memory
RingbufHandle_t handle = xRingbufferCreateStatic(BUFFER_SIZE, BUFFER_TYPE, buffer_
↪storage, buffer_struct);

...

//Delete the ring buffer after used
vRingbufferDelete(handle);

//Manually free all blocks of memory
free(buffer_struct);
free(buffer_storage);
```

ESP-IDF Tick and Idle Hooks

FreeRTOS allows applications to provide a tick hook and an idle hook at compile time:

- FreeRTOS tick hook can be enabled via the *CONFIG_FREERTOS_USE_TICK_HOOK* option. The application must provide the `void vApplicationTickHook( void )` callback.
- FreeRTOS idle hook can be enabled via the *CONFIG_FREERTOS_USE_IDLE_HOOK* option. The application must provide the `void vApplicationIdleHook( void )` callback.

However, the FreeRTOS tick hook and idle hook have the following draw backs:

- The FreeRTOS hooks are registered at compile time
- Only one of each hook can be registered
- On multi-core targets, the FreeRTOS hooks are symmetric, meaning each CPU's tick interrupt and idle tasks ends up calling the same hook.

Therefore, ESP-IDF tick and idle hooks are provided to supplement the features of FreeRTOS tick and idle hooks. The ESP-IDF hooks have the following features:

- The hooks can be registered and deregistered at run-time
- Multiple hooks can be registered (with a maximum of 8 hooks of each type per CPU)
- On multi-core targets, the hooks can be asymmetric, meaning different hooks can be registered to each CPU

ESP-IDF hooks can be registered and deregistered using the following APIs:

- For tick hooks:
  - Register using *esp_register_freertos_tick_hook()* or *esp_register_freertos_tick_hook_for_cpu()*
  - Deregister using *esp_deregister_freertos_tick_hook()* or *esp_deregister_freertos_tick_hook_for_cpu()*
- For idle hooks:
  - Register using *esp_register_freertos_idle_hook()* or *esp_register_freertos_idle_hook_for_cpu()*
  - Deregister using *esp_deregister_freertos_idle_hook()* or *esp_deregister_freertos_idle_hook_for_cpu()*

备注： The tick interrupt stays active while the cache is disabled, therefore any tick hook (FreeRTOS or ESP-IDF) functions must be placed in internal RAM. Please refer to the *SPI flash API documentation* for more details.

TLSP Deletion Callbacks

Vanilla FreeRTOS provides a Thread Local Storage Pointers (TLSP) feature. These are pointers stored directly in the Task Control Block (TCB) of a particular task. TLSPs allow each task to have its own unique set of pointers to data structures. Vanilla FreeRTOS expects users to…

- set a task's TLSPs by calling `vTaskSetThreadLocalStoragePointer()` after the task has been created.
- get a task's TLSPs by calling `pvTaskGetThreadLocalStoragePointer()` during the task's lifetime.
- free the memory pointed to by the TLSPs before the task is deleted.

However, there can be instances where users may want the freeing of TLSP memory to be automatic. Therefore, ESP-IDF provides the additional feature of TLSP deletion callbacks. These user provided deletion callbacks are called automatically when a task is deleted, thus allowing the TLSP memory to be cleaned up without needing to add the cleanup logic explicitly to the code of every task.

The TLSP deletion callbacks are set in a similar fashion to the TLSPs themselves.

- `vTaskSetThreadLocalStoragePointerAndDelCallback()` sets both a particular TLSP and its associated callback.
- Calling the Vanilla FreeRTOS function `vTaskSetThreadLocalStoragePointer()` will simply set the TLSP's associated Deletion Callback to *NULL* meaning that no callback will be called for that TLSP during task deletion.

When implementing TLSP callbacks, users should note the following:

- The callback **must never attempt to block or yield** and critical sections should be kept as short as possible
- The callback is called shortly before a deleted task's memory is freed. Thus, the callback can either be called from `vTaskDelete()` itself, or from the idle task.

IDF Additional API

The freertos/esp_additions/include/freertos/idf_additions.h header contains FreeRTOS related helper functions added by ESP-IDF. Users can include this header via `#include "freertos/idf_additions.h"`.

Component Specific Properties

Besides standard component variables that are available with basic cmake build properties, FreeRTOS component also provides arguments (only one so far) for simpler integration with other modules:

- *ORIG_INCLUDE_PATH* - contains an absolute path to freertos root include folder. Thus instead of *#include "freertos/FreeRTOS.h"* you can refer to headers directly: *#include "FreeRTOS.h"*.

API Reference

Ring Buffer API

Header File

- components/esp_ringbuf/include/freertos/ringbuf.h

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 to remove from the queue set
- **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

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**

Hooks API

Header File

- components/esp_system/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)

Additional API

Header File

- components/freertos/esp_additions/include/freertos/idf_additions.h

Functions

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()

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.

2.10.13 Heap Memory Allocation

Stack and Heap

ESP-IDF applications use the common computer architecture patterns of *stack* (dynamic memory allocated by program control flow) and *heap* (dynamic memory allocated by function calls), as well as statically allocated memory (allocated at compile time).

Because ESP-IDF is a multi-threaded RTOS environment, each RTOS task has its own stack. By default, each of these stacks is allocated from the heap when the task is created. (See `xTaskCreateStatic()` for the alternative where stacks are statically allocated.)

Because ESP32-C2 uses multiple types of RAM, it also contains multiple heaps with different capabilities. A capabilities-based memory allocator allows apps to make heap allocations for different purposes.

For most purposes, the standard libc `malloc()` and `free()` functions can be used for heap allocation without any special consideration.

However, in order to fully make use of all of the memory types and their characteristics, ESP-IDF also has a capabilities-based heap memory allocator. If you want to have memory with certain properties (for example, *DMA-Capable Memory* or executable-memory), you can create an OR-mask of the required capabilities and pass that to `heap_caps_malloc()`.

Memory Capabilities

The ESP32-C2 contains multiple types of RAM:

- DRAM (Data RAM) is memory used to hold data. This is the most common kind of memory accessed as heap.
- IRAM (Instruction RAM) usually holds executable data only. If accessed as generic memory, all accesses must be *32-bit aligned*.
- D/IRAM is RAM which can be used as either Instruction or Data RAM.

For more details on these internal memory types, see 存储器类型.

DRAM uses capability `MALLOC_CAP_8BIT` (accessible in single byte reads and writes). To test the free DRAM heap size at runtime, call cpp:func:*heap_caps_get_free_size(MALLOC_CAP_8BIT)*.

When calling `malloc()`, the ESP-IDF `malloc()` implementation internally calls cpp:func:*heap_caps_malloc_default(size)*. This will allocate memory with capability `MALLOC_CAP_DEFAULT`, which is byte-addressable.

Because `malloc()` uses the capabilities-based allocation system, memory allocated using `heap_caps_malloc()` can be freed by calling the standard `free()` function.

Available Heap

DRAM At startup, the DRAM heap contains all data memory which is not statically allocated by the app. Reducing statically allocated buffers will increase the amount of available free heap.

To find the amount of statically allocated memory, use the *idf.py size* command.

备注： At runtime, the available heap DRAM may be less than calculated at compile time, because at startup some memory is allocated from the heap before the FreeRTOS scheduler is started (including memory for the stacks of initial FreeRTOS tasks).

IRAM At startup, the IRAM heap contains all instruction memory which is not used by the app executable code.

The *idf.py size* command can be used to find the amount of IRAM used by the app.

D/IRAM Some memory in the ESP32-C2 is available as either DRAM or IRAM. If memory is allocated from a D/IRAM region, the free heap size for both types of memory will decrease.

Heap Sizes At startup, all ESP-IDF apps log a summary of all heap addresses (and sizes) at level 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
```

Finding available heap See *Heap Information*.

Special Capabilities

DMA-Capable Memory Use the `MALLOC_CAP_DMA` flag to allocate memory which is suitable for use with hardware DMA engines (for example SPI and I2S). This capability flag excludes any external PSRAM.

32-Bit Accessible Memory If a certain memory structure is only addressed in 32-bit units, for example an array of ints or pointers, it can be useful to allocate it with the `MALLOC_CAP_32BIT` flag. This also allows the allocator to give out IRAM memory; something which it can't do for a normal malloc() call. This can help to use all the available memory in the ESP32-C2.

Memory allocated with `MALLOC_CAP_32BIT` can *only* be accessed via 32-bit reads and writes, any other type of access will generate a fatal LoadStoreError exception.

Thread Safety

Heap functions are thread safe, meaning they can be called from different tasks simultaneously without any limitations.

It is technically possible to call `malloc`, `free`, and related functions from interrupt handler (ISR) context (see *Calling heap related functions from ISR*). However this is not recommended, as heap function calls may delay other interrupts. It is strongly recommended to refactor applications so that any buffers used by an ISR are pre-allocated outside of the ISR. Support for calling heap functions from ISRs may be removed in a future update.

Calling heap related functions from ISR

The following functions from the heap component can be called form interrupt handler (ISR):

- *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()*

Note however this practice is strongly discouraged.

Heap Tracing & Debugging

The following features are documented on the *Heap Memory Debugging* page:

- *Heap Information* (free space, etc.)
- *Heap allocation and free function hooks*
- *Heap Corruption Detection*
- *Heap Tracing* (memory leak detection, monitoring, etc.)

Implementation Notes

Knowledge about the regions of memory in the chip comes from the "soc" component, which contains memory layout information for the chip, and the different capabilities of each region. Each region's capabilities are prioritised, so that (for example) dedicated DRAM and IRAM regions will be used for allocations ahead of the more versatile D/IRAM regions.

Each contiguous region of memory contains its own memory heap. The heaps are created using the *multi_heap* functionality. multi_heap allows any contiguous region of memory to be used as a heap.

The heap capabilities allocator uses knowledge of the memory regions to initialize each individual heap. Allocation functions in the heap capabilities API will find the most appropriate heap for the allocation (based on desired capabilities, available space, and preferences for each region's use) and then calling `multi_heap_malloc()` for the heap situated in that particular region.

Calling `free()` involves finding the particular heap corresponding to the freed address, and then calling `multi_heap_free()` on that particular multi_heap instance.

API Reference - Heap Allocation

Header File

- components/heap/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.

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 paramters

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 paramters

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.

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_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

API Reference - Initialisation

Header File

- components/heap/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 Reference - Multi Heap API

(Note: The multi heap API is used internally by the heap capabilities allocator. Most IDF programs will never need to call this API directly.)

Header File

- components/heap/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.

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.

2.10.14 Memory Management for MMU Supported Memory

Introduction

ESP32-C2 Memory Management Unit (MMU) is relatively simple. It can do memory address translation between physical memory addresses and virtual memory addresses. So CPU can access physical memories via virtual addresses. There are multiple types of virtual memory addresses, which have different capabilities.

ESP-IDF provides a memory mapping driver that manages the relation between these physical memory addresses and virtual memory addresses, so as to achieve some features such as reading from SPI Flash via a pointer.

Memory mapping driver is actually a capabilities-based virtual memory address allocator that allows apps to make virtual memory address allocations for different purposes. In the following chapters, we call this driver *esp_mmap* driver.

ESP-IDF also provides a memory synchronisation driver which can be used for potential memory desychronisation scenarios.

Physical Memory Types

Memory mapping driver currently supports mapping to following physical memory types:

- SPI Flash

Virtual Memory Capabilities

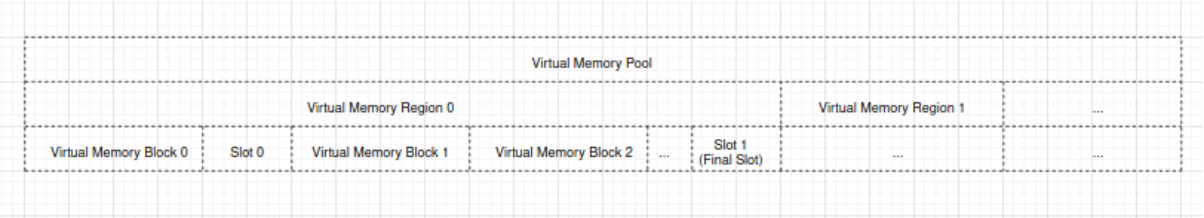
- `MMU_MEM_CAP_EXEC`. This capability indicates that the virtual memory address has the execute permission. Note this permission scope is within the MMU hardware.
- `MMU_MEM_CAP_READ`. This capability indicates that the virtual memory address has the read permission. Note this permission scope is within the MMU hardware.
- `MMU_MEM_CAP_WRITE`. This capability indicates that the virtual memory address has the write permission. Note this permission scope is within the MMU hardware.
- `MMU_MEM_CAP_32BIT`. This capability indicates that the virtual memory address allows for 32 bits or multiples of 32 bits access.
- `MMU_MEM_CAP_8BIT`. This capability indicates that the virtual memory address allows for 8 bits or multiples of 8 bits access.

You can call *`esp_mmu_map_get_max_consecutive_free_block_size()`* to know the largest consecutive mappable block size with certain capabilities.

Memory Management Drivers

Driver Concept

Terminology The virtual memory pool is made up with one or multiple virtual memory regions, see below figure:

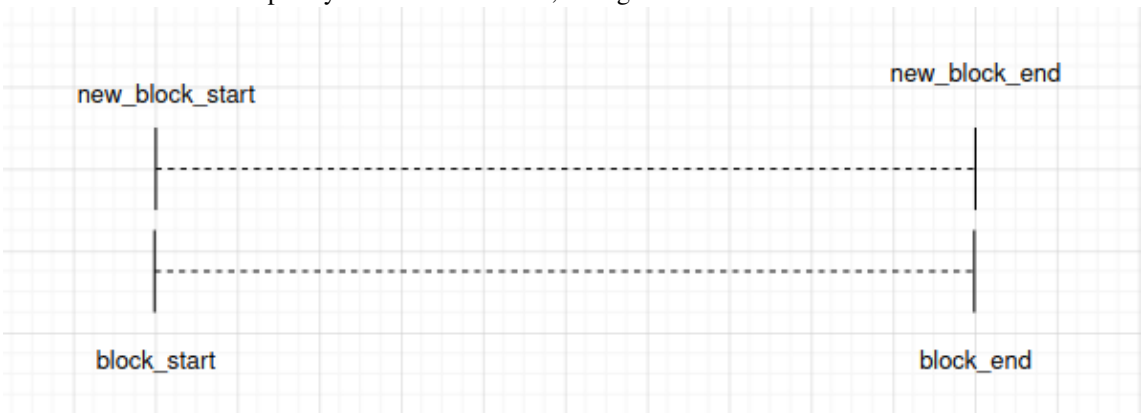


- A virtual memory pool stands for the whole virtual address range that can be mapped to physical memory
- A virtual memory region is a range of virtual address with same attributes
- A virtual memory block is a piece of virtual address range that is dynamically mapped.
- A slot is the virtual address range between two virtual memory blocks.
- A physical memory block is a piece of physical address range that is to-be-mapped or already mapped to a virtual memory block.
- Dynamical mapping is done by calling *esp_mmap* driver API `esp_mmu_map()`, this API will map the given physical memory block to a virtual memory block which is allocated by the *esp_mmap* driver.

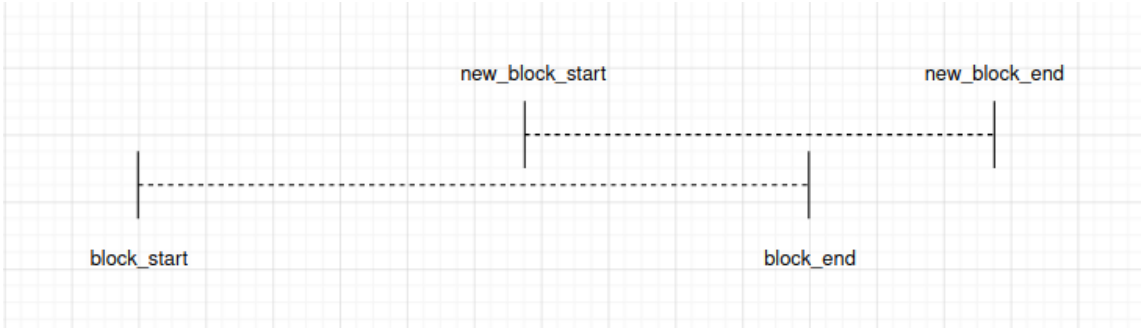
Relation between Memory Blocks When mapping a physical memory block A, block A can have one of the following relations with another previously mapped physical memory block B:

- Enclosed: block A is completely enclosed within block B, see figure below:

- Identical: block A is completely the same as block B, see figure below:



  Note *esp_mmap* driver will consider the identical scenario **the same as the enclosed scenario**.
- Overlapped: block A is overlapped with block B, see figure below:



There is a special condition, when block A entirely encloses block B, see figure below:

esp_mmap driver will consider this scenario **the same as the overlapped scenario**.

Driver Behaviour

Memory Map You can call `esp_mmu_map()` to do a dynamical mapping. This API will allocate a certain size of virtual memory block according to the virtual memory capabilities you selected, then map this virtual memory block to the physical memory block as you requested. The *esp_mmap* driver supports mapping to one or more types of physical memory, so you should specify the physical memory target when mapping.

By default, physical memory blocks and virtual memory blocks are one-to-one mapped. This means, when calling `esp_mmu_map()`:

- If it's the enclosed scenario, this API will return an `ESP_ERR_INVALID_STATE`. The *out_ptr* will be assigned to the start virtual memory address of the previously mapped one which encloses the to-be-mapped one.
- If it's the identical scenario, this API will behaves exactly the same as the enclosed scenario.
- If it's the overlapped scenario, this API will by default return an `ESP_ERR_INVALID_ARG`. This means, *esp_mmap* driver by default doesn't allow mapping a physical memory address to multiple virtual memory addresses.

Specially, you can use `ESP_MMU_MMAP_FLAG_PADDR_SHARED`. This flags stands for one-to-multiple mapping between a physical address and multiple virtual addresses:

- If it's the overlapped scenario, this API will allocate a new virtual memory block as requested, then map to the given physical memory block.

Memory Unmap You can call `esp_mmu_unmap()` to unmap a previously mapped memory block. This API will return an `ESP_ERR_NOT_FOUND` if you are trying to unmap a virtual memory block that isn't mapped to any physical memory block yet.

Memory Address Conversion The *esp_mmap* driver provides two helper APIs to do the conversion between virtual memory address and physical memory address.

- `esp_mmu_vaddr_to_paddr()`, convert virtual address to physical address.
- `esp_mmu_paddr_to_vaddr()`, convert physical address to virtual address.

Memory Synchronisation MMU supported physical memories can be accessed by one or multiple methods.

SPI Flash can be accessed by SPI1 (ESP-IDF *esp_flash* driver APIs), or by pointers. ESP-IDF *esp_flash* driver APIs have already considered the memory synchronisation, so users don't need to worry about this.

Thread Safety

APIs in *esp_mmu_map.h* are not guaranteed to be thread-safe.

APIs in *esp_cache.h* are guaranteed to be thread-safe.

API Reference

API Reference - ESP MMAP Driver

Header File

- components/esp_mm/include/esp_mmu_map.h

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 of 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.

API Reference - ESP MSYNC Driver

Header File

- components/esp_mm/include/esp_cache.h

Functions

esp_err_t **esp_cache_msync**(void *addr, size_t size, int flags)

Memory sync between Cache and external memory.

- 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 the PSRAM
  - 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

This API is cache-safe and thread-safe

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
  - 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 with the values that just written.

Cache msync flags

ESP_CACHE_MSYNC_FLAG_UNALIGNED

Allow writeback a block that are not aligned to the data cache line size.

2.10.15 Heap Memory Debugging

Overview

ESP-IDF integrates tools for requesting *heap information*, *detecting heap corruption*, and *tracing memory leaks*. These can help track down memory-related bugs.

For general information about the heap memory allocator, see the *Heap Memory Allocation* page.

Heap Information

To obtain information about the state of the heap:

- `xPortGetFreeHeapSize()` is a FreeRTOS function which returns the number of free bytes in the (data memory) heap. This is equivalent to calling `heap_caps_get_free_size(MALLOC_CAP_8BIT)`.
- *`heap_caps_get_free_size()`* can also be used to return the current free memory for different memory capabilities.
- *`heap_caps_get_largest_free_block()`* can be used to return the largest free block in the heap. This is the largest single allocation which is currently possible. Tracking this value and comparing to total free heap allows you to detect heap fragmentation.
- `xPortGetMinimumEverFreeHeapSize()` and the related *`heap_caps_get_minimum_free_size()`* can be used to track the heap "low watermark" since boot.
- *`heap_caps_get_info()`* returns a *`multi_heap_info_t`* structure which contains the information from the above functions, plus some additional heap-specific data (number of allocations, etc.).

- *heap_caps_print_heap_info()* prints a summary to stdout of the information returned by *heap_caps_get_info()*.
- *heap_caps_dump()* and *heap_caps_dump_all()* will output detailed information about the structure of each block in the heap. Note that this can be large amount of output.

Heap allocation and free function hooks

Heap allocation and free detection hooks allows you to be notified of every successful allocation and free operations: - Providing a definition of `esp_heap_trace_alloc_hook()` will allow you to be notified of every successful memory allocation operations - Providing a definition of `esp_heap_trace_free_hook()` will allow you to be notified of every memory free operations

To activate the feature, navigate to `Component config` -> `Heap Memory Debugging` in the configuration menu and select `Use allocation and free hooks` option (see *CONFIG_HEAP_USE_HOOKS*). `esp_heap_trace_alloc_hook()` and `esp_heap_trace_free_hook()` have weak declarations, it is not necessary to provide a declarations for both hooks. Since allocating and freeing memory is allowed even though strongly recommended against, `esp_heap_trace_alloc_hook()` and `esp_heap_trace_free_hook()` can potentially be called from ISR.

Heap Corruption Detection

Heap corruption detection allows you to detect various types of heap memory errors:

- Out of bounds writes & buffer overflow.
- Writes to freed memory.
- Reads from freed or uninitialized memory,

Assertions The heap implementation (`multi_heap.c`, etc.) includes a lot of assertions which will fail if the heap memory is corrupted. To detect heap corruption most effectively, ensure that assertions are enabled in the project configuration menu under `Compiler options` -> *CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL*.

If a heap integrity assertion fails, a line will be printed like `CORRUPT HEAP: multi_heap.c:225 detected at 0x3ffbb71c`. The memory address which is printed is the address of the heap structure which has corrupt content.

It's also possible to manually check heap integrity by calling *heap_caps_check_integrity_all()* or related functions. This function checks all of requested heap memory for integrity, and can be used even if assertions are disabled. If the integrity check prints an error, it will also contain the address(es) of corrupt heap structures.

Memory Allocation Failed Hook Users can use *heap_caps_register_failed_alloc_callback()* to register a callback that will be invoked every time an allocation operation fails.

Additionally, users can enable the generation of a system abort if an allocation operation fails by following the steps below: - In the project configuration menu, navigate to `Component config` -> `Heap Memory Debugging` and select `Abort if memory allocation fails` option (see *CONFIG_HEAP_ABORT_WHEN_ALLOCATION_FAILS*).

The example below shows how to register an allocation failure callback:

```
#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);
}
```

(下页继续)

(续上页)

```
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);
    ...
}
```

Finding Heap Corruption Memory corruption can be one of the hardest classes of bugs to find and fix, as one area of memory can be corrupted from a totally different place. Some tips:

- A crash with a `CORRUPT HEAP:` message will usually include a stack trace, but this stack trace is rarely useful. The crash is the symptom of memory corruption when the system realises the heap is corrupt, but usually the corruption happened elsewhere and earlier in time.
- Increasing the Heap memory debugging *Configuration* level to "Light impact" or "Comprehensive" can give you a more accurate message with the first corrupt memory address.
- Adding regular calls to `heap_caps_check_integrity_all()` or `heap_caps_check_integrity_addr()` in your code will help you pin down the exact time that the corruption happened. You can move these checks around to "close in on" the section of code that corrupted the heap.
- Based on the memory address which is being corrupted, you can use *JTAG debugging* to set a watchpoint on this address and have the CPU halt when it is written to.
- If you don't have JTAG, but you do know roughly when the corruption happens, then you can set a watchpoint in software just beforehand via `esp_cpu_set_watchpoint()`. A fatal exception will occur when the watchpoint triggers. The following is an example of how to use the function - `esp_cpu_set_watchpoint(0, (void *)addr, 4, ESP_WATCHPOINT_STORE)`. Note that watchpoints are per-CPU and are set on the current running CPU only, so if you don't know which CPU is corrupting memory then you will need to call this function on both CPUs.
- For buffer overflows, *heap tracing* in `HEAP_TRACE_ALL` mode lets you see which callers are allocating which addresses from the heap. See *Heap Tracing To Find Heap Corruption* for more details. If you can find the function which allocates memory with an address immediately before the address which is corrupted, this will probably be the function which overflows the buffer.
- Calling `heap_caps_dump()` or `heap_caps_dump_all()` can give an indication of what heap blocks are surrounding the corrupted region and may have overflowed/underflowed/etc.

Configuration Temporarily increasing the heap corruption detection level can give more detailed information about heap corruption errors.

In the project configuration menu, under `Component config` there is a menu `Heap memory debugging`. The setting *CONFIG_HEAP_CORRUPTION_DETECTION* can be set to one of three levels:

Basic (no poisoning) This is the default level. No special heap corruption features are enabled, but provided assertions are enabled (the default configuration) then a heap corruption error will be printed if any of the heap's internal data structures appear overwritten or corrupted. This usually indicates a buffer overrun or out of bounds write.

If assertions are enabled, an assertion will also trigger if a double-free occurs (the same memory is freed twice).

Calling `heap_caps_check_integrity()` in Basic mode will check the integrity of all heap structures, and print errors if any appear to be corrupted.

Light Impact At this level, heap memory is additionally "poisoned" with head and tail "canary bytes" before and after each block which is allocated. If an application writes outside the bounds of allocated buffers, the canary bytes will be corrupted and the integrity check will fail.

The head canary word is 0xABBA1234 (3412BAAB in byte order), and the tail canary word is 0xBAAD5678 (7856ADBA in byte order).

"Basic" heap corruption checks can also detect most out of bounds writes, but this setting is more precise as even a single byte overrun can be detected. With Basic heap checks, the number of overrun bytes before a failure is detected will depend on the properties of the heap.

Enabling "Light Impact" checking increases memory usage, each individual allocation will use 9 to 12 additional bytes of memory (depending on alignment).

Each time `free()` is called in Light Impact mode, the head and tail canary bytes of the buffer being freed are checked against the expected values.

When *`heap_caps_check_integrity()`* is called, all allocated blocks of heap memory have their canary bytes checked against the expected values.

In both cases, the check is that the first 4 bytes of an allocated block (before the buffer returned to the user) should be the word 0xABBA1234. Then the last 4 bytes of the allocated block (after the buffer returned to the user) should be the word 0xBAAD5678.

Different values usually indicate buffer underrun or overrun, respectively.

Comprehensive This level incorporates the "light impact" detection features plus additional checks for uninitialised-access and use-after-free bugs. In this mode, all freshly allocated memory is filled with the pattern 0xCE, and all freed memory is filled with the pattern 0xFE.

Enabling "Comprehensive" detection has a substantial runtime performance impact (as all memory needs to be set to the allocation patterns each time a malloc/free completes, and the memory also needs to be checked each time.) However, it allows easier detection of memory corruption bugs which are much more subtle to find otherwise. It is recommended to only enable this mode when debugging, not in production.

Crashes in Comprehensive Mode If an application crashes reading/writing an address related to 0xCECECECE in Comprehensive mode, this indicates it has read uninitialized memory. The application should be changed to either use calloc() (which zeroes memory), or initialize the memory before using it. The value 0xCECECECE may also be seen in stack-allocated automatic variables, because in IDF most task stacks are originally allocated from the heap and in C stack memory is uninitialized by default.

If an application crashes and the exception register dump indicates that some addresses or values were 0xFEFEFEFE, this indicates it is reading heap memory after it has been freed (a "use after free bug".) The application should be changed to not access heap memory after it has been freed.

If a call to malloc() or realloc() causes a crash because it expected to find the pattern 0xFEFEFEFE in free memory and a different pattern was found, then this indicates the app has a use-after-free bug where it is writing to memory which has already been freed.

Manual Heap Checks in Comprehensive Mode Calls to *`heap_caps_check_integrity()`* may print errors relating to 0xFEFEFEFE, 0xABBA1234 or 0xBAAD5678. In each case the checker is expecting to find a given pattern, and will error out if this is not found:

- For free heap blocks, the checker expects to find all bytes set to 0xFE. Any other values indicate a use-after-free bug where free memory has been incorrectly overwritten.
- For allocated heap blocks, the behaviour is the same as for *Light Impact* mode. The canary bytes 0xABBA1234 and 0xBAAD5678 are checked at the head and tail of each allocated buffer, and any variation indicates a buffer overrun/underrun.

Heap Task Tracking

Heap Task Tracking can be used to get per task info for heap memory allocation. Application has to specify the heap capabilities for which the heap allocation is to be tracked.

Example code is provided in system/heap_task_tracking

Heap Tracing

Heap Tracing allows tracing of code which allocates/frees memory. Two tracing modes are supported:

- Standalone. In this mode trace data are kept on-board, so the size of gathered information is limited by the buffer assigned for that purposes. Analysis is done by the on-board code. There are a couple of APIs available for accessing and dumping collected info.
- Host-based. This mode does not have the limitation of the standalone mode, because trace data are sent to the host over JTAG connection using app_trace library. Later on they can be analysed using special tools.

Heap tracing can perform two functions:

- Leak checking: find memory which is allocated and never freed.
- Heap use analysis: show all functions that are allocating/freeing memory while the trace is running.

How To Diagnose Memory Leaks If you suspect a memory leak, the first step is to figure out which part of the program is leaking memory. Use the `xPortGetFreeHeapSize()`, *`heap_caps_get_free_size()`*, or *related functions* to track memory use over the life of the application. Try to narrow the leak down to a single function or sequence of functions where free memory always decreases and never recovers.

Standalone Mode Once you've identified the code which you think is leaking:

- In the project configuration menu, navigate to `Component settings` -> `Heap Memory Debugging` -> `Heap tracing` and select `Standalone` option (see *CONFIG_HEAP_TRACING_DEST*).
- Call the function *`heap_trace_init_standalone()`* early in the program, to register a buffer which can be used to record the memory trace.
- Call the function *`heap_trace_start()`* to begin recording all mallocs/frees in the system. Call this immediately before the piece of code which you suspect is leaking memory.
- Call the function *`heap_trace_stop()`* to stop the trace once the suspect piece of code has finished executing.
- Call the function *`heap_trace_dump()`* to dump the results of the heap trace.

An example:

```
#include "esp_heap_trace.h"

#define NUM_RECORDS 100
static heap_trace_record_t trace_record[NUM_RECORDS]; // This buffer must be in␣
↪internal RAM

...

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();
    ...
}
```

The output from the heap trace will look something like this:

```
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
```

(Above example output is using *IDF Monitor* to automatically decode PC addresses to their source files & line number.)

The first line indicates how many allocation entries are in the buffer, compared to its total size.

In `HEAP_TRACE_LEAKS` mode, for each traced memory allocation which has not already been freed a line is printed with:

- `XX bytes` is the number of bytes allocated
- `@ 0x...` is the heap address returned from malloc/calloc.
- `Internal` or `PSRAM` is the general location of the allocated memory.
- `CPU x` is the CPU (0 or 1) running when the allocation was made.
- `ccount 0x...` is the CCOUNT (CPU cycle count) register value when the allocation was mode. Is different for CPU 0 vs CPU 1.

Finally, the total number of 'leaked' bytes (bytes allocated but not freed while trace was running) is printed, and the total number of allocations this represents.

A warning will be printed if the trace buffer was not large enough to hold all the allocations which happened. If you see this warning, consider either shortening the tracing period or increasing the number of records in the trace buffer.

Host-Based Mode Once you've identified the code which you think is leaking:

- In the project configuration menu, navigate to `Component settings` -> `Heap Memory Debugging` -> *CONFIG_HEAP_TRACING_DEST* and select `Host-Based`.
- In the project configuration menu, navigate to `Component settings` -> `Application Level Tracing` -> *CONFIG_APPTRACE_DESTINATION1* and select `Trace memory`.
- In the project configuration menu, navigate to `Component settings` -> `Application Level Tracing` -> `FreeRTOS SystemView Tracing` and enable *CONFIG_APPTRACE_SV_ENABLE*.
- Call the function *`heap_trace_init_tohost()`* early in the program, to initialize JTAG heap tracing module.
- Call the function *`heap_trace_start()`* to begin recording all mallocs/frees in the system. Call this immediately before the piece of code which you suspect is leaking memory. In host-based mode, the argument to this function is ignored, and the heap tracing module behaves like `HEAP_TRACE_ALL` was passed: all allocations and deallocations are sent to the host.
- Call the function *`heap_trace_stop()`* to stop the trace once the suspect piece of code has finished executing.

An example:

```
#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) );
```

(下页继续)

(续上页)

```
    do_something_you_suspect_is_leaking();

    ESP_ERROR_CHECK( heap_trace_stop() );
    ...
}
```

To gather and analyse heap trace do the following on the host:

1. Build the program and download it to the target as described in *Getting Started Guide*.
2. Run OpenOCD (see *JTAG Debugging*).

备注： In order to use this feature you need OpenOCD version *v0.10.0-esp32-20181105* or later.

3. You can use GDB to start and/or stop tracing automatically. To do this you need to prepare special `gdbinit` file:

```
target remote :3333

mon reset halt
flushregs

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
```

Using this file GDB will connect to the target, reset it, and start tracing when program hits breakpoint at *`heap_trace_start()`*. Trace data will be saved to `/tmp/heap_log.svdat`. Tracing will be stopped when program hits breakpoint at *`heap_trace_stop()`*.

4. Run GDB using the following command `riscv32-esp-elf-gdb -x gdbinit </path/to/program/elf>`
5. Quit GDB when program stops at *`heap_trace_stop()`*. Trace data are saved in `/tmp/heap.svdat`
6. Run processing script `$IDF_PATH/tools/esp_app_trace/sysviewtrace_proc.py -p -b </path/to/program/elf> /tmp/heap_log.svdat`

The output from the heap trace will look something like this:

```
Parse trace from '/tmp/heap.svdat'...
Stop parsing trace. (Timeout 0.000000 sec while reading 1 bytes!)
Process events from '['/tmp/heap.svdat']'...
[0.002244575] HEAP: Allocated 1 bytes @ 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 task "free" 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:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.002782950] HEAP: Freed bytes @ 0x3ffb40b8 from 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 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

[0.102436025] 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:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.102449800] HEAP: Allocated 4 bytes @ 0x3ffaffe8 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)

[0.102666150] HEAP: Freed bytes @ 0x3ffaffe8 from 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 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 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 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 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 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 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.
```

(下页继续)

(续上页)

```
[0.002244575] HEAP: Allocated 1 bytes @ 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)

[0.102436025] 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:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202436200] HEAP: Allocated 3 bytes @ 0x3ffaffe8 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)

[0.302436000] HEAP: Allocated 4 bytes @ 0x3ffafff0 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)

Found 10 leaked bytes in 4 blocks.
```

Heap Tracing To Find Heap Corruption Heap tracing can also be used to help track down heap corruption. When a region in heap is corrupted, it may be from some other part of the program which allocated memory at a nearby address.

If you have some idea at what time the corruption occurred, enabling heap tracing in `HEAP_TRACE_ALL` mode allows you to record all the functions which allocated memory, and the addresses of the allocations.

Using heap tracing in this way is very similar to memory leak detection as described above. For memory which is allocated and not freed, the output is the same. However, records will also be shown for memory which has been freed.

Performance Impact Enabling heap tracing in menuconfig increases the code size of your program, and has a very small negative impact on performance of heap allocation/free operations even when heap tracing is not running.

When heap tracing is running, heap allocation/free operations are substantially slower than when heap tracing is stopped. Increasing the depth of stack frames recorded for each allocation (see above) will also increase this performance impact.

False-Positive Memory Leaks Not everything printed by *`heap_trace_dump()`* is necessarily a memory leak. Among things which may show up here, but are not memory leaks:

- Any memory which is allocated after *`heap_trace_start()`* but then freed after *`heap_trace_stop()`* will appear in the leak dump.
- Allocations may be made by other tasks in the system. Depending on the timing of these tasks, it's quite possible this memory is freed after *`heap_trace_stop()`* is called.
- The first time a task uses stdio - for example, when it calls `printf()` - a lock (RTOS mutex semaphore) is allocated by the libc. This allocation lasts until the task is deleted.
- Certain uses of `printf()`, such as printing floating point numbers, will allocate some memory from the heap on demand. These allocations last until the task is deleted.
- The Bluetooth, Wi-Fi, and TCP/IP libraries will allocate heap memory buffers to handle incoming or outgoing data. These memory buffers are usually short-lived, but some may be shown in the heap leak trace if the data was received/transmitted by the lower levels of the network while the leak trace was running.
- TCP connections will continue to use some memory after they are closed, because of the `TIME_WAIT` state. After the `TIME_WAIT` period has completed, this memory will be freed.

One way to differentiate between "real" and "false positive" memory leaks is to call the suspect code multiple times while tracing is running, and look for patterns (multiple matching allocations) in the heap trace output.

API Reference - Heap Tracing

Header File

- components/heap/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**

2.10.16 High Resolution Timer (ESP Timer)

Overview

Although FreeRTOS provides software timers, these timers have a few limitations:

- Maximum resolution is equal to RTOS tick period
- Timer callbacks are dispatched from a low-priority task

Hardware timers are free from both of the limitations, but often they are less convenient to use. For example, application components may need timer events to fire at certain times in the future, but the hardware timer only contains one "compare" value used for interrupt generation. This means that some facility needs to be built on top of the hardware timer to manage the list of pending events can dispatch the callbacks for these events as corresponding hardware interrupts happen.

An interrupt level of the handler depends on the *CONFIG_ESP_TIMER_INTERRUPT_LEVEL* option. It allows to set this: 1, 2 or 3 level (by default 1). Raising the level, the interrupt handler can reduce the timer processing delay.

`esp_timer` set of APIs provides one-shot and periodic timers, microsecond time resolution, and 52-bit range.

Internally, `esp_timer` uses a 52-bit hardware timer, where the implementation depends on the target. SYSTIMER is used for ESP32-C2.

Timer callbacks can be dispatched by two methods:

- `ESP_TIMER_TASK`
- `ESP_TIMER_ISR`. Available only if *CONFIG_ESP_TIMER_SUPPORTS_ISR_DISPATCH_METHOD* is enabled (by default disabled).

`ESP_TIMER_TASK`. Timer callbacks are dispatched from a high-priority `esp_timer` task. Because all the callbacks are dispatched from the same task, it is recommended to only do the minimal possible amount of work from the callback itself, posting an event to a lower priority task using a queue instead.

If other tasks with priority higher than `esp_timer` are running, callback dispatching will be delayed until `esp_timer` task has a chance to run. For example, this will happen if an SPI Flash operation is in progress.

`ESP_TIMER_ISR`. Timer callbacks are dispatched directly from the timer interrupt handler. This method is useful for some simple callbacks which aim for lower latency.

Creating and starting a timer, and dispatching the callback takes some time. Therefore, there is a lower limit to the timeout value of one-shot `esp_timer`. If *`esp_timer_start_once()`* is called with a timeout value less than 20us, the callback will be dispatched only after approximately 20us.

Periodic esp_timer also imposes a 50us restriction on the minimal timer period. Periodic software timers with period of less than 50us are not practical since they would consume most of the CPU time. Consider using dedicated hardware peripherals or DMA features if you find that a timer with small period is required.

Using esp_timer APIs

Single timer is represented by *esp_timer_handle_t* type. Timer has a callback function associated with it. This callback function is called from the esp_timer task each time the timer elapses.

- To create a timer, call *esp_timer_create()*.
- To delete the timer when it is no longer needed, call *esp_timer_delete()*.

The timer can be started in one-shot mode or in periodic mode.

- To start the timer in one-shot mode, call *esp_timer_start_once()*, passing the time interval after which the callback should be called. When the callback gets called, the timer is considered to be stopped.
- To start the timer in periodic mode, call *esp_timer_start_periodic()*, passing the period with which the callback should be called. The timer keeps running until *esp_timer_stop()* is called.

Note that the timer must not be running when *esp_timer_start_once()* or *esp_timer_start_periodic()* is called. To restart a running timer, call *esp_timer_stop()* first, then call one of the start functions.

Callback functions

备注： Keep the callback functions as short as possible otherwise it will affect all timers.

Timer callbacks which are processed by ESP_TIMER_ISR method should not call the context switch call - portYIELD_FROM_ISR(), instead of this you should use the *esp_timer_isr_dispatch_need_yield()* function. The context switch will be done after all ISR dispatch timers have been processed, if required by the system.

esp_timer during light sleep

During light sleep, the esp_timer counter stops and no callback functions are called. Instead, the time is counted by the RTC counter. Upon waking up, the system gets the difference between the counters and calls a function that advances the esp_timer counter. Since the counter has been advanced, the system starts calling callbacks that were not called during sleep. The number of callbacks depends on the duration of the sleep and the period of the timers. It can lead to overflow of some queues. This only applies to periodic timers, one-shot timers will be called once.

This behavior can be changed by calling *esp_timer_stop()* before sleeping. In some cases, this can be inconvenient, and instead of the stop function, you can use the *skip_unhandled_events* option during *esp_timer_create()*. When the *skip_unhandled_events* is true, if a periodic timer expires one or more times during light sleep then only one callback is called on wake.

Using the *skip_unhandled_events* option with *automatic light sleep* (see *Power Management APIs*) helps to reduce the power consumption of the system when it is in light sleep. The duration of light sleep is also determined by esp_timers. Timers with *skip_unhandled_events* option will not wake up the system.

Handling callbacks

esp_timer is designed to achieve a high-resolution low latency timer and the ability to handle delayed events. If the timer is late then the callback will be called as soon as possible, it will not be lost. In the worst case, when the timer has not been processed for more than one period (for periodic timers), the callbacks will be called one after the other without waiting for the set period. This can be bad for some applications, and the *skip_unhandled_events* option was introduced to eliminate this behavior. If *skip_unhandled_events* is set then a periodic timer that has expired multiple times without being able to call the callback will still result in only one callback event once processing is possible.

Obtaining Current Time

`esp_timer` also provides a convenience function to obtain the time passed since start-up, with microsecond precision: *`esp_timer_get_time()`*. This function returns the number of microseconds since `esp_timer` was initialized, which usually happens shortly before `app_main` function is called.

Unlike *gettimeofday* function, values returned by *`esp_timer_get_time()`*:

- Start from zero after the chip wakes up from deep sleep
- Do not have timezone or DST adjustments applied

Application Example

The following example illustrates usage of `esp_timer` APIs: system/esp_timer.

API Reference

Header File

- components/esp_timer/include/esp_timer.h

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 CONFIG_ESP_TIMER_ISR_AFFINITY_NO_AFFINITY is enabled, 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 done using the timer, delete it with esp_timer_delete function.

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 which will hold 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 one-shot timer.

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 should not be running when this function is called. This function will start 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.

If the given timer is a one-shot timer, the timer is restarted immediately and will timeout once in `timeout_us` microseconds. If the given timer is a periodic timer, the timer is 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 the 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 allocated 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 underlying timer has been started

int64_t **esp_timer_get_next_alarm**(void)

Get the timestamp when the next timeout is expected to occur.

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 when the next timeout is expected to occur skipping those which have skip_unhandled_events flag.

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.

备注: The timeout period is the time interval with which a timer restarts after expiry. For one-shot timers, the period is 0 as there is no periodicity associated with such timers.

参数

- **timer** –timer handle allocated 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: This API returns a valid expiry time only for a one-shot timer. It returns an error if the timer handle passed to the function is for a periodic timer.

参数

- **timer** –timer handle allocated 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.

If CONFIG_ESP_TIMER_PROFILING option is enabled, this prints the list of all the existing timers. Otherwise, only the list active timers is printed.

The format is:

name period alarm times_armed times_triggered total_callback_run_time

where:

name —timer name (if CONFIG_ESP_TIMER_PROFILING is defined), or 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

The following fields are printed if CONFIG_ESP_TIMER_PROFILING is defined:

times_armed —number of times the timer was armed via esp_timer_start_X times_triggered - number of times the callback was called total_callback_run_time - total time taken by callback to execute, across all calls

参数 **stream** –stream (such as stdout) to dump the information to

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 or not.

参数 **timer** –timer handle created using esp_timer_create

返回

- 1 if timer is still active
- 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 by calling `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**

Function to call when timer expires.

void ***arg**

Argument to pass to the callback.

esp_timer_dispatch_t **dispatch_method**

Call the callback from task or from ISR.

const char ***name**

Timer name, used in esp_timer_dump function.

bool **skip_unhandled_events**

Skip unhandled events for periodic timers.

Type Definitions

typedef struct esp_timer ***esp_timer_handle_t**

Opaque type representing a single esp_timer.

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 for dispatching timer callback.

Values:

enumerator **ESP_TIMER_TASK**

Callback is called from timer task.

enumerator **ESP_TIMER_ISR**

Callback is called from timer ISR.

enumerator **ESP_TIMER_MAX**

Count of the methods for dispatching timer callback.

2.10.17 Internal and Unstable APIs

This section is listing some APIs that are internal or likely to be changed or removed in the next releases of ESP-IDF.

API Reference

Header File

- components/esp_rom/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_uart_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 interupt
d. Enable peripheral interrupt

参数

- **cpu_core** –The CPU number, which the peripheral interupt will inform to
- **periph_intr_id** –The peripheral interrupt source number
- **cpu_intr_num** –The CPU interrupt number

uint32_t **esp_rom_get_cpu_ticks_per_us**(void)

Get the real CPU ticks per us.

返回 CPU ticks per us

2.10.18 Interrupt allocation

Overview

The ESP32-C2 has one core, with 31 interrupts. Each interrupt has a programmable priority level.

Because there are more interrupt sources than interrupts, sometimes it makes sense to share an interrupt in multiple drivers. The *esp_intr_alloc()* abstraction exists to hide all these implementation details.

A driver can allocate an interrupt for a certain peripheral by calling *esp_intr_alloc()* (or *esp_intr_alloc_intrstatus()*). It can use the flags passed to this function to set the type of interrupt allocated, specifying a particular level or trigger method. The interrupt allocation code will then find an applicable interrupt, use the interrupt mux to hook it up to the peripheral, and install the given interrupt handler and ISR to it.

This code presents two different types of interrupts, handled differently: shared interrupts and non-shared interrupts. The simplest ones are non-shared interrupts: a separate interrupt is allocated per *esp_intr_alloc()* call and this interrupt is solely used for the peripheral attached to it, with only one ISR that will get called. On the other hand, shared interrupts can have multiple peripherals triggering them, with multiple ISRs being called when one of the peripherals attached signals an interrupt. Thus, ISRs that are intended for shared interrupts should check the interrupt status of the peripheral they service in order to check if any action is required.

Non-shared interrupts can be either level- or edge-triggered. Shared interrupts can only be level interrupts due to the chance of missed interrupts when edge interrupts are used.

For example, let's say DevA and DevB share an interrupt. DevB signals an interrupt, so INT line goes high. The ISR handler calls code for DevA but does nothing. Then, ISR handler calls code for DevB, but while doing that, DevA signals an interrupt. DevB's ISR is done, it clears interrupt status for DevB and exits interrupt code. Now, an interrupt for DevA is still pending, but because the INT line never went low, as DevA kept it high even when the interrupt for DevB was cleared, the interrupt is never serviced.

IRAM-Safe Interrupt Handlers

The `ESP_INTR_FLAG_IRAM` flag registers an interrupt handler that always runs from IRAM (and reads all its data from DRAM), and therefore does not need to be disabled during flash erase and write operations.

This is useful for interrupts which need a guaranteed minimum execution latency, as flash write and erase operations can be slow (erases can take tens or hundreds of milliseconds to complete).

It can also be useful to keep an interrupt handler in IRAM if it is called very frequently, to avoid flash cache misses.

Refer to the *SPI flash API documentation* for more details.

Multiple Handlers Sharing A Source

Several handlers can be assigned to a same source, given that all handlers are allocated using the `ESP_INTR_FLAG_SHARED` flag. They will all be allocated to the interrupt, which the source is attached to, and called sequentially when the source is active. The handlers can be disabled and freed individually. The source is attached to the interrupt (enabled), if one or more handlers are enabled, otherwise detached. A handler will never be called when disabled, while **its source may still be triggered** if any one of its handler enabled.

Sources attached to non-shared interrupt do not support this feature.

Though the framework support this feature, you have to use it *very carefully*. There usually exist two ways to stop an interrupt from being triggered: *disable the source* or *mask peripheral interrupt status*. IDF only handles enabling and disabling of the source itself, leaving status and mask bits to be handled by users. **Status bits shall either be masked before the handler responsible for it is disabled, either be masked and then properly handled in another enabled interrupt**. Please note that leaving some status bits unhandled without masking them, while disabling the handlers for them, will cause the interrupt(s) to be triggered indefinitely, resulting therefore in a system crash.

API Reference

Header File

- components/esp_hw_support/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

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

Type Definitions

typedef void (***intr_handler_t**)(void *arg)

Function prototype for interrupt handler function

typedef struct *intr_handle_data_t* **intr_handle_data_t**

Interrupt handler associated data structure

typedef *intr_handle_data_t* ***intr_handle_t**

Handle to an interrupt handler

2.10.19 Logging library

Overview

The logging library provides two ways for setting log verbosity:

- **At compile time**: in menuconfig, set the verbosity level using the option *CONFIG_LOG_DEFAULT_LEVEL*.
- Optionally, also in menuconfig, set the maximum verbosity level using the option *CONFIG_LOG_MAXIMUM_LEVEL*. By default this is the same as the default level, but it can be set higher in order to compile more optional logs into the firmware.
- **At runtime**: all logs for verbosity levels lower than *CONFIG_LOG_DEFAULT_LEVEL* are enabled by default. The function `esp_log_level_set()` can be used to set a logging level on a per module basis. Modules are identified by their tags, which are human-readable ASCII zero-terminated strings.

There are the following verbosity levels:

- Error (lowest)
- Warning
- Info
- Debug
- Verbose (highest)

备注: The function `esp_log_level_set()` cannot set logging levels higher than specified by *CONFIG_LOG_MAXIMUM_LEVEL*. To increase log level for a specific file above this maximum at compile time, use the macro *LOG_LOCAL_LEVEL* (see the details below).

How to use this library

In each C file that uses logging functionality, define the TAG variable as shown below:

```
static const char* TAG = "MyModule";
```

Then use one of logging macros to produce output, e.g:

```
ESP_LOGW(TAG, "Baud rate error %.1f%%. Requested: %d baud, actual: %d baud", error␣
↪* 100, baud_req, baud_real);
```

Several macros are available for different verbosity levels:

- `ESP_LOGE` - error (lowest)
- `ESP_LOGW` - warning
- `ESP_LOGI` - info
- `ESP_LOGD` - debug
- `ESP_LOGV` - verbose (highest)

Additionally, there are ESP_EARLY_LOGx versions for each of these macros, e.g. *ESP_EARLY_LOGE*. These versions have to be used explicitly in the early startup code only, before heap allocator and syscalls have been initialized. Normal ESP_LOGx macros can also be used while compiling the bootloader, but they will fall back to the same implementation as ESP_EARLY_LOGx macros.

There are also ESP_DRAM_LOGx versions for each of these macros, e.g. *ESP_DRAM_LOGE*. These versions are used in some places where logging may occur with interrupts disabled or with flash cache inaccessible. Use of this macros should be as sparing as possible, as logging in these types of code should be avoided for performance reasons.

备注： Inside critical sections interrupts are disabled so it's only possible to use ESP_DRAM_LOGx (preferred) or ESP_EARLY_LOGx. Even though it's possible to log in these situations, it's better if your program can be structured not to require it.

To override default verbosity level at file or component scope, define the LOG_LOCAL_LEVEL macro.

At file scope, define it before including esp_log.h, e.g.:

```
#define LOG_LOCAL_LEVEL ESP_LOG_VERBOSE
#include "esp_log.h"
```

At component scope, define it in the component CMakeLists:

```
target_compile_definitions(${COMPONENT_LIB} PUBLIC "-DLOG_LOCAL_LEVEL=ESP_LOG_
↪VERBOSE")
```

To configure logging output per module at runtime, add calls to the function *esp_log_level_set()* as follows:

```
esp_log_level_set("*", ESP_LOG_ERROR);        // set all components to ERROR level
esp_log_level_set("wifi", ESP_LOG_WARN);      // enable WARN logs from WiFi stack
esp_log_level_set("dhcpc", ESP_LOG_INFO);     // enable INFO logs from DHCP client
```

备注： The "DRAM" and "EARLY" log macro variants documented above do not support per module setting of log verbosity. These macros will always log at the "default" verbosity level, which can only be changed at runtime by calling esp_log_level("*", level).

Logging to Host via JTAG By default, the logging library uses the vprintf-like function to write formatted output to the dedicated UART. By calling a simple API, all log output may be routed to JTAG instead, making logging several times faster. For details, please refer to Section 记录日志到主机.

Application Example

The logging library is commonly used by most esp-idf components and examples. For demonstration of log functionality, check ESP-IDF's examples directory. The most relevant examples that deal with logging are the following:

- system/ota
- storage/sd_card
- protocols/https_request

API Reference

Header File

- components/log/include/esp_log.h

Functions

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.

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", orESP_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.

2.10.20 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，CPU 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-C2 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。

备注： ESP32-C2 内部未集成以太网 MAC 地址，但仍可以计算得出该地址。不过，以太网 MAC 地址只能与外部以太网接口（如 SPI 以太网设备）一起使用，具体请参阅以太网。

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_ESP32C2_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-C2 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

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.

Header File

- components/esp_common/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

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

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_POSIX_LINUX**

The code is running on POSIX/Linux simulator.

Header File

- components/esp_hw_support/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_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.

参数

- **wp_num** –Hardware watchpoint number [0..SOC_CPU_WATCHPOINTS_NUM - 1]
- **wp_addr** –Watchpoint's base address
- **size** –Size of the region to watch. Must be one of 2^n, with n in [0..6].
- **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

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

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)

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

uint32_t **reserv2**[20]

reserv2

Macros

ESP_APP_DESC_MAGIC_WORD

The magic word for the esp_app_desc structure that is in DROM.

2.10.21 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 或蓝牙）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。

注解：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* 没有使能，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) {
```

(下页继续)

(续上页)

```
    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 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 partool.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
- 分区表 *API*
- 低层 *SPI Flash API*
- *ESP HTTPS OTA*

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

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.
- 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.

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.

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().

OTA 升级失败排查

2.10.22 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。

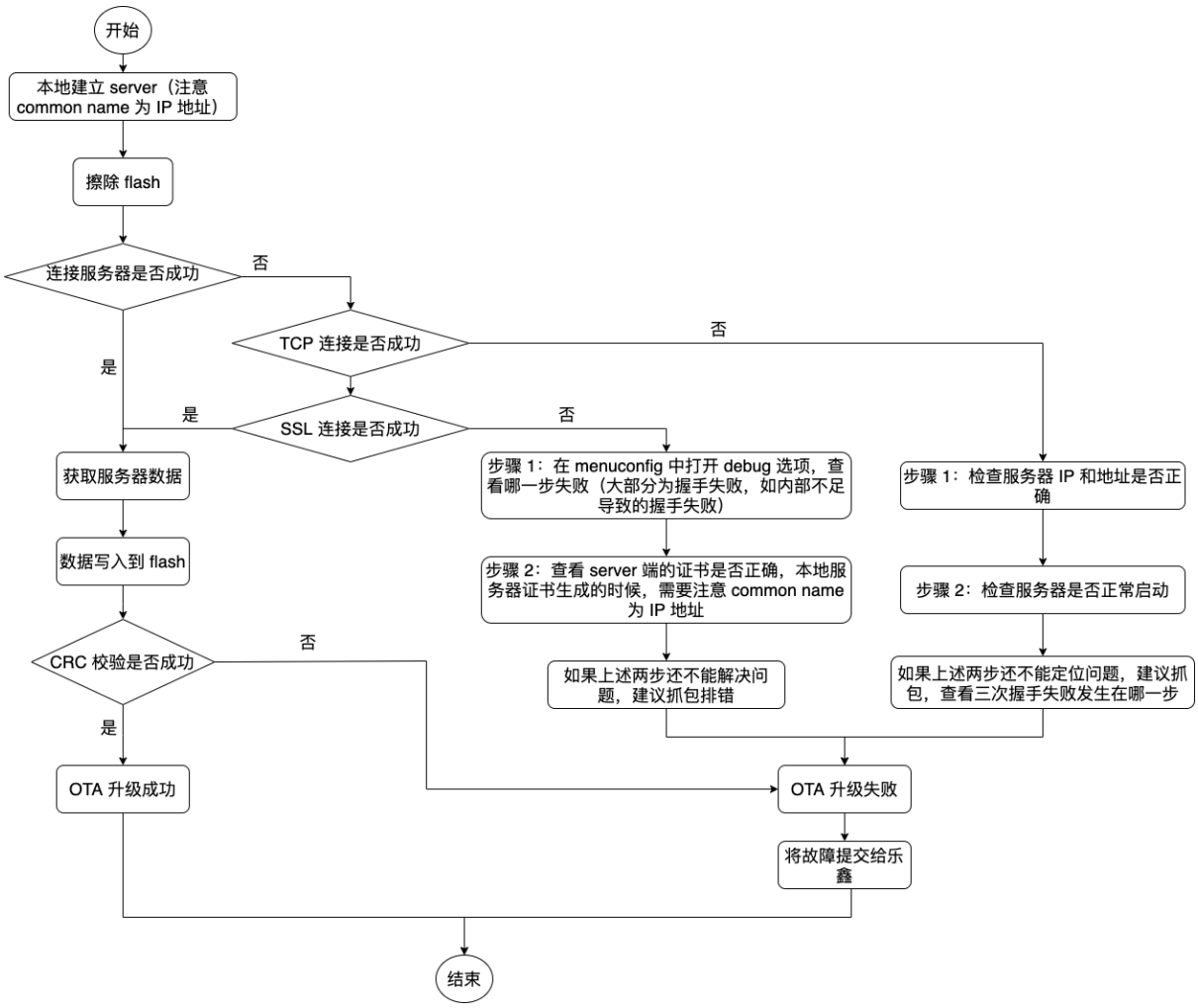


图 18: OTA 升级失败时如何排查（点击放大）

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)，即仅获取 `ESP_PM_APB_FREQ_MAX` 锁后所使用的频率。该字段可设置为晶振 (XTAL) 频率值，或者 XTAL 频率值除以整数。注意，10 MHz 是生成 1 MHz 的 REF_TICK 默认时钟所需的最小频率。
- `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。

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-C2 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-C2 可以将该值设置为 80 MHz 或 120 MHz。
`ESP_PM_APB_FREQ_MAX`	请求将 APB 频率设置为最大值，ESP32-C2 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-C2 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。您可以使用`esp_pm_configure()` 或者*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
120	获取 ESP_PM_CPU_FREQ_MAX	CPU: 120 MHz APB: 40 Mhz
	获取 ESP_PM_APB_FREQ_MAX，未获得 ESP_PM_CPU_FREQ_MAX	CPU: 80 MHz APB: 40 Mhz
	无	使用 *esp_pm_configure()* 为二者设置最小值
80	获取 ESP_PM_CPU_FREQ_MAX 或 ESP_PM_APB_FREQ_MAX	CPU: 80 MHz APB: 40 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

您也可以设置 Light-sleep 模式在最近事件（任务解除阻塞，或计时器超时）之前持续多久才唤醒芯片。

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。

以下外设不受 APB 频率变更的影响：

- **UART**：如果 REF_TICK 或者 XTAL 用作时钟源，则 UART 不受 APB 频率变更影响。请查看 *uart_config_t::source_clk*。
- **LEDC**：如果 REF_TICK 用作时钟源，则 LEDC 不受 APB 频率变更影响。请查看 *ledc_timer_config()* 函数。
- **RMT**：如果 REF_TICK 或者 XTAL 被用作时钟源，则 RMT 不受 APB 频率变更影响。请查看 rmt_config_t::flags 以及 *RMT_CHANNEL_FLAGS_AWARE_DFS* 宏。
- **GPTimer**：如果 XTAL 用作时钟源，则 GPTimer 不受 APB 频率变更影响。请查看 *gptimer_config_t::clk_src*。
- **TSENS**：XTAL 或 RTC_8M 用作时钟源，因此不受 APB 频率变化影响。

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。
- **Ethernet**：从调用 *esp_eth_driver_install()* 至 *esp_eth_driver_uninstall()* 期间。
- **WiFi**：从调用 *esp_wifi_start()* 至 *esp_wifi_stop()* 期间。如果启用了调制解调器睡眠模式，广播关闭时将释放此管理锁。
- **Bluetooth**：从调用 *esp_bt_controller_enable()* 至 *esp_bt_controller_disable()* 期间。如果启用了蓝牙调制解调器，广播关闭时将释放此管理锁。但依然占用 ESP_PM_NO_LIGHT_SLEEP 锁。

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：

- PCNT
- Sigma-delta
- 旧版定时器驱动（Timer Group）

Light-sleep 外设下电

API 参考

Header File

- components/esp_pm/include/esp_pm.h

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.

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 or type argument is not valid
- 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.

2.10.23 POSIX Threads Support

Overview

ESP-IDF is based on FreeRTOS but offers a range of POSIX-compatible APIs that allow easy porting of third party code. This includes support for common parts of the POSIX Threads "pthreads" API.

POSIX Threads are implemented in ESP-IDF as wrappers around equivalent FreeRTOS features. The runtime memory or performance overhead of using the pthreads API is quite low, but not every feature available in either pthreads or FreeRTOS is available via the ESP-IDF pthreads support.

Pthreads can be used in ESP-IDF by including standard `pthread.h` header, which is included in the toolchain libc. An additional ESP-IDF specific header, `esp_pthread.h`, provides additional non-POSIX APIs for using some ESP-IDF features with pthreads.

C++ Standard Library implementations for `std::thread`, `std::mutex`, `std::condition_variable`, etc. are implemented using pthreads (via GCC libstdc++). Therefore, restrictions mentioned here also apply to the equivalent C++ standard library functionality.

RTOS Integration

Unlike many operating systems using POSIX Threads, ESP-IDF is a real-time operating system with a real-time scheduler. This means that a thread will only stop running if a higher priority task is ready to run, the thread blocks on an OS synchronization structure like a mutex, or the thread calls any of the functions `sleep`, *`vTaskDelay()`*, or `usleep`.

备注: If calling a standard libc or C++ sleep function, such as `usleep` defined in `unistd.h`, then the task will only block and yield the CPU if the sleep time is longer than *one FreeRTOS tick period*. If the time is shorter, the thread will busy-wait instead of yielding to another RTOS task.

By default, all POSIX Threads have the same RTOS priority, but it is possible to change this by calling a *custom API*.

Standard features

The following standard APIs are implemented in ESP-IDF.

Refer to standard POSIX Threads documentation, or pthread.h, for details about the standard arguments and behaviour of each function. Differences or limitations compared to the standard APIs are noted below.

Thread APIs

- `pthread_create()` - The `attr` argument is supported for setting stack size and detach state only. Other attribute fields are ignored. - Unlike FreeRTOS task functions, the `start_routine` function is allowed to return. A "detached" type thread is automatically deleted if the function returns. The default "joinable" type thread will be suspended until pthread_join() is called on it.
- `pthread_join()`
- `pthread_detach()`
- `pthread_exit()`

- `sched_yield()`
- `pthread_self()` - An assert will fail if this function is called from a FreeRTOS task which is not a pthread.
- `pthread_equal()`

Thread Attributes

- `pthread_attr_init()`
- `pthread_attr_destroy()` - This function doesn't need to free any resources and instead resets the `attr` structure to defaults (implementation is same as `pthread_attr_init()`).
- `pthread_attr_getstacksize()` / `pthread_attr_setstacksize()`
- `pthread_attr_getdetachstate()` / `pthread_attr_setdetachstate()`

Once

- `pthread_once()`

Static initializer constant `PTHREAD_ONCE_INIT` is supported.

备注： This function can be called from tasks created using either pthread or FreeRTOS APIs

Mutexes POSIX Mutexes are implemented as FreeRTOS Mutex Semaphores (normal type for "fast" or "error check" mutexes, and Recursive type for "recursive" mutexes). This means that they have the same priority inheritance behaviour as mutexes created with `xSemaphoreCreateMutex()`.

- `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()`

Static initializer constant `PTHREAD_MUTEX_INITIALIZER` is supported, but the non-standard static initializer constants for other mutex types are not supported.

备注： These functions can be called from tasks created using either pthread or FreeRTOS APIs

Condition Variables

- `pthread_cond_init()` - The `attr` argument is not implemented and is ignored.
- `pthread_cond_destroy()`
- `pthread_cond_signal()`
- `pthread_cond_broadcast()`
- `pthread_cond_wait()`
- `pthread_cond_timedwait()`

Static initializer constant `PTHREAD_COND_INITIALIZER` is supported.

- The resolution of `pthread_cond_timedwait()` timeouts is the RTOS tick period (see *CONFIG_FREERTOS_HZ*). Timeouts may be delayed up to one tick period after the requested timeout.

备注： These functions can be called from tasks created using either pthread or FreeRTOS APIs

Semaphores In IDF, POSIX *unnamed* semaphores are implemented. The accessible API is described below. It implements semaphores as specified in the POSIX standard, unless specified otherwise.

- sem_init()
- sem_destroy()
  - `pshared` is ignored. Semaphores can always be shared between FreeRTOS tasks.
- sem_post()
  - If the semaphore has a value of `SEM_VALUE_MAX` already, -1 is returned and `errno` is set to `EAGAIN`.
- sem_wait()
- sem_trywait()
- sem_timedwait()
  - The time value passed by abstime will be rounded up to the next FreeRTOS tick.
  - The actual timeout will happen after the tick the time was rounded to and before the following tick.
  - It is possible, though unlikely, that the task is preempted directly after the timeout calculation, delaying the timeout of the following blocking operating system call by the duration of the preemption.
- sem_getvalue()

Read/Write Locks

- `pthread_rwlock_init()` - The `attr` argument is not implemented and is ignored.
- `pthread_rwlock_destroy()`
- `pthread_rwlock_rdlock()`
- `pthread_rwlock_wrlock()`
- `pthread_rwlock_unlock()`

Static initializer constant `PTHREAD_RWLOCK_INITIALIZER` is supported.

备注： These functions can be called from tasks created using either pthread or FreeRTOS APIs

---

Thread-Specific Data

- `pthread_key_create()` - The `destr_function` argument is supported and will be called if a thread function exits normally, calls `pthread_exit()`, or if the underlying task is deleted directly using the FreeRTOS function *`vTaskDelete()`*.
- `pthread_key_delete()`
- `pthread_setspecific()` / `pthread_getspecific()`

备注： These functions can be called from tasks created using either pthread or FreeRTOS APIs. When calling these functions from tasks created using FreeRTOS APIs, *CONFIG_FREERTOS_TLSP_DELETION_CALLBACKS* config option must be enabled to ensure the thread-specific data is cleaned up before the task is deleted.

备注： There are other options for thread local storage in ESP-IDF, including options with higher performance. See *Thread Local Storage*.

Not Implemented

The `pthread.h` header is a standard header and includes additional APIs and features which are not implemented in ESP-IDF. These include:

- `pthread_cancel()` returns `ENOSYS` if called.
- `pthread_condattr_init()` returns `ENOSYS` if called.

Other POSIX Threads functions (not listed here) are not implemented and will produce either a compiler or a linker error if referenced from an ESP-IDF application. If you identify a useful API that you would like to see implemented in ESP-IDF, please open a *feature request on GitHub <https://github.com/espressif/esp-idf/issues>* with the details.

ESP-IDF Extensions

The API *esp_pthread_set_cfg()* defined in the esp_pthreads.h header offers custom extensions to control how subsequent calls to pthread_create() will behave. Currently, the following configuration can be set:

- Default stack size of new threads, if not specified when calling pthread_create() (overrides *CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT*).
- RTOS priority of new threads (overrides *CONFIG_PTHREAD_TASK_PRIO_DEFAULT*).
- FreeRTOS task name for new threads (overrides *CONFIG_PTHREAD_TASK_NAME_DEFAULT*)

This configuration is scoped to the calling thread (or FreeRTOS task), meaning that *esp_pthread_set_cfg()* can be called independently in different threads or tasks. If the inherit_cfg flag is set in the current configuration then any new thread created will inherit the creator's configuration (if that thread calls pthread_create() recursively), otherwise the new thread will have the default configuration.

Examples

- system/pthread demonstrates using the pthreads API to create threads
- cxx/pthread demonstrates using C++ Standard Library functions with threads

API Reference

Header File

- components/pthread/include/esp_pthread.h

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: 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_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.

Macros

PTHREAD_STACK_MIN

2.10.24 Random Number Generation

ESP32-C2 contains a hardware random number generator, values from it can be obtained using the APIs *esp_random()* and *esp_fill_random()*.

The hardware RNG produces true random numbers under any of the following conditions:

- RF subsystem is enabled (i.e. Wi-Fi or Bluetooth are enabled).
- An internal entropy source has been enabled by calling *bootloader_random_enable()* and not yet disabled by calling *bootloader_random_disable()*.
- While the ESP-IDF *二级引导程序* is running. This is because the default ESP-IDF bootloader implementation calls *bootloader_random_enable()* when the bootloader starts, and *bootloader_random_disable()* before executing the app.

When any of these conditions are true, samples of physical noise are continuously mixed into the internal hardware RNG state to provide entropy. Consult the *ESP32-C2 Technical Reference Manual* > *Random Number Generator (RNG)* [PDF] chapter for more details.

If none of the above conditions are true, the output of the RNG should be considered pseudo-random only.

Startup

During startup, ESP-IDF bootloader temporarily enables a non-RF entropy source (internal reference voltage noise) that provides entropy for any first boot key generation. However, after the app starts executing then normally only pseudo-random numbers are available until Wi-Fi or Bluetooth are initialized.

To re-enable the entropy source temporarily during app startup, or for an application that does not use Wi-Fi or Bluetooth, call the function `bootloader_random_enable()` to re-enable the internal entropy source. The function `bootloader_random_disable()` must be called to disable the entropy source again before using ADC, Wi-Fi or Bluetooth.

备注： The entropy source enabled during the boot process by the ESP-IDF Second Stage Bootloader will seed the internal RNG state with some entropy. However, the internal hardware RNG state is not large enough to provide a continuous stream of true random numbers. This is why a continuous entropy source must be enabled whenever true random numbers are required.

备注： If an application requires a source of true random numbers but it is not possible to permanently enable a hardware entropy source, consider using a strong software DRBG implementation such as the mbedTLS CTR-DRBG or HMAC-DRBG, with an initial seed of entropy from hardware RNG true random numbers.

Secondary Entropy

ESP32-C2 RNG contains a secondary entropy source, based on sampling an asynchronous 8MHz internal oscillator (see the Technical Reference Manual for details). This entropy source is always enabled in ESP-IDF and continuously mixed into the RNG state by hardware. In testing, this secondary entropy source was sufficient to pass the Dieharder random number test suite without the main entropy source enabled (test input was created by concatenating short samples from a continuously resetting ESP32-C2). However, it is currently only guaranteed that true random numbers will be produced when the main entropy source is also enabled as described above.

API Reference

Header File

- components/esp_hw_support/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

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

A compatible version of the Linux `getrandom()` function is also provided for ease of porting:

```
#include <sys/random.h>

ssize_t getrandom(void *buf, size_t buflen, unsigned int flags);
```

This function is implemented by calling *esp_fill_random()* internally.

The `flags` argument is ignored, this function is always non-blocking but the strength of any random numbers is dependent on the same conditions described above.

Return value is -1 (with `errno` set to `EFAULT`) if the `buf` argument is NULL, and equal to `buflen` otherwise.

2.10.25 睡眠模式

概述

ESP32-C2 具有 Light-sleep 和 Deep-sleep 两种睡眠节能模式。

在 Light-sleep 模式下，数字外设、CPU、以及大部分 RAM 都使用时钟门控，同时电源电压降低。退出该模式后，数字外设、CPU 和 RAM 恢复运行，内部状态保持不变。

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

Light-sleep 和 Deep-sleep 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。通过 API `esp_sleep_enable_X_wakeup` 可启用唤醒源，通过 API *`esp_sleep_disable_wakeup_source()`* 可禁用唤醒源，详见下一小节。在系统进入 Light-sleep 或 Deep-sleep 模式前，可以在任意时刻配置唤醒源。

此外，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。

配置唤醒源后，应用程序就可以使用 API *`esp_light_sleep_start()`* 或*`esp_deep_sleep_start()`* 进入睡眠模式。此时，系统将按照被请求的唤醒源配置硬件，同时 RTC 控制器会给 CPU 和数字外设断电。

如需保持 Wi-Fi 连接，请启用 Wi-Fi Modem-sleep 模式和自动 Light-sleep 模式（请参阅*电源管理 API*）。在这两种模式下，Wi-Fi 驱动程序发出请求时，系统将自动从睡眠中被唤醒，从而保持与 AP 的连接。

睡眠模式下的 Wi-Fi 和 Bluetooth 功能

在 Light-sleep 和 Deep-sleep 模式下，无线外设会被断电。因此，在进入这两种睡眠模式前，应用程序必须调用恰当的函数（*`esp_bluedroid_disable()`*、*`esp_bt_controller_disable()`* 或*`esp_wifi_stop()`*）来禁用 Wi-Fi 和 Bluetooth。在 Light-sleep 或 Deep-sleep 模式下，即使不调用这些函数也无法连接 Wi-Fi 和 Bluetooth。

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。

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。

GPIO 唤醒 任何一个 IO 都可以用作外部输入管脚，将芯片从 Light-sleep 状态唤醒。调用*`gpio_wakeup_enable()`* 函数可以将任意管脚单独配置为在高电平或低电平触发唤醒。此后，应调用*`esp_sleep_enable_gpio_wakeup()`* 函数来启用此唤醒源。

此外，可将由 VDD3P3_RTC 电源域供电的 IO 用于芯片的 Deep-sleep 唤醒。调用*`esp_deep_sleep_enable_gpio_wakeup()`* 函数可以配置相应的唤醒管脚和唤醒触发电平，该函数用于启用相应管脚的 Deep-sleep 唤醒功能。

UART 唤醒（仅适用于 Light-sleep 模式） 当 ESP32-C2 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-C2 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。

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。

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。

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 时，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。

进入 Light-sleep 模式

函数 *esp_light_sleep_start()* 可用于在配置唤醒源后进入 Light-sleep 模式，也可用于在未配置唤醒源的情况下进入 Light-sleep 模式。在后一种情况中，芯片将一直处于睡眠模式，直到从外部被复位。

进入 Deep-sleep 模式

函数*esp_deep_sleep_start()* 可用于在配置唤醒源后进入 Deep-sleep 模式，也可用于在未配置唤醒源的情况下进入 Deep-sleep 模式模式。在后一种情况中，芯片将一直处于睡眠模式，直到从外部被复位。

配置 IO

一些 ESP32-C2 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。

在 Deep-sleep 模式中：

- 数字 GPIO (GPIO6 ~ 21) 处于高阻态。
- **RTC GPIO (GPIO0 ~ 5) 可能处于以下状态：**
  - 如果未启用保持 (hold) 功能，RTC GPIO 将处于高阻态。
  - 如果启用保持功能，RTC GPIO 管脚将会在保持功能开启时处于锁存状态。

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。

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。

应用程序示例

- protocols/sntp：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。
- wifi/power_save：如何实现 Modem-sleep 模式。
- system/deep_sleep：如何通过定时器触发 Deep-sleep 唤醒。

API 参考

Header File

- components/esp_hw_support/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_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)

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_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 care to pull-up or pull-down before using this function, because this will be set internally in esp_deep_sleep_start based on the wakeup mode. 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

void **esp_deep_sleep_start**(void)

Enter deep sleep with the configured wakeup options.

This function does not return.

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

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.

esp_deep_sleep does not shut down WiFi, BT, and higher level protocol connections gracefully. Make sure relevant WiFi and BT stack functions are called to close any connections and deinitialize the peripherals. These include:

- esp_bluedroid_disable
- esp_bt_controller_disable
- esp_wifi_stop

This function does not return.

备注: The device will wake up immediately if the deep-sleep time is set to 0

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

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_ALL_LOW**

Wake the chip when all selected GPIOs go low.

enumerator **ESP_EXT1_WAKEUP_ANY_HIGH**

Wake the chip when any of the selected GPIOs go high.

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_XTAL**

XTAL oscillator.

enumerator **ESP_PD_DOMAIN_RC_FAST**

Internal Fast oscillator.

enumerator **ESP_PD_DOMAIN_VDDSDIO**

VDD_SDIO.

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**

2.10.26 SoC Capabilities

This section lists definitions of the ESP32-C2’s SoC hardware capabilities. These definitions are commonly used in IDF to control which hardware dependent features are supported and thus compiled into the binary.

备注： These defines are currently not considered to be part of the public API, and may be changed at any time.

API Reference

Header File

- components/soc/esp32c2/include/soc/soc_caps.h

Macros

SOC_ADC_SUPPORTED

SOC_DEDICATED_GPIO_SUPPORTED

SOC_UART_SUPPORTED

SOC_GDMA_SUPPORTED

SOC_GPTIMER_SUPPORTED

SOC_BT_SUPPORTED

SOC_WIFI_SUPPORTED

SOC_ASYNC_MEMCPY_SUPPORTED

SOC_SUPPORTS_SECURE_DL_MODE

SOC_EFUSE_KEY_PURPOSE_FIELD

SOC_EFUSE_CONSISTS_OF_ONE_KEY_BLOCK

SOC_TEMP_SENSOR_SUPPORTED

SOC_LEDC_SUPPORTED

SOC_I2C_SUPPORTED

SOC_GPSPI_SUPPORTED

SOC_SHA_SUPPORTED

SOC_ECC_SUPPORTED

SOC_FLASH_ENC_SUPPORTED

SOC_SECURE_BOOT_SUPPORTED

SOC_SYSTIMER_SUPPORTED

SOC_BOD_SUPPORTED

SOC_XTAL_SUPPORT_26M

SOC_XTAL_SUPPORT_40M

SOC_ADC_DIG_CTRL_SUPPORTED

< SAR ADC Module

SOC_ADC_DIG_IIR_FILTER_SUPPORTED

SOC_ADC_MONITOR_SUPPORTED

SOC_ADC_DIG_SUPPORTED_UNIT(UNIT)

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

One pattern table, each contains 8 items. Each item takes 1 byte

SOC_ADC_DIGI_MIN_BITWIDTH

SOC_ADC_DIGI_MAX_BITWIDTH

SOC_ADC_DIGI_IIR_FILTER_NUM

SOC_ADC_DIGI_MONITOR_NUM

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

SOC_ADC_SELF_HW_CALI_SUPPORTED

support HW offset self calibration

SOC_BROWNOUT_RESET_SUPPORTED

SOC_SHARED_IDCACHE_SUPPORTED

SOC_CPU_CORES_NUM

SOC_CPU_INTR_NUM

SOC_CPU_HAS_FLEXIBLE_INTC

SOC_CPU_BREAKPOINTS_NUM

SOC_CPU_WATCHPOINTS_NUM

SOC_CPU_WATCHPOINT_SIZE

SOC_CPU_IDRAM_SPLIT_USING_PMP

SOC_GDMA_GROUPS

SOC_GDMA_PAIRS_PER_GROUP

SOC_GDMA_TX_RX_SHARE_INTERRUPT

SOC_GPIO_PORT

SOC_GPIO_PIN_COUNT

SOC_GPIO_SUPPORT_PIN_GLITCH_FILTER

SOC_GPIO_FILTER_CLK_SUPPORT_APB

SOC_GPIO_SUPPORT_FORCE_HOLD

SOC_GPIO_SUPPORT_DEEPSLEEP_WAKEUP

SOC_GPIO_VALID_GPIO_MASK

SOC_GPIO_VALID_OUTPUT_GPIO_MASK

SOC_GPIO_DEEP_SLEEP_WAKE_VALID_GPIO_MASK

SOC_GPIO_VALID_DIGITAL_IO_PAD_MASK

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_I2C_NUM

SOC_I2C_FIFO_LEN

I2C hardware FIFO depth

SOC_I2C_SUPPORT_HW_CLR_BUS

SOC_I2C_SUPPORT_XTAL

SOC_I2C_SUPPORT_RTC

SOC_LEDC_SUPPORT_PLL_DIV_CLOCK

SOC_LEDC_SUPPORT_XTAL_CLOCK

SOC_LEDC_CHANNEL_NUM

SOC_LEDC_TIMER_BIT_WIDTH

SOC_LEDC_SUPPORT_FADE_STOP

SOC_MMU_PAGE_SIZE_CONFIGURABLE

SOC_MMU_LINEAR_ADDRESS_REGION_NUM

SOC_MMU_PERIPH_NUM

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_RTC_CNTL_CPU_PD_DMA_BUS_WIDTH

SOC_RTC_CNTL_CPU_PD_REG_FILE_NUM

SOC_RTC_CNTL_CPU_PD_DMA_ADDR_ALIGN

SOC_RTC_CNTL_CPU_PD_DMA_BLOCK_SIZE

SOC_RTC_CNTL_CPU_PD_RETENTION_MEM_SIZE

SOC_RTCIO_PIN_COUNT

SOC_RSA_MAX_BIT_LEN

SOC_SHA_SUPPORT_RESUME

SOC_SHA_SUPPORT_SHA1

SOC_SHA_SUPPORT_SHA224

SOC_SHA_SUPPORT_SHA256

SOC_SPI_PERIPH_NUM

SOC_SPI_PERIPH_CS_NUM(i)

SOC_SPI_MAX_CS_NUM

SOC_SPI_MAXIMUM_BUFFER_SIZE

SOC_SPI_SUPPORT_DDRCLK

SOC_SPI_SLAVE_SUPPORT_SEG_TRANS

SOC_SPI_SUPPORT_CD_SIG

SOC_SPI_SUPPORT_CONTINUOUS_TRANS

SOC_SPI_SUPPORT_SLAVE_HD_VER2

SOC_SPI_SUPPORT_CLK_XTAL

SOC_SPI_SUPPORT_CLK_PLL_F40M

SOC_SPI_PERIPH_SUPPORT_MULTILINE_MODE(host_id)

SOC_SPI_PERIPH_SUPPORT_CONTROL_DUMMY_OUT

SOC_MEMSPI_IS_INDEPENDENT

SOC_SPI_MAX_PRE_DIVIDER

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_WRAP

SOC_MEMSPI_SRC_FREQ_60M_SUPPORTED

SOC_MEMSPI_SRC_FREQ_30M_SUPPORTED

SOC_MEMSPI_SRC_FREQ_20M_SUPPORTED

SOC_MEMSPI_SRC_FREQ_15M_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_INT_LEVEL

SOC_SYSTIMER_ALARM_MISS_COMPENSATE

SOC_TIMER_GROUPS

SOC_TIMER_GROUP_TIMERS_PER_GROUP

SOC_TIMER_GROUP_COUNTER_BIT_WIDTH

SOC_TIMER_GROUP_SUPPORT_XTAL

SOC_TIMER_GROUP_TOTAL_TIMERS

SOC_EFUSE_DIS_DOWNLOAD_ICACHE

SOC_EFUSE_DIS_PAD_JTAG

SOC_EFUSE_DIS_DIRECT_BOOT

SOC_SECURE_BOOT_V2_ECC

SOC_EFUSE_SECURE_BOOT_KEY_DIGESTS

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_128_DERIVED

SOC_UART_NUM

SOC_UART_FIFO_LEN
The UART hardware FIFO length

SOC_UART_BITRATE_MAX
Max bit rate supported by UART

SOC_UART_SUPPORT_WAKEUP_INT
Support UART wakeup interrupt

SOC_UART_SUPPORT_PLL_F40M_CLK
Support APB 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_FSM_TX_WAIT_SEND

SOC_COEX_HW_PTI

SOC_EXTERNAL_COEX_ADVANCE

SOC_PHY_DIG_REGS_MEM_SIZE

SOC_WIFI_LIGHT_SLEEP_CLK_WIDTH

SOC_PM_SUPPORT_WIFI_WAKEUP

SOC_PM_SUPPORT_BT_WAKEUP

SOC_PM_SUPPORT_RC_FAST_PD

SOC_PM_SUPPORT_VDDSDIO_PD

SOC_CLK_RC_FAST_D256_SUPPORTED

SOC_RTC_SLOW_CLK_SUPPORT_RC_FAST_D256

SOC_CLK_RC_FAST_SUPPORT_CALIBRATION

SOC_CLK_OSC_SLOW_SUPPORTED
ESP32C2 only supports to connect an external oscillator, not a crystal

SOC_WIFI_HW_TSF
Support hardware TSF

SOC_WIFI_SUPPORT_VARIABLE_BEACON_WINDOW
Support delta early time for rf phy on/off

SOC_BLE_SUPPORTED
Support Bluetooth Low Energy hardware

SOC_BLE_MESH_SUPPORTED
Support BLE MESH

SOC_ESP_NIMBLE_CONTROLLER
Support BLE EMBEDDED controller V1

SOC_BLE_50_SUPPORTED
Support Bluetooth 5.0

SOC_BLE_DEVICE_PRIVACY_SUPPORTED
Support BLE device privacy mode

SOC_BLUFI_SUPPORTED
Support BLUFI

2.10.27 系统时间

概述

ESP32-C2 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。
- `管脚 GPIO0 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 GPIO0。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 GPIO0 无法用作 GPIO 管脚。
- `内置 17.5 MHz 振荡器的 256 分频时钟 (~68 kHz)`：频率稳定性优于 `内置 136 kHz RC 振荡器`，同样无需外部元件，但 Deep-sleep 模式下电流消耗更高（比默认模式高 5 µA）。

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*。

获取当前时间

要获取当前时间，请使用 POSIX 函数 `gettimeofday()`。此外，您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。

您也可以直接使用 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

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

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)

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**

2.10.28 The Async memcpy API

Overview

ESP32-C2 has a DMA engine which can help to offload internal memory copy operations from the CPU in a asynchronous way.

The async memcpy API wraps all DMA configurations and operations, the signature of *esp_async_memcpy()* is almost the same to the standard libc one.

Thanks to the benefit of the DMA, we don't have to wait for each memory copy to be done before we issue another memcpy request. By the way, it's still possible to know when memcpy is finished by listening in the memcpy callback function.

Configure and Install driver

esp_async_memcpy_install() is used to install the driver with user's configuration. Please note that async memcpy has to be called with the handle returned from *esp_async_memcpy_install()*.

Driver configuration is described in *async_memcpy_config_t*:

- backlog: This is used to configure the maximum number of DMA operations being processed at the same time.
- sram_trans_align: Declare SRAM alignment for both data address and copy size, set to zero if the data has no restriction in alignment. If set to a quadruple value (i.e. 4X), the driver will enable the burst mode internally, which is helpful for some performance related application.
- psram_trans_align: Declare PSRAM alignment for both data address and copy size. User has to give it a valid value (only 16, 32, 64 are supported) if the destination of memcpy is located in PSRAM. The default alignment (i.e. 16) will be applied if it's set to zero. Internally, the driver configures the size of block used by DMA to access PSRAM, according to the alignment.
- flags: This is used to enable some special driver features.

ASYNC_MEMCPY_DEFAULT_CONFIG provides a default configuration, which specifies the backlog to 8.

```
async_memcpy_config_t config = ASYNC_MEMCPY_DEFAULT_CONFIG();
// update the maximum data stream supported by underlying DMA engine
config.backlog = 16;
async_memcpy_t driver = NULL;
ESP_ERROR_CHECK(esp_async_memcpy_install(&config, &driver)); // install driver,␣
↪return driver handle
```

Send memory copy request

esp_async_memcpy() is the API to send memory copy request to DMA engine. It must be called after driver is installed successfully. This API is thread safe, so it can be called from different tasks.

Different from the libc version of memcpy, user should also pass a callback to *esp_async_memcpy()*, if it's necessary to be notified when the memory copy is done. The callback is executed in the ISR context, make sure you won't violate the restriction applied to ISR handler.

Besides that, the callback function should reside in IRAM space by applying IRAM_ATTR attribute. The prototype of the callback function is *async_memcpy_isr_cb_t*, please note that, the callback function should return true if it wakes up a high priority task by some API like xSemaphoreGiveFromISR().

```
// Callback implementation, running in ISR context
static IRAM_ATTR bool my_async_memcpy_cb(async_memcpy_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); // high_task_wakeup set to␣
↪pdTRUE if some high priority task unblocked
    return high_task_wakeup == pdTRUE;
}

// Create a semaphore used to report the completion of async memcpy
SemaphoreHandle_t semphr = xSemaphoreCreateBinary();

// Called from user's context
ESP_ERROR_CHECK(esp_async_memcpy(driver_handle, to, from, copy_len, my_async_
↪memcpy_cb, my_semaphore));
```

(下页继续)

(续上页)

```
// Do something else here
xSemaphoreTake(my_semaphore, portMAX_DELAY); // Wait until the buffer copy is done
```

Uninstall driver (optional)

esp_async_memcpy_uninstall() is used to uninstall asynchronous memcpy driver. It's not necessary to uninstall the driver after each memcpy operation. If you know your application won't use this driver anymore, then this API can recycle the memory for you.

API Reference

Header File

- components/esp_hw_support/include/esp_async_memcpy.h

Functions

esp_err_t **esp_async_memcpy_install** (const *async_memcpy_config_t* *config, *async_memcpy_t* *asmcp)

Install async memcpy driver.

参数

- **config** –**[in]** Configuration of async memcpy
- **asmcp** –**[out]** Handle of async memcpy that returned from this API. If driver installation is failed, asmcp would be assigned to NULL.

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_t* asmcp)

Uninstall async memcpy driver.

参数 **asmcp** –**[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_t* asmcp, 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

- **asmcp** –**[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_t* asmcp, *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

- **asmcp** –**[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

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**

Type of async memcpy event object.

Public Members

void ***data**

Event data

struct **async_memcpy_config_t**

Type of async memcpy configuration.

Public Members

uint32_t **backlog**

Maximum number of streams that can be handled simultaneously

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

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_t**

Type of async memcpy handle.

typedef bool (***async_memcpy_isr_cb_t**)(*async_memcpy_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

2.10.29 Watchdogs

Overview

The ESP-IDF has support for multiple types of watchdogs, with the two main ones being: The Interrupt Watchdog Timer and the Task Watchdog Timer (TWDT). The Interrupt Watchdog Timer and the TWDT can both be enabled using 项目配置菜单, however the TWDT can also be enabled during runtime. The Interrupt Watchdog is responsible for detecting instances where FreeRTOS task switching is blocked for a prolonged period of time. The TWDT is responsible for detecting instances of tasks running without yielding for a prolonged period.

ESP-IDF has support for the following types of watchdog timers:

- Interrupt Watchdog Timer (IWDT)
- Task Watchdog Timer (TWDT)

The various watchdog timers can be enabled using the 项目配置菜单. However, the TWDT can also be enabled during runtime.

Interrupt Watchdog Timer (IWDT)

The purpose of the IWDT is to ensure that interrupt service routines (ISRs) are not blocked from running for a prolonged period of time (i.e., the IWDT timeout period). Blocking ISRs from running in a timely manner is undesirable

as it can increases ISR latency, and also prevents task switching (as task switching is executed form an ISR). The things that can block ISRs from running include:

- Disabling interrupts
- Critical Sections (also disables interrupts)
- Other same/higher priority ISRs (will block same/lower priority ISRs from running it completes execution)

The IWDT utilizes the watchdog timer in Timer Group 0 as its underlying hardware timer and leverages the FreeRTOS tick interrupt on each CPU to feed the watchdog timer. If the tick interrupt on a particular CPU is not run at within the IWDT timeout period, it is indicative that something is blocking ISRs from being run on that CPU (see the list of reasons above).

When the IWDT times out, the default action is to invoke the panic handler and display the panic reason as `Interrupt wdt timeout on CPU0` or `Interrupt wdt timeout on CPU1` (as applicable). Depending on the panic handler's configured behavior (see *CONFIG_ESP_SYSTEM_PANIC*), users can then debug the source of the IWDT timeout (via the backtrace, OpenOCD, gdbstub etc) or simply reset the chip (which may be preferred in a production environment).

If for whatever reason the panic handler is unable to run after an IWDT timeout, the IWDT has a secondary timeout that will hard-reset the chip (i.e., a system reset).

Configuration

- The IWDT is enabled by default via the *CONFIG_ESP_INT_WDT* option.
- The IWDT's timeout is configured by setting the *CONFIG_ESP_INT_WDT_TIMEOUT_MS* option.
  - Note that the default timeout is higher if PSRAM support is enabled, as a critical section or interrupt routine that accesses a large amount of PSRAM will take longer to complete in some circumstances.
  - The timeout should always at least twice longer than the period between FreeRTOS ticks (see *CONFIG_FREERTOS_HZ*).

Tuning If you find the IWDT timeout is triggered because an interrupt or critical section is running longer than the timeout period, consider rewriting the code:

- Critical sections should be made as short as possible. Any non-critical code/computation should be placed outside the critical section.
- Interrupt handlers should also perform the minimum possible amount of computation. Users can consider deferring any computation to a task by having the ISR push data to a task using queues.

Neither critical sections or interrupt handlers should ever block waiting for another event to occur. If changing the code to reduce the processing time is not possible or desirable, it's possible to increase the *CONFIG_ESP_INT_WDT_TIMEOUT_MS* setting instead.

Task Watchdog Timer (TWDT)

The Task Watchdog Timer (TWDT) is used to monitor particular tasks, ensuring that they are able to execute within a given timeout period. The TWDT primarily watches the Idle task, however any task can subscribe to be watched by the TWDT. By watching the Idle task, the TWDT can detect instances of tasks running for a prolonged period of time wihtout yielding. This can be an indicator of poorly written code that spinloops on a peripheral, or a task that is stuck in an infinite loop.

The ESP32-C2 has only a single Timer Group, used by Interrupt Watchdog (IWDT). Thus, the Task Watchdog is built around the *esp_timer* component in order to implement a software timer. When a timeout occurs, an interrupt is triggered, notifying the *esp_timer*'s main task. The later will then execute the TWDT callback previously registered. Users can define the function *esp_task_wdt_isr_user_handler* in the user code, in order to receive the timeout event and extend the default behavior.

Usage The following functions can be used to watch tasks using the TWDT:

- `esp_task_wdt_init()` to initialize the TWDT and subscribe the idle tasks.
- `esp_task_wdt_add()` subscribes other tasks to the TWDT.

- Once subscribed, `esp_task_wdt_reset()` should be called from the task to feed the TWDT.
- `esp_task_wdt_delete()` unsubscribes a previously subscribed task
- `esp_task_wdt_deinit()` unsubscribes the idle tasks and deinitializes the TWDT

In the case where applications need to watch at a more granular level (i.e., ensure that a particular functions/stub/code-path is called), the TWDT allows subscription of "users".

- `esp_task_wdt_add_user()` to subscribe an arbitrary user of the TWDT. This function will return a user handle to the added user.
- `esp_task_wdt_reset_user()` must be called using the user handle in order to prevent a TWDT timeout.
- `esp_task_wdt_delete_user()` unsubscribes an arbitrary user of the TWDT.

Configuration The default timeout period for the TWDT is set using config item *CONFIG_ESP_TASK_WDT_TIMEOUT_S*. This should be set to at least as long as you expect any single task will need to monopolize the CPU (for example, if you expect the app will do a long intensive calculation and should not yield to other tasks). It is also possible to change this timeout at runtime by calling `esp_task_wdt_init()`.

备注: Erasing large flash areas can be time consuming and can cause a task to run continuously, thus triggering a TWDT timeout. The following two methods can be used to avoid this:

- Increase *CONFIG_ESP_TASK_WDT_TIMEOUT_S* in menuconfig for a larger watchdog timeout period.
- You can also call `esp_task_wdt_init()` to increase the watchdog timeout period before erasing a large flash area.

For more information, you can refer to *SPI Flash*.

The following config options control TWDT configuration. They are all enabled by default:

- *CONFIG_ESP_TASK_WDT_EN* - enables TWDT feature. If this option is disabled, TWDT cannot be used, even if initialized at runtime.
- *CONFIG_ESP_TASK_WDT_INIT* - the TWDT is initialized automatically during startup. If this option is disabled, it is still possible to initialize the Task WDT at runtime by calling `esp_task_wdt_init()`.
- *CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0* - Idle task is subscribed to the TWDT during startup. If this option is disabled, it is still possible to subscribe the idle task by calling `esp_task_wdt_init()` again.

备注: On a TWDT timeout the default behaviour is to simply print a warning and a backtrace before continuing running the app. If you want a timeout to cause a panic and a system reset then this can be configured through *CONFIG_ESP_TASK_WDT_PANIC*.

JTAG & Watchdogs

While debugging using OpenOCD, the CPUs will be halted every time a breakpoint is reached. However if the watchdog timers continue to run when a breakpoint is encountered, they will eventually trigger a reset making it very difficult to debug code. Therefore OpenOCD will disable the hardware timers of both the interrupt and task watchdogs at every breakpoint. Moreover, OpenOCD will not reenable them upon leaving the breakpoint. This means that interrupt watchdog and task watchdog functionality will essentially be disabled. No warnings or panics from either watchdogs will be generated when the ESP32-C2 is connected to OpenOCD via JTAG.

API Reference

Task Watchdog A full example using the Task Watchdog is available in esp-idf: system/task_watchdog

Header File

- components/esp_system/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.

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**

Mask of the cores who’s idle task should be subscribed on initialization

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.

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-C2 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-C2 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。

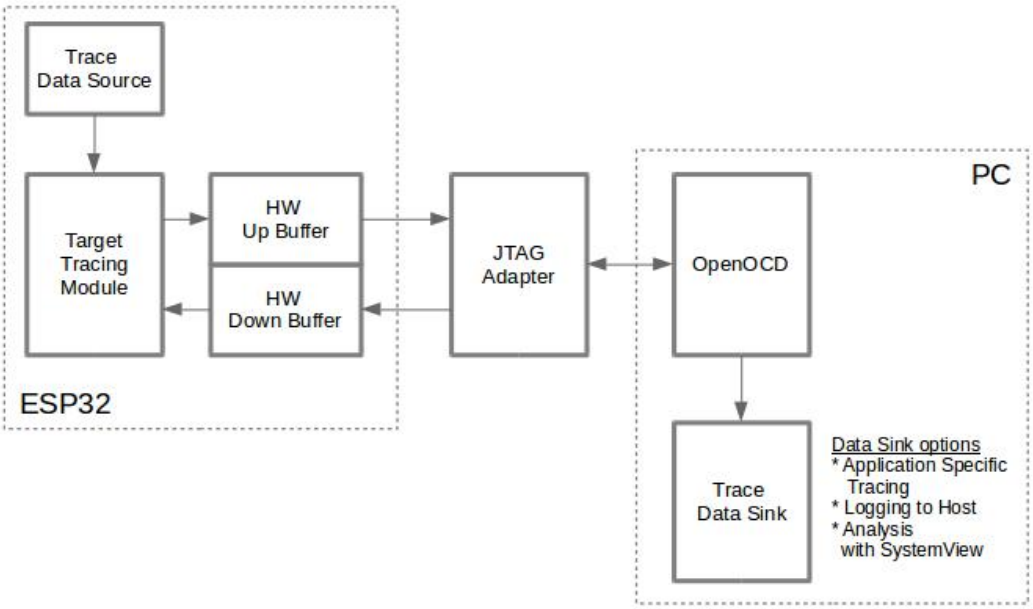


图 1: 使用 JTAG 接口的跟踪组件

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-C2 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。

   备注： 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。
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-C2 并实时接收数据。
2. *ESP32-C2 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-C2 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。您需要复位板卡或者在 OpenOCD 的窗口（非 telnet 会话窗口）输入 Ctrl+C 命令来手动停止跟踪。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。

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。

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-C2 gcov` 来触发运行时的实时转储。一旦被调用，OpenOCD 将立即抢占 ESP32-C2 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-C2 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-C2，检索代码覆盖数据，然后断开与 ESP32-C2 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。

请在您工程的 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)
```

您可使用以下命令：

- 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-C2 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-C2 内部的 ROM 中，它会从 flash 的 0x0 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 的调度器启动。

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，CPU 会立即开始运行，执行所有的初始化操作。复位向量代码位于 ESP32-C2 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-C2 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-C2 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 0x0 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。
- 将 CPU 时钟设置为项目配置的频率。

`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`。

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。

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。

4.3 BluFi

4.3.1 概览

BluFi 是一项基于蓝牙通道的 Wi-Fi 网络配置功能，适用于 ESP32-C2。它通过安全协议将 Wi-Fi 的 SSID、密码等配置信息传输到 ESP32-C2。基于这些信息，ESP32-C2 可进而连接到 AP 或建立 SoftAP。

BluFi 流程的关键部分包括数据的分片、加密以及校验和验证。

用户可按需自定义用于对称加密、非对称加密以及校验的算法。此处，我们采用 DH 算法进行密钥协商，128-AES 算法用于数据加密，CRC16 算法用于校验和验证。

4.3.2 BluFi 流程

BluFi 配网流程包含配置 SoftAP 和配置 Station 两部分。

下面以配置 Station 为例，介绍了广播、连接、服务发现、协商共享密钥、传输数据、回传连接状态等关键步骤。

1. ESP32-C2 开启 GATT Server 模式，发送带有特定 *advertising data* 的广播。该广播不属于 BluFi Profile，您可以按需对其进行自定义。
2. 使用手机应用程序搜索到该广播后，手机将作为 GATT Client 连接 ESP32-C2。该步骤对具体使用哪款手机应用程序并无特殊要求。
3. 成功建立 GATT 连接后，手机会向 ESP32-C2 发送数据帧进行密钥协商（详见*BluFi 中定义的帧格式*）。
4. ESP32-C2 收到密钥协商的数据帧后，会按照您自定义的协商方法进行解析。
5. 手机与 ESP32-C2 进行密钥协商。协商过程可使用 DH/RSA/ECC 等加密算法。
6. 协商结束后，手机端向 ESP32-C2 发送控制帧，用于设置安全模式。
7. ESP32-C2 收到控制帧后，使用共享密钥以及安全配置对通信数据进行加密和解密。
8. 手机向 ESP32-C2 发送*BluFi 中定义的帧格式* 中定义的数据帧，包括 SSID、密码等 Wi-Fi 配置信息。
9. 手机向 ESP32-C2 发送 Wi-Fi 连接请求的控制帧。ESP32-C2 收到控制帧后，即默认手机已完成必要信息的传输，准备连接 Wi-Fi。
10. 连接到 Wi-Fi 后，ESP32-C2 发送 Wi-Fi 连接状态报告的控制帧到手机。至此，配网结束。

备注:

1. ESP32-C2 收到安全模式配置的控制帧后，会根据定义的安全模式进行相关操作。
2. 进行对称加密和解密时，加密和解密前后的数据长度必须一致。支持原地加密和解密。

4.3.3 BluFi 流程图

4.3.4 BluFi 中定义的帧格式

手机应用程序与 ESP32-C2 之间的 BluFi 通信格式定义如下：

帧不分片格式：

字段	值（字节）
类型（最低有效位）	1
帧控制	1
序列号	1
数据长度	1
数据	${Data Length}
校验（最高有效位）	2

如果使能 **帧控制**字段中的分片位，则 **数据**字段中会出现 2 字节的 **内容总长度**。该 **内容总长度**表示帧的剩余部分的总长度，并用于报告终端需要分配的内存大小。

帧分片格式：

图 2: BluFi Flow Chart

字段	值（字节）
类型（最低有效位）	1
帧控制（分片）	1
序列号	1
数据长度	1
数据	• 内容总长度：2 • 数据内容长度：${Data Length} - 2
校验（最高有效位）	2

通常情况下，控制帧不包含数据位，ACK 帧类型除外。

ACK 帧格式（8 bit）：

字段	值（字节）
类型 - ACK（最低有效位）	1
帧控制	1
序列号	1
数据长度	1
数据	ACK 序列号: 2
校验（最高有效位）	2

1. 类型字段
   类型字段占 1 字节，分为 **类型**和 **子类型**两部分。其中，**类型**占低 2 位，表明该帧为数据帧或是控制帧；**子类型**占高 6 位，表示此数据帧或者控制帧的具体含义。
   - 控制帧，暂不进行加密，可校验。
   - 数据帧，可加密，可校验。

1.1 控制帧（二进制：0x0 b’00）

控制帧	含义	解释	备注
0x0 (b’000000)	ACK	ACK 帧的数据字段使用回复对象帧的序列值。	数据字段占用 1 字节，其序列值与回复对象帧的序列值相同。
0x1 (b’000001)	将 ESP 设备设置为安全模式。	通知 ESP 设备发送数据时使用的安全模式，在数据发送过程中可多次重置。设置后，将影响后续使用的安全模式。 如果不设置，ESP 设备将默认发送不带校验和加密的控制帧和数据帧。从手机到 ESP 设备的数据传输是由这个控制帧控制的。	数据字段占一个字节。高 4 位用于控制帧的安全模式设置，低 4 位用于数据帧的安全模式设置。 • b’0000：无校验、无加密； • b’0001：有校验、无加密； • b’0010：无校验、有加密； • b’0011：有校验、有加密。
0x2 (b’000010)	设置 Wi-Fi 的 opmode。	该帧包含设置 ESP 设备 Wi-Fi 模式 (opmode) 的设置信息。	data[0] 用于设置 opmode，包括： • 0x00: NULL • 0x01: STA • 0x02: SoftAP • 0x03: SoftAP & STA 如果设置中包含 AP，请尽量优先设置 AP 模式的 SSID/密码/最大连接数等。
0x3 (b’000011)	将 ESP 设备连接至 AP。	通知 ESP 设备必要的信息已经发送完毕，可以连接至 AP。	不包含数据字段。
0x4 (b’000100)	断开 ESP 设备与 AP 的连接。		不包含数据字段。
0x5 (b’000101)	获取 ESP 设备的 Wi-Fi 模式和状态等信息。		• 不包含数据字段。ESP 设备收到此控制帧后，会向手机回发一个报告 Wi-Fi 连接状态的帧来告知手机端当前所处的 opmode、连接状态、SSID 等信息。 • 提供给手机端的信息类型由手机上的应用程序决定。
0x6 (b’000110)	断开 STA 设备与 SoftAP 的连接（SoftAP 模式）。		data[0~5] 为 STA 设备的 MAC 地址。如有多个 STA 设备，则第二个使用 data[6-11]，依次类推。
0x7 (b’000111)	获取版本信息。		
0x8 (b’001000)	断开 BLE GATT 连接。		ESP 设备收到该指令后主动断开 BLE GATT 连接。
0x9 (b’001001)	获取 Wi-Fi 列表。	通知 ESP 设备扫描周围的 Wi-Fi 热点。	不包含数据字段。ESP 设备收到此控制帧后，会向手机回发一个包含 Wi-Fi 热点报告的帧。

1.2 数据帧 (二进制：0x1 b’01)

数据帧	含义	解释	备注
0x0 (b’000000)	发送协商数据。	协商数据会发送到应用层注册的回调函数中。	数据的长度取决于数据长度字段。
0x1 (b’000001)	发送 STA 模式的 BSSID。	在 SSID 隐藏的情况下，发送 STA 设备要连接的 AP 的 BSSID。	请参考备注 1。
0x2 (b’000010)	发送 STA 模式的 SSID	发送 STA 设备要连接的 AP 的 SSID。	请参考备注 1。
0x3 (b’000011)	发送 STA 模式的密码。	发送 STA 设备要连接的 AP 的密码。	请参考备注 1。
0x4 (b’000100)	发送 SoftAP 模式的 SSID。		请参考备注 1。
0x5 (b’000101)	发送 SoftAPmode 模式的密码。		请参考备注 1。
0x6 (b’000110)	设置 SoftAPmode 模式的最大连接数。		data[0] 为连接数的值，范围从 1 到 4。当传输方向是 ESP 设备到手机时，表示向手机端提供所需信息。
0x7 (b’000111)	设置 SoftAP 的认证模式。		data[0] 包括： • 0x00: OPEN • 0x01: WEP • 0x02: WPA_PSK • 0x03: WPA2_PSK • 0x04: WPA_WPA2_PSK 若传输方向是从 ESP 设备到手机，则表示向手机端提供所需信息。
0x8 (b’001000)	设置 SoftAP 模式的通道数量。		data[0] 代表支持的通道的数量，范围从 1 到 14。若传输方向是从 ESP 设备到手机，则表示向手机端提供所需信息。
0x9 (b’001001)	用户名	在进行企业级加密时提供 GATT 客户端的用户名。	数据的长度取决于数据长度字段。
0xa (b’001010)	CA 认证	在进行企业级加密时提供 CA 认证。	请参考备注 2。
0xb (b’001011)	客户端认证	在进行企业级加密时提供客户端认证。是否包含私钥，取决于认证的内容。	请参考备注 2。
0xc (b’001100)	服务端认证	在进行企业级加密时提供服务端认证。是否包含私钥，取决于认证的内容。	请参考备注 2。
0xd (b’001101)	客户端私钥	在进行企业级加密时提供客户端私钥。	请参考备注 2。
0xe (b’001110)	服务端私钥	在进行企业级加密时提供服务端私钥。	请参考备注 2。
0xf (b’001111)	Wi-Fi 连接状态报告	通知手机 ESP 设备的 Wi-Fi 状态，包括 STA 状态和 SoftAP 状态。用于 STA 设备连接手机或 SoftAP。但是，当手机接收到 Wi-Fi 状态时，除了本帧之外，还可以回复其他帧。	data[0] 表示 opmode，包括： • 0x00: NULL • 0x01: STA • 0x02: SoftAP • 0x03: SoftAP & STA data[1]：STA 设备的连接状态。0x0 表示处于连接状态且获得 IP 地址，0x1 表示处于非连接状态, 0x2 表示处于正在连接状态，0x3 表示处于连接状态但未获得 IP 地址。 data[2]：SoftAP 的连接状态，即表示有多

备注:

- 备注 1: 数据的长度取决于数据长度字段。若传输方向是从 ESP 设备到手机，则表示向手机端提供所需信息。
- 备注 2: 数据的长度取决于数据长度字段。如果数据长度不够，该帧可用分片。

2. Frame Control
   帧控制字段，占 1 字节，每个位表示不同含义。

位	含义
0x01	表示帧是否加密。 • 1 表示加密。 • 0 表示未加密。 该帧的加密部分包括数据字段加密之前的完整明文数据（不包括校验部分）。控制帧暂不加密，故控制帧此位为 0。
0x02	该数据字段表示帧尾是否包含校验位，如 SHA1、MD5、CRC 等。该数据字段包含序列、数据长度以及明文。控制帧和数据帧都可以选择包含或不包含校验位。
0x04	表示数据方向。 • 0 表示传输方向是从手机到 ESP 设备。 • 1 表示传输方向是从 ESP 设备到手机。
0x08	表示是否要求对方回复 ACK。 • 0 表示不要求回复 ACK。 • 1 表示要求回复 ACK。
0x10	表示是否有后续的数据分片。 • 0 表示此帧没有后续数据分片。 • 1 表示还有后续数据分片，用来传输较长的数据。 对于分片帧，在数据字段的前两个字节中，会给定当前内容部分和随后内容部分的总长度（即最大支持 64 K 的数据内容）。
0x10~0x80	保留

3. 序列控制
   序列控制字段。帧发送时，无论帧的类型是什么，序列都会自动加 1，用来防止重放攻击 (Replay Attack)。每次重新连接后，序列清零。
4. 长度
   数据字段的长度，不包含校验部分。
5. 数据
   对于不同的类型或子类型，数据字段的含义均不同。请参考上方表格。
6. 校验
   此字段占两个字节，用来校验序列、数据长度以及明文。

4.3.5 ESP32-C2 端的安全实现

1. 数据安全
   为了保证 Wi-Fi SSID 和密码的传输过程是安全的，需要使用对称加密算法（例如 AES、DES 等）对报文进行加密。在使用对称加密算法之前，需要使用非对称加密算法（DH、RSA、ECC 等）协商出（或生成出）一个共享密钥。
2. 保证数据完整性
   为了保证数据完整性，需要加入校验算法，例如 SHA1、MD5、CRC 等。
3. 身份安全（签名）
   某些算法如 RSA 可以保证身份安全。但如 DH 这类的算法，本身不能保证身份安全，需要添加其他算法来签名。
4. 防止重放攻击 (Replay Attack)
   添加其到序列字段中，并且在数据校验过程中使用。

在 ESP32-C2 端的代码中，你可以决定和开发如密钥协商等安全处理的流程。手机应用向 ESP32-C2 发送协商数据，数据会传送给应用层处理。如果应用层不处理，可使用 BluFi 提供的 DH 加密算法来协商密钥。
应用层需向 BluFi 注册以下几个与安全相关的函数：

```
typedef void (*esp_blufi_negotiate_data_handler_t)(uint8_t *data, int len, uint8_t␣
↪**output_data, int *output_len, bool *need_free)
```

该函数用来接收协商期间的正常数据 (normal data)。数据处理完成后，需要将待发送的数据使用 output_data 和 output_len 传出。

BluFi 会在调用完 Negotiate_data_handler 后，发送 Negotiate_data_handler 传出的 output_data。

这里的两个 “*” 是因为需要发出去的数据长度未知，所以需要函数自行分配 (malloc) 或者指向全局变量，并告知是否需要通过 NEED_FREE 释放内存。

```
typedef int (* esp_blufi_encrypt_func_t)(uint8_t iv8, uint8_t *crypt_data, int␣
↪crypt_len)
```

加密和解密的数据长度必须一致。其中 IV8 为帧的 8 位序列，可作为 IV 的某 8 个位来使用。

```
typedef int (* esp_blufi_decrypt_func_t)(uint8_t iv8, uint8_t *crypt_data, int␣
↪crypt_len)
```

加密和解密的数据长度必须一致。其中 IV8 为帧的 8 位序列，可作为 IV 的某 8 个位来使用。

```
typedef uint16_t (*esp_blufi_checksum_func_t)(uint8_t iv8, uint8_t *data, int len)
```

该函数用来进行校验，返回值为校验的值。BluFi 会使用该函数返回值与帧的校验值进行比较。

4.3.6 GATT 相关说明

UUID

BluFi Service UUID：0xFFFF，16 bit

BluFi （手机 -> ESP32-C2）特性：0xFF01，主要权限：可写

BluFi （ESP32-C2 -> 手机）特性：0xFF02，主要权限：可读可通知

4.4 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 0x0 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*。

4.4.1 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。

4.4.2 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*Logging library*）。

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。

4.4.3 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-C2 的技术规格书。

有时应用程序需要知道设备是否触发了出厂重置，但 ESP32-C2 没有 RTC FAST 内存，因此没有相应的 API 可用于监测。然而也有方法实现出厂重置监测，比如，设置一个在出厂重置时会被引导加载程序擦除的 NVS 分区（需将此分区添加到*CONFIG_BOOTLOADER_DATA_FACTORY_RESET* 中）。在这个 NVS 分区中保存一个令牌数据“factory_reset_state”，让该令牌在应用程序中自增。”factory_reset_state” 为 0 时则表明触发了出厂重置。

4.4.4 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。

为了使主项目的引导加载程序支持这个功能，请设置*CONFIG_BOOTLOADER_APP_TEST* 并配置以下两个选项：

- *CONFIG_BOOTLOADER_NUM_PIN_APP_TEST* - 设置启动 TEST 分区的管脚编号。选中的管脚将被配置为启用了内部上拉的输入。要触发测试应用，必须在重置时将此管脚拉低。
  当管脚输入被释放（则被拉高）并将设备重新启动后，正常配置的应用程序将启动（工厂或任意 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，管脚在设定的时间内必须持续保持低电平，然后才会执行出厂重置或引导测试分区（如适用）。

4.4.5 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。

4.4.6 看门狗

默认情况下，硬件 RTC 看门狗定时器在引导加载程序运行时保持运行，如果 9 秒后没有应用程序成功启动，它将自动重置芯片。

- 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。
- 可以通过调整应用程序的行为使 RTC 看门狗在应用程序启动后保持启用。看门狗需要由应用程序显示地重置（即“喂狗”），以避免重置。为此，请设置*CONFIG_BOOTLOADER_WDT_DISABLE_IN_USER_CODE* 选项，根据需要修改应用程序，然后重新编译应用程序。
- 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。

4.4.7 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-C2 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）。

4.4.8 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。

4.5 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。如需您想了解如何组织和构建新的 ESP-IDF 项目或组件，请阅读本文档。

4.5.1 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。

4.5.2 使用构建系统

idf.py

idf.py 命令行工具提供了一个前端，可以帮助您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，但是您愿意，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： 如果您已经熟悉了 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 您可以直接使用 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。您可以在操作系统或 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

您还可以使用集成了 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 3.7 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`。您可以通过 `python3 $IDF_PATH/tools/idf.py ...` 命令切换到别的 Python 解释器，或者您可以通过设置 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，它们会帮助您更改默认的 python 版本。

4.5.3 示例项目

示例项目的目录树结构可能如下所示:

```
- myProject/
             - CMakeLists.txt
             - sdkconfig
             - components/ - component1/ - CMakeLists.txt
                                         - Kconfig
                                         - src1.c
                           - component2/ - CMakeLists.txt
                                         - Kconfig
                                         - src1.c
                                         - include/ - component2.h
             - main/       - CMakeLists.txt
                           - src1.c
```

(下页继续)

(续上页)

```
                        - 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“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，您也可以在顶层 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，但您可以修改此变量。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*。

4.5.4 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。如果您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。

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。

4.5.5 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
    -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。

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。

4.5.6 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。

4.5.7 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。

4.5.8 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，我们都建议您使用 `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)
```

4.5.9 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，您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。

`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。

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，您需要在项目的 `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，您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。

4.5.10 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。

4.5.11 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。

4.5.12 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()

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)
```

（下页继续）

(续上页)

```
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

有时您的组件希望使用一个二进制文件或者文本文件，但是您又不希望将它们重新格式化为 C 源文件。

这时，您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，您也可以设置 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。

4.5.13 自定义 sdkconfig 的默认值

对于示例工程或者其他您不想指定完整 sdkconfig 配置的项目，但是您确实希望覆盖 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

除了 `sdkconfig.defaults` 之外，构建系统还将从 `sdkconfig.defaults.TARGET_NAME` 文件加载默认值，其中 `IDF_TARGET` 的值为 `TARGET_NAME`。例如，对于 `ESP32` 这个硬件目标，sdkconfig 的默认值会首先从 `sdkconfig.defaults` 获取，然后再从 `sdkconfig.defaults.esp32` 获取。

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。

4.5.14 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。

您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 esp32c2 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。

4.5.15 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）。

4.5.16 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。

4.5.17 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)

# 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()
```

4.5.18 组件中使用预建库

还有一种情况是您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。

4.5.19 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)
```

(下页继续)

(续上页)

```
# 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。

4.5.20 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 intepreter 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` EMBED_TXTFILES 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`。

4.5.21 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。

4.5.22 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。

您可以通过 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。

4.5.23 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：

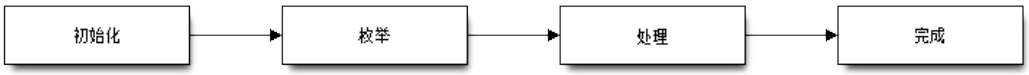


图 3: ESP-IDF Build System Process

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。

4.5.24 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*。

4.6 Core Dump

4.6.1 Overview

ESP-IDF provides support to generate core dumps on unrecoverable software errors. This useful technique allows post-mortem analysis of software state at the moment of failure. Upon the crash system enters panic state, prints some information and halts or reboots depending configuration. User can choose to generate core dump in order to analyse the reason of failure on PC later on. Core dump contains snapshots of all tasks in the system at the moment of failure. Snapshots include tasks control blocks (TCB) and stacks. So it is possible to find out what task, at what instruction (line of code) and what callstack of that task lead to the crash. It is also possible dumping variables content on demand if previously attributed accordingly. ESP-IDF provides special commands to help users to retrieve and analyse core dumps:

- idf.py coredump-info - prints crashed task's registers, callstack, list of available tasks in the system, memory regions and contents of memory stored in core dump (TCBs and stacks)
- idf.py coredump-debug - creates core dump ELF file and runs GDB debug session with this file. User can examine memory, variables and tasks states manually. Note that since not all memory is saved in core dump only values of variables allocated on stack will be meaningful

For more information about core dump internals see the - *Core dump internals*

4.6.2 Configurations

There are a number of core dump related configuration options which user can choose in project configuration menu (`idf.py menuconfig`).

Core dump data destination (Components -> Core dump -> Data destination)

- Save core dump to Flash (Flash)
- Print core dump to UART (UART)
- Disable core dump generation (None)

Core dump data format (Components -> Core dump -> Core dump data format)

- ELF format (Executable and Linkable Format file for core dump)
- Binary format (Basic binary format for core dump)

The ELF format contains extended features and allow to save more information about broken tasks and crashed software but it requires more space in the flash memory. This format of core dump is recommended for new software designs and is flexible enough to extend saved information for future revisions.

The Binary format is kept for compatibility reasons, it uses less space in the memory to keep data and provides better performance.

Core dump data integrity check (Components -> Core dump -> Core dump data integrity check)

- Use CRC32 for core dump integrity verification

Maximum number of tasks snapshots in core dump (Components -> Core dump -> Maximum number of tasks)

Delay before core dump is printed to UART (Components -> Core dump -> Delay before print to UART)

The value is in ms.

Handling of UART core dumps in IDF Monitor (Components -> Core dump -> Delay before print to UART)

The value is base64 encoded.

- Decode and show summary (info_corefile)
- Don’t decode

Reserved stack size (Components -> Core dump -> Reserved stack size)

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 above 800.

4.6.3 Save core dump to flash

When this option is selected core dumps are saved to special partition on flash. When using default partition table files which are provided with ESP-IDF it automatically allocates necessary space on flash, But if user wants to use its own layout file together with core dump feature it should define separate partition for core dump as it is shown below:

```
# Name,   Type, SubType, Offset,  Size
# Note: if you have increased the bootloader size, make sure to update the offsets␣
↪to avoid overlap
nvs,      data, nvs,     0x9000,  0x6000
phy_init, data, phy,     0xf000,  0x1000
factory,  app,  factory, 0x10000, 1M
coredump, data, coredump,,        64K
```

重要： If *Flash Encryption* is enabled on the device then please add *encrypted* flag to the coredump partition:

```
coredump, data, coredump,,        64K, encrypted
```

There are no special requirements for partition name. It can be chosen according to the user application needs, but partition type should be ‘data’ and sub-type should be ‘coredump’. Also when choosing partition size note that core dump data structure introduces constant overhead of 20 bytes and per-task overhead of 12 bytes. This overhead does not include size of TCB and stack for every task. So partition size should be at least 20 + max tasks number x (12 + TCB size + max task stack size) bytes.

The example of generic command to analyze core dump from flash is:

```
idf.py coredump-info
```

or

```
idf.py coredump-debug
```

4.6.4 Print core dump to UART

When this option is selected base64-encoded core dumps are printed on UART upon system panic. In this case user should save core dump text body to some file manually and then run the following command:

```
idf.py coredump-info -c </path/to/saved/base64/text>
```

or

```
idf.py coredump-debug -c </path/to/saved/base64/text>
```

Base64-encoded body of core dump will be between the following header and footer:

```
================ CORE DUMP START ================
<body of base64-encoded core dump, save it to file on disk>
================ CORE DUMP END ================
```

The `CORE DUMP START` and `CORE DUMP END` lines must not be included in core dump text file.

4.6.5 ROM Functions in Backtraces

It is possible situation that at the moment of crash some tasks or/and crashed task itself have one or more ROM functions in their callstacks. Since ROM is not part of the program ELF it will be impossible for GDB to parse such callstacks, because it tries to analyse functions’ prologues to accomplish that. In that case callstack printing will be broken with error message at the first ROM function. To overcome this issue, ROM ELF provided by Espressif is loaded automatically based on the target and its revision. More details about ROM ELFs can be found here.

4.6.6 Dumping variables on demand

Sometimes you want to read the last value of a variable to understand the root cause of a crash. Core dump supports retrieving variable data over GDB by attributing special notations declared variables.

Supported notations and RAM regions

- `COREDUMP_DRAM_ATTR` places variable into DRAM area which will be included into dump.

Example

1. In 项目配置菜单, enable *COREDUMP TO FLASH*, then save and exit.
2. In your project, create a global variable in DRAM area as such as:

```
// uint8_t global_var;
COREDUMP_DRAM_ATTR uint8_t global_var;
```

3. In main application, set the variable to any value and `assert(0)` to cause a crash.

```
global_var = 25;
assert(0);
```

4. Build, flash and run the application on a target device and wait for the dumping information.
5. Run the command below to start core dumping in GDB, where `PORT` is the device USB port:

```
idf.py coredump-debug
```

6. In GDB shell, type `p global_var` to get the variable content:

```
(gdb) p global_var
$1 = 25 '\031'
```

4.6.7 Running `idf.py coredump-info` and `idf.py coredump-debug`

`idf.py coredump-info --help` and `idf.py coredump-debug --help` commands can be used to get more details on usage

Related Documents

Anatomy of core dump image Core dump component can be configured to use old legacy binary format or the new ELF one. The ELF format is recommended for new designs. It provides more information about the CPU and memory state of a program at the moment when panic handler is entered. The memory state embeds a snapshot of all tasks mapped in the memory space of the program. The CPU state contains register values when the core dump has been generated. Core dump file uses a subset of the ELF structures to register these information. Loadable ELF segments are used for the memory state of the process while ELF notes (ELF.PT_NOTE) are used for process metadata (pid, registers, signal, ⋯). Especially, the CPU status is stored in a note with a special name and type (`CORE`, `NT_PRSTATUS type`).

Here is an overview of coredump layout:

Note: The format of image file showed on the above pictures represents current version of image and can be changed in future releases.

Overview of implementation The figure below describes some basic aspects related to implementation of core dump:

Note: The diagram above hide some details and represents current implementation of the core dump and can be changed later.

4.7 C++ Support

ESP-IDF is primarily written in C and provides C APIs. However, ESP-IDF supports development of applications in C++. This document covers various topics relevant to C++ development.

The following C++ features are supported:

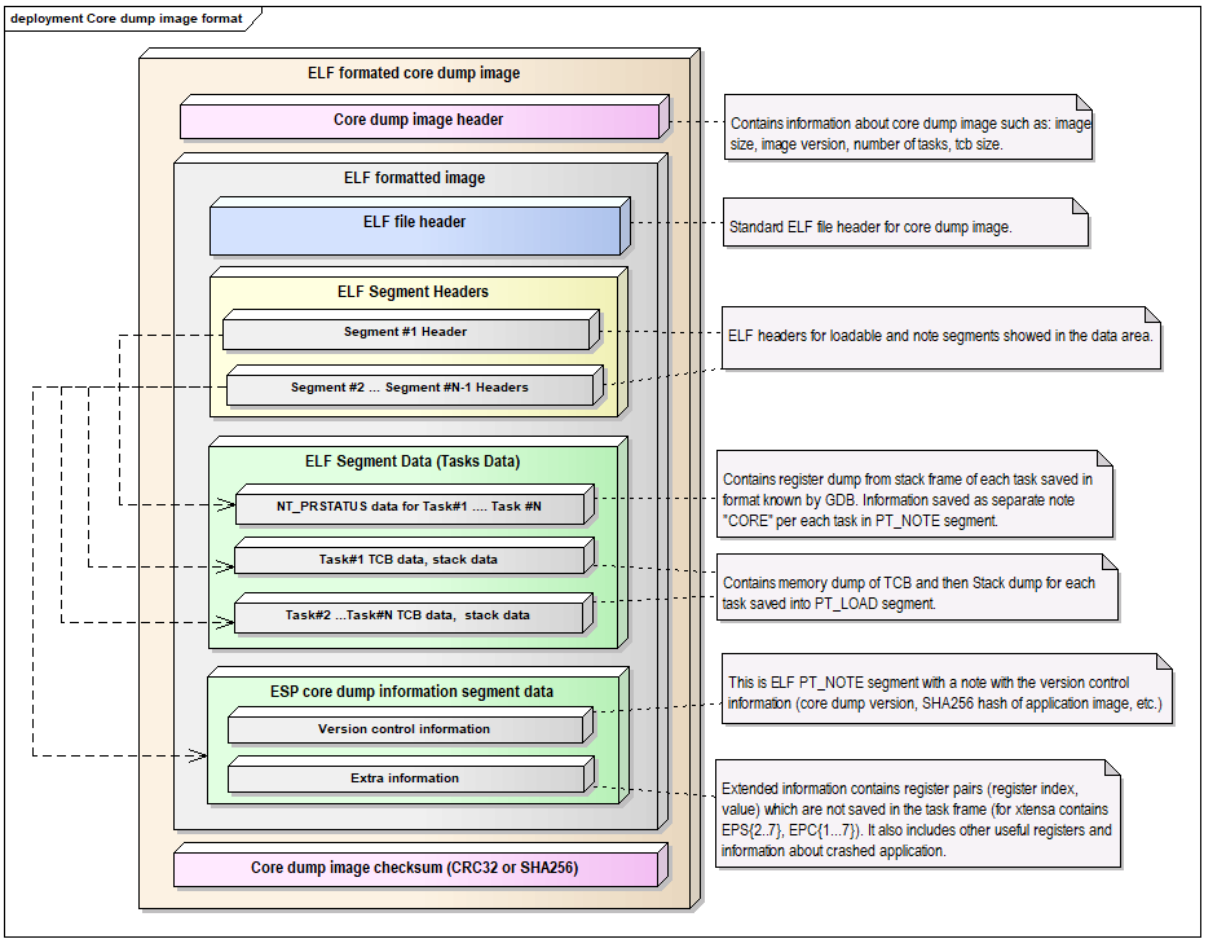


图 4: Core dump ELF image format

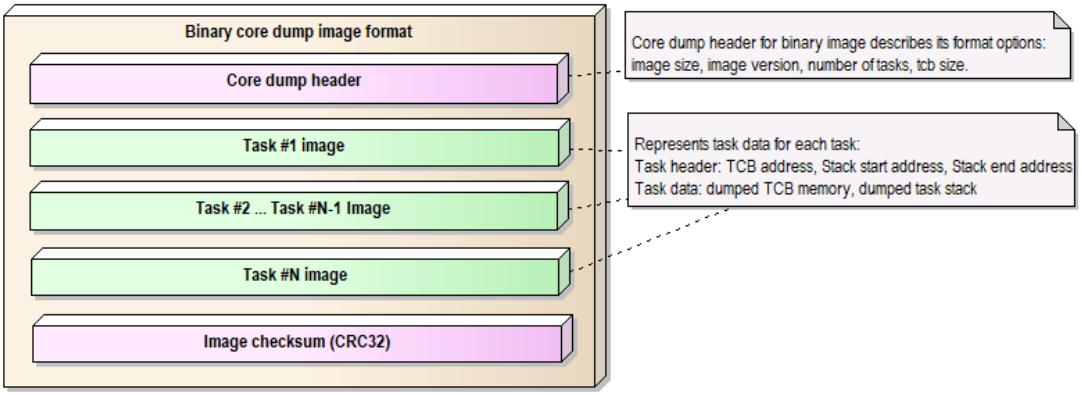


图 5: Core dump binary image format

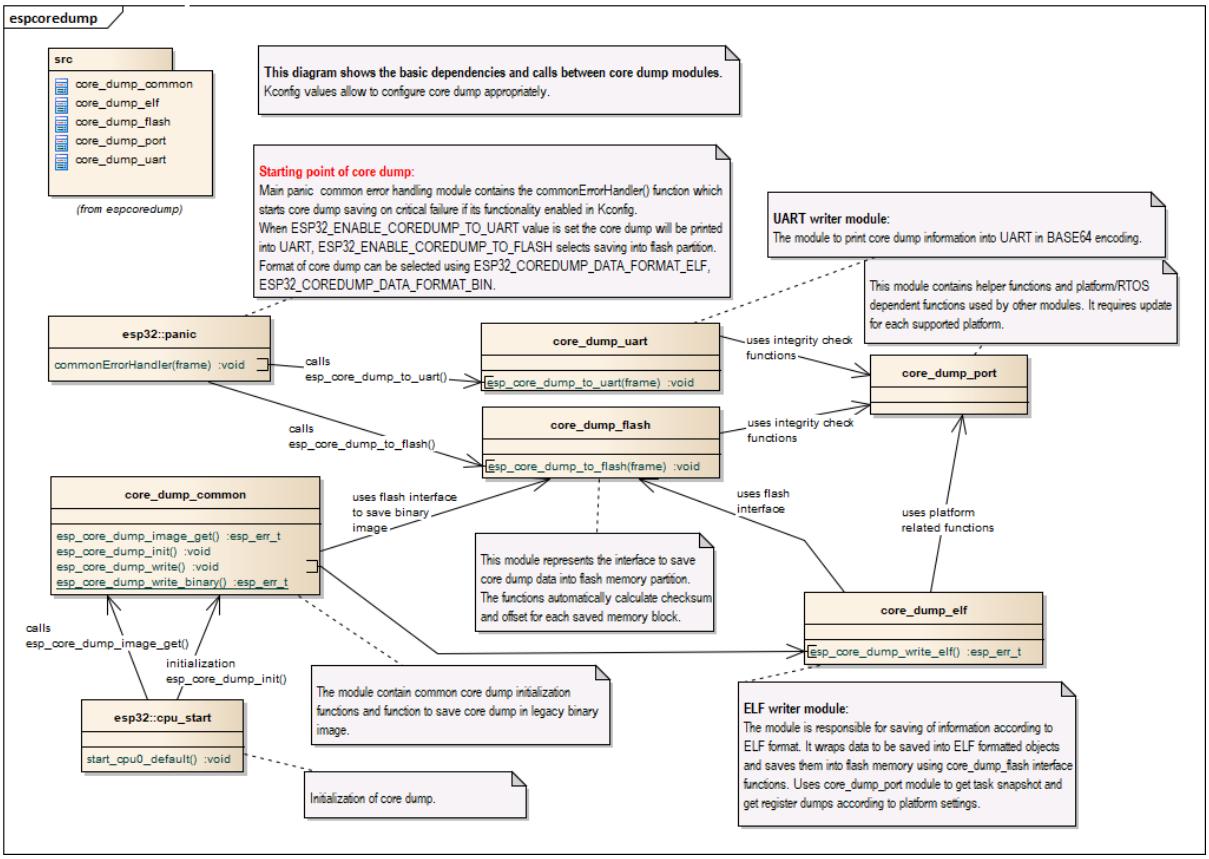


图 6: Core dump implementation overview

- *Exception handling*
- *C++ language standard*
- *Runtime Type Information (RTTI)*
- *Thread Local Storage* (`thread_local` keyword)
- All C++ features implemented by GCC, except for some *limitations*. See GCC documentation for details on features implemented by GCC.

4.7.1 esp-idf-cxx Component

esp-idf-cxx component provides higher-level C++ APIs for some of the ESP-IDF features. This component is available from the IDF Component Registry.

4.7.2 C++ language standard

By default, ESP-IDF compiles C++ code with C++23 language standard with GNU extensions (`-std=gnu++23`).

To compile the source code of a certain component using a different language standard, set the desired compiler flag in the component CMakeLists.txt file:

```
idf_component_register( ... )
target_compile_options(${COMPONENT_LIB} PRIVATE -std=gnu++11)
```

Use `PUBLIC` instead of `PRIVATE` if the public header files of the component also need to be compiled with the same language standard.

4.7.3 Multithreading

C++ threads, mutexes, and condition variables are supported. C++ threads are built on top of pthreads, which in turn wrap FreeRTOS tasks.

See cxx/pthread for an example of creating threads in C++.

4.7.4 Exception handling

Support for C++ Exceptions in ESP-IDF is disabled by default, but can be enabled using the *CONFIG_COMPILER_CXX_EXCEPTIONS* option.

If an exception is thrown, but there is no `catch` block, the program will be terminated by the `abort` function, and the backtrace will be printed. See *Fatal Errors* for more information about backtraces.

C++ Exceptions should *only* be used for exceptional cases, something happening unexpectedly and that is quite rare, e.g. an event that happens less frequently than 1 every 100 times. *Do not* use them for control flow (see also the section about resource usage below)! For more information on how to use C++ Exceptions, see the ISO C++ FAQ and CPP Core Guidelines.

See cxx/exceptions for an example of C++ exception handling.

C++ Exception Handling and Resource Usage

Enabling exception handling normally increases application binary size by a few KB.

Additionally, it may be necessary to reserve some amount of RAM for exception emergency pool. Memory from this pool will be used if it is not possible to allocate exception object from the heap. The amount of memory in the emergency pool can be set using the *CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE* variable. Some additional stack memory (around 200 bytes) will also be used if and only if a C++ Exception is actually thrown, because it requires calling some functions from the top of the stack to initiate exception handling.

The run time of code using C++ exceptions depends on what actually happens at run time. If no exception is thrown, the code tends to be somewhat faster since there is no need to check error codes. If an exception is thrown, the run time of the code that handles exceptions will be orders of magnitude slower than code returning an error code. This increase may or may not be significant, however, in the entire application, in particular if the error handling requires additional action, such as a user input or messaging to a cloud. But exception-throwing code should never be used in real-time critical code paths.

4.7.5 Runtime Type Information (RTTI)

Support for RTTI is disabled by default, but can be enabled using *CONFIG_COMPILER_CXX_RTTI* option.

Enabling this option compiles all C++ files with RTTI support enabled, which allows using `dynamic_cast` conversion and `typeid` operator. Enabling this option typically increases the binary size by tens of kB.

See cxx/rtti for an example of using RTTI in ESP-IDF.

4.7.6 Developing in C++

The following sections provide tips on developing ESP-IDF applications in C++.

Combining C and C++ code

When part of the application is developed in C and part in C++, it is important to understand the concept of language linkage.

In order for a C++ function to be callable from C code, it has to be both *declared* and *defined* with C linkage (`extern "C"`):

```
// declaration in the header file:
#ifdef __cplusplus
extern "C" {
#endif

void my_cpp_func(void);

#ifdef __cplusplus
}
#endif

// definition in a .cpp file:
extern "C" void my_cpp_func(void) {
    // ...
}
```

In order for a C function to be callable from C++, it has to be *declared* with C linkage:

```
// declaration in the header file:
#ifdef __cplusplus
extern "C" {
#endif

void my_c_func(void);

#ifdef __cplusplus
}
#endif

// definition in a .c file:
void my_c_func(void) {
    // ...
}
```

Defining `app_main` in C++

ESP-IDF expects the application entry point, `app_main`, to be defined with C linkage. When `app_main` is defined in a .cpp source file, it has to be designated as `extern "C"`:

```
extern "C" void app_main()
{
}
```

Designated initializers

Many of the ESP-IDF components use *configuration structures* as arguments to the initialization functions. ESP-IDF examples written in C routinely use designated initializers to fill these structures in a readable and a maintainable way.

C and C++ languages have different rules with regards to the designated initializers. For example, C++ language version C++23, currently the default in ESP-IDF, does not support out-of-order designated initialization, nested designated initialization, mixing of designated initializers and regular initializers, and designated initialization of arrays. Therefore, when porting ESP-IDF C examples to C++, some changes to the structure initializers may be necessary. See the C++ aggregate initialization reference for more details.

iostream

`iostream` functionality is supported in ESP-IDF, with a couple of caveats:

1. Normally ESP-IDF build process eliminates the unused code. However in the case of iostreams, simply including `<iostream>` header in one of the source files significantly increases the binary size (by about 200 kB).
2. By default, ESP-IDF uses a simple non-blocking implementation of the standard input stream (`stdin`). To get the usual behavior of `std::cin`, the application has to initialize the UART driver and enable the blocking mode as shown in common_components/protocol_examples_common/stdin_out.c.

4.7.7 Limitations

- Linker script generator doesn't support function level placements for functions with C++ linkage.
- Various section attributes (such as `IRAM_ATTR`) are ignored when used with template functions.
- Vtables are placed into Flash and are not accessible when the flash cache is disabled. Therefore, virtual function calls should be avoided in *IRAM-safe interrupt handlers*. Placement of Vtables cannot be adjusted using the linker script generator, yet.
- C++ filesystem (`std::filesystem`) features are not supported.

4.7.8 What to Avoid

Do not use `setjmp/longjmp` in C++! `longjmp` blindly jumps up the stack without calling any destructors, easily introducing undefined behavior and memory leaks. Use C++ exceptions instead, they will guarantee correctly calling destructors. If you cannot use C++ exceptions, use alternatives (except `setjmp/longjmp` themselves) such as simple return codes.

4.8 错误处理

4.8.1 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。

4.8.2 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。

4.8.3 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。

4.8.4 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* 。

4.8.5 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().

4.8.6 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。

4.8.7 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*.

4.8.8 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*.

4.8.9 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*.

4.8.10 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`.
    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。

4.8.11 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));
```

4.8.12 C++ 异常

请参考*Exception handling*。

4.9 严重错误

4.9.1 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。

4.9.2 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 (Illegal instruction). 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*。
- 调用动态 GDB Stub (`ESP_SYSTEM_GDBSTUB_RUNTIME`)
  启动 GDB 服务器，通过控制台 UART 接口与 GDB 进行通信。该选项允许用户在程序运行时对其进行调试、设置断点和改变其执行方式等，详细信息请参阅*GDB Stub*。

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-C2 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：

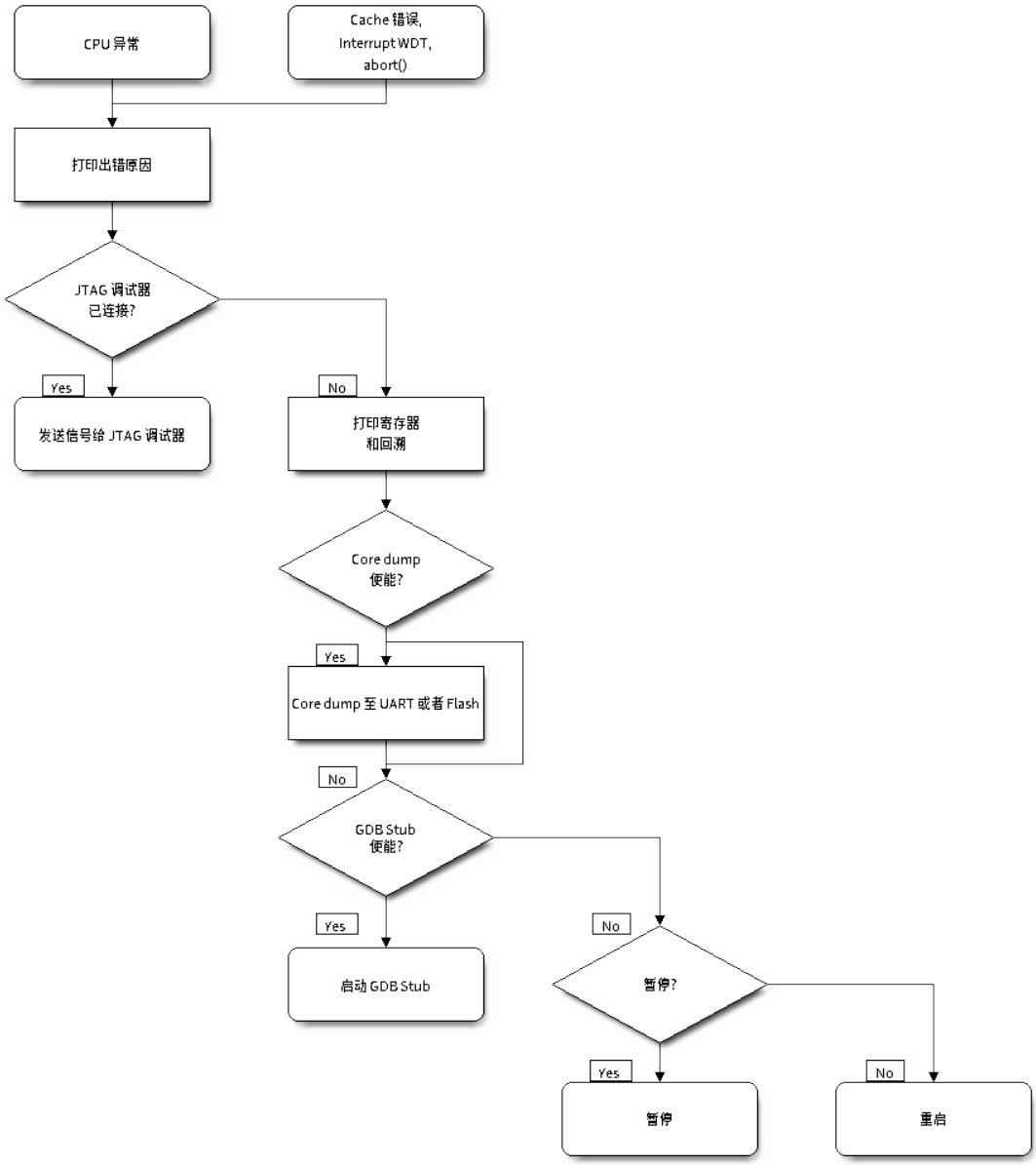


图 7: 紧急处理程序流程图（点击放大）

4.9.3 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。

4.9.4 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。

4.9.5 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 (RTCWDT_RTC_RST)
```

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”。

4.9.6 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 。

Illegal instruction

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-C2 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 Illegal instruction 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 Illegal instruction 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

当执行 EBREAK 指令时，会发生此 CPU 异常。

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*。

4.9.7 其他严重错误

掉电

ESP32-C2 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

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

4.10 flash 加密

本文档旨在引导用户快速了解 ESP32-C2 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。

4.10.1 概述

flash 加密功能用于加密与 ESP32-C2 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。

启用 flash 加密后，系统将默认加密下列类型的 flash 数据：

- 固件引导加载程序
- 分区表
- 所有“app”类型的分区

其他类型的数据将视情况进行加密：

- 任何在分区表中标有“加密”标志的分区。详情请见加密分区标志。
- 如果启用了安全启动，则可以加密安全启动引导程序摘要（见下文）。

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-C2 更新。在使用 flash 加密功能前，请务必阅读本文档了解其影响。

4.10.2 相关 eFuses

flash 加密操作由 ESP32-C2 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
`XTS_KEY_LENGTH_256`	控制用于得出最终 256 位 AES 密钥的 eFuse 比特的实际数量。可能的值：`0` 使用 eFuse 块的全部 256 位作为密钥，`1` 使用 eFuse 块的低 128 位作为密钥（高 128 位保留给安全启动密钥）。对于 128 位选项，最终的 AES 密钥会以 SHA256 (EFUSE_KEY0_FE_128BIT) 的形式得出。	1
`BLOCK_KEY0`	AES 密钥存储	256 位或 128 位密钥块
`DIS_DOWNLOAD_MANUAL_ENCRYPT`	设置后，则在下载引导模式时禁用 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 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-C2 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`。

重要： ESP32-C2 具有安全启动和 flash 加密两个密钥，但仅有一个 eFuse 密钥块。由于 eFuse 密钥块仅支持一次烧录，故应将密钥同时同批进行烧录。请勿单独启用“安全启动”或“flash 加密”，否则在 eFuse 密钥块随后的写入中将返回错误。

4.10.3 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-C2 技术参考手册。
3. 固件的引导加载程序使用 RNG（随机数生成）模块生成 256 位或 128 位密钥（具体位数取决于*Size of generated AES-XTS key*），然后将其写入 *BLOCK_KEY0* eFuse。同时，根据所选选项，软件对 XTS_KEY_LENGTH_256 进行更新。由于 BLOCK_KEY0 eFuse 已设置编写和读取保护位，故无法通过软件访问密钥。flash 加密操作完全在硬件中完成，无法通过软件访问密钥。若使用 128 位 flash 加密密钥，则整个 eFuse 密钥块都受写保护，但只有低 128 位受读保护，高 128 位是可读的，以满足安全启动的需要。如果 flash 加密的密钥是 256 位，那么 XTS_KEY_LENGTH_256 为 1，否则为 0。为防止意外将 eFuse 从 0 改为 1，RELEASE 模式中设置了一个写保护位。
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，默认情况下，固件引导加载程序设置 DIS_DOWNLOAD_ICACHE、DIS_PAD_JTAG 和 DIS_DIRECT_BOOT eFuse 位。
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 位以及 DIS_DOWNLOAD_MANUAL_ENCRYPT。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。

4.10.4 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-C2 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-C2 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. 确保您的 ESP32-C2 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-C2 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 *AES-XTS* 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-C2 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-C2 首次启动时的样例输出：

```
ESP-ROM:esp8684-api1-20211015
Build:Oct 15 2021
rst:0x1 (POWERON),boot:0xc (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fcd6190,len:0x2a84
load:0x403ae000,len:0x830
load:0x403b0000,len:0x42a0
entry 0x403ae000
I (21) boot: ESP-IDF v5.0-dev-2717-g0d1e015-dirty 2nd stage bootloader
I (21) boot: compile time 19:36:15
I (21) boot: chip revision: 0
I (24) boot.esp32c2: MMU Page Size  : 64K
I (29) boot.esp32c2: SPI Speed      : 60MHz
I (34) boot.esp32c2: SPI Mode       : DIO
I (39) boot.esp32c2: SPI Flash Size : 2MB
I (43) boot: Enabling RNG early entropy source...
I (49) boot: Partition Table:
I (52) boot: ## Label            Usage          Type ST Offset   Length
I (60) boot:  0 nvs              WiFi data        01 02 00010000 00006000
I (67) boot:  1 phy_init         RF data          01 01 00016000 00001000
I (75) boot:  2 factory          factory app      00 00 00020000 00100000
I (82) boot: End of partition table
I (86) esp_image: segment 0: paddr=00020020 vaddr=3c010020 size=06858h ( 26712) map
I (101) esp_image: segment 1: paddr=00026880 vaddr=3fca9a60 size=01430h (  5168)␣
↪load
I (104) esp_image: segment 2: paddr=00027cb8 vaddr=40380000 size=08360h ( 33632)␣
↪load
I (120) esp_image: segment 3: paddr=00030020 vaddr=42000020 size=0f67ch ( 63100)␣
↪map
I (134) esp_image: segment 4: paddr=0003f6a4 vaddr=40388360 size=01700h (  5888)␣
↪load
I (139) boot: Loaded app from partition at offset 0x20000
I (139) boot: Checking flash encryption...
I (142) efuse: Batch mode of writing fields is enabled
```

(下页继续)

(续上页)

```
I (148) flash_encrypt: Generating new flash encryption key...
I (155) efuse: Writing EFUSE_BLK_KEY0 with purpose 1
W (161) flash_encrypt: Not disabling UART bootloader encryption
I (167) flash_encrypt: Disable UART bootloader cache...
I (175) flash_encrypt: Disable JTAG...
I (190) efuse: BURN BLOCK3
I (195) efuse: BURN BLOCK3 - OK (write block == read block)
I (204) efuse: BURN BLOCK0
I (208) efuse: BURN BLOCK0 - OK (write block == read block)
I (213) efuse: Batch mode. Prepared fields are committed
I (219) esp_image: segment 0: paddr=00000020 vaddr=3fcd6190 size=02a84h ( 10884)
I (229) esp_image: segment 1: paddr=00002aac vaddr=403ae000 size=00830h (  2096)
I (236) esp_image: segment 2: paddr=000032e4 vaddr=403b0000 size=042a0h ( 17056)
I (679) flash_encrypt: bootloader encrypted successfully
I (731) flash_encrypt: partition table encrypted and loaded successfully
I (731) esp_image: segment 0: paddr=00020020 vaddr=3c010020 size=06858h ( 26712)␣
↪map
I (741) esp_image: segment 1: paddr=00026880 vaddr=3fca9a60 size=01430h (  5168)
I (745) esp_image: segment 2: paddr=00027cb8 vaddr=40380000 size=08360h ( 33632)
I (759) esp_image: segment 3: paddr=00030020 vaddr=42000020 size=0f67ch ( 63100)␣
↪map
I (774) esp_image: segment 4: paddr=0003f6a4 vaddr=40388360 size=01700h (  5888)
I (776) flash_encrypt: Encrypting partition 2 at offset 0x20000 (length 0x100000)..
↪.
I (6429) flash_encrypt: Done encrypting
I (6429) efuse: BURN BLOCK0
I (6432) efuse: BURN BLOCK0 - OK (all write block bits are set)
I (6438) flash_encrypt: Flash encryption completed
I (6443) 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:esp8684-api1-20211015
Build:Oct 15 2021
rst:0x3 (RTC_SW_SYS_RST),boot:0xc (SPI_FAST_FLASH_BOOT)
Saved PC:0x403b0f9e
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fcd6190,len:0x2a84
load:0x403ae000,len:0x830
load:0x403b0000,len:0x42a0
entry 0x403ae000
I (23) boot: ESP-IDF v5.0-dev-2717-g0d1e015-dirty 2nd stage bootloader
I (23) boot: compile time 19:36:15
I (23) boot: chip revision: 0
I (27) boot.esp32c2: MMU Page Size  : 64K
I (32) boot.esp32c2: SPI Speed      : 60MHz
I (36) boot.esp32c2: SPI Mode       : DIO
I (41) boot.esp32c2: SPI Flash Size : 2MB
I (46) boot: Enabling RNG early entropy source...
I (51) boot: Partition Table:
I (55) boot: ## Label            Usage          Type ST Offset   Length
I (62) boot:  0 nvs              WiFi data        01 02 00010000 00006000
I (70) boot:  1 phy_init         RF data          01 01 00016000 00001000
I (77) boot:  2 factory          factory app      00 00 00020000 00100000
I (85) boot: End of partition table
I (89) esp_image: segment 0: paddr=00020020 vaddr=3c010020 size=06858h ( 26712) map
I (103) esp_image: segment 1: paddr=00026880 vaddr=3fca9a60 size=01430h (  5168)␣
↪load
I (107) esp_image: segment 2: paddr=00027cb8 vaddr=40380000 size=08360h ( 33632)␣
↪load
```

(下页继续)

(续上页)

```
I (123) esp_image: segment 3: paddr=00030020 vaddr=42000020 size=0f67ch ( 63100)␣
↪map
I (138) esp_image: segment 4: paddr=0003f6a4 vaddr=40388360 size=01700h (  5888)␣
↪load
I (143) boot: Loaded app from partition at offset 0x20000
I (143) boot: Checking flash encryption...
I (146) flash_encrypt: flash encryption is enabled (1 plaintext flashes left)
I (154) boot: Disabling RNG early entropy source...
I (171) cpu_start: Pro cpu up.
I (179) cpu_start: Pro cpu start user code
I (179) cpu_start: cpu freq: 120000000 Hz
I (179) cpu_start: Application information:
I (182) cpu_start: Project name:     hello_world
I (187) cpu_start: App version:      v5.0-dev-2717-g0d1e015-dirty
I (194) cpu_start: Compile time:     May 20 2022 19:35:55
I (200) cpu_start: ELF file SHA256:  04592ac3c9304cdc...
I (206) cpu_start: ESP-IDF:          v5.0-dev-2717-g0d1e015-dirty
I (213) heap_init: Initializing. RAM available for dynamic allocation:
I (220) heap_init: At 3FCABCB0 len 0002C350 (176 KiB): D/IRAM
I (226) heap_init: At 3FCD8000 len 0000742C (29 KiB): STACK/DRAM
I (234) spi_flash: detected chip: generic
I (238) spi_flash: flash io: dio
W (242) flash_encrypt: Flash encryption mode is DEVELOPMENT (not secure)
I (249) sleep: Configure to isolate all GPIO pins in sleep state
I (256) sleep: Enable automatic switching of GPIO sleep configuration
W (263) INT_WDT: ESP32-C2 only has one timer group
I (268) cpu_start: Starting scheduler.
Hello world!
This is esp32c2 chip with 1 CPU core(s), WiFi/BLE, silicon revision 0, 2MB␣
↪external flash
Minimum free heap size: 195052 bytes

FLASH_CRYPT_CNT eFuse value is 1
Flash encryption feature is enabled in DEVELOPMENT mode
```

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-C2 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。

备注： 请注意，ESP32-C2 只有一个 eFuse 密钥块，同时用于安全启动和 flash 加密密钥。因此，如果使用了安全启动密钥，则主机生成的 flash 加密密钥必须与安全启动密钥一起写入，否则将无法使用安全启动。

使用主机生成的密钥需完成以下步骤：

1. 确保您的 ESP32-C2 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-C2 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 *AES-XTS 密钥大小* 是 AES-128（256 位密钥）：

```
espsecure.py generate_flash_encryption_key my_flash_encryption_key.bin
```

(下页继续)

(续上页)

```
或者如果 :ref:`生成的 AES-XTS 密钥大小 <CONFIG_SECURE_FLASH_ENCRYPTION_KEYSIZE>`␣
↪是由 128 位导出的 AES-128 密钥（SHA256（128 位））：

.. code-block:: bash

    espsecure.py generate_flash_encryption_key --keylen 128 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**。

   对于 AES-128（256 位密钥）- `XTS_AES_128_KEY`（`XTS_KEY_LENGTH_256` eFuse 将被烧录为 1）：

   ```
   espefuse.py --port PORT burn_key BLOCK_KEY0 flash_encryption_key256.bin␣
   ↪XTS_AES_128_KEY
   ```

   对于由 128 位导出的 AES-128 密钥（SHA256（128 位））- `XTS_AES_128_KEY_DERIVED_FROM_128_EFUSE_BITS`。flash 加密密钥会被写入 eFuse BLOCK_KEY0 的低位，留出高 128 位以支持软件读取。如小节 同时烧录两个密钥所示，在 espefuse 工具的特殊模式下，您可以使用任意 espefuse 命令来写入数据。

   ```
   espefuse.py --port PORT burn_key BLOCK_KEY0 flash_encryption_key128.bin␣
   ↪XTS_AES_128_KEY_DERIVED_FROM_128_EFUSE_BITS
   ```

   同时烧录两个密钥（安全启动和 flash 加密）：

   ```
   espefuse.py --port PORT --chip esp32c2 burn_key_digest secure_boot_
   ↪signing_key.pem \
                                          burn_key BLOCK_KEY0 flash_
   ↪encryption_key128.bin XTS_AES_128_KEY_DERIVED_FROM_128_EFUSE_BITS
   ```

   如果未烧录密钥并在启用 flash 加密后启动设备，ESP32-C2 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-C2 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. 确保您的 ESP32-C2 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-C2 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-C2 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。

4.10.5 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-C2 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
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
```

4.10.6 ESP32-C2 flash 加密状态

1. 确保您的 ESP32-C2 设备有相关 *eFuses* 中所示的 flash 加密 eFuse 的默认设置。

要检查您的 ESP32-C2 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 烧录到您的设备上。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
```

4.10.7 在加密的 flash 中读写数据

ESP32-C2 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。

4.10.8 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*。

4.10.9 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-C2 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-C2，flash 加密应处于关闭状态，引导加载程序将正常启动。

4.10.10 flash 加密的要点

- 使用 XTS-AES-128 加密 flash。flash 加密密钥为 128 位或 256 位，存储于芯片内部的 `BLOCK_KEY0` eFuse 中，并（默认）受保护，防止软件访问。
- 通过 ESP32-C2 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-C2 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。

4.10.11 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，推荐您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，请检查您使用的存储方式（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。

4.10.12 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。

4.10.13 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。

关于分区加密您还需要了解以下信息：

- 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_DOWNLOAD_ICACHE` 在 UART 引导加载程序模式下运行时禁止整个 MMU flash 缓存。
- `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-C2 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。

4.10.14 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-C2 使用 XTS-AES 块密码模式进行 flash 加密，密钥大小为 256 位。如果 128 位的密钥存储于 eFuse 密钥块中，那么最终的 256 位 AES 密钥将以 SHA256(EFUSE_KEY0_FE_128BIT) 的形式获得。
- XTS-AES 是一种专门为光盘加密设计的块密码模式，它弥补了其他潜在模式如 AES-CTR 在此使用场景下的不足。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 `BLOCK_KEY0` 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()`。

4.11 Hardware Abstraction

Hardware abstraction in ESP-IDF are a group of API that allow users to control peripherals at differing levels of abstraction, as opposed to interfacing with hardware using only the ESP-IDF drivers. ESP-IDF Hardware abstraction will likely be useful for users writing high performance bare-metal drivers, or for those attempting to port an ESP chip to another platform.

This guide is split into the following sections:

1. *Architecture*
2. *LL (Low Level) Layer*
3. *HAL (Hardware Abstraction Layer)*

警告: Hardware abstraction API (excluding the driver and `xxx_types.h`) should be considered an experimental feature, thus cannot be considered public API. Hardware abstraction API do not adhere to the API name changing restrictions of ESP-IDF's versioning scheme. In other words, it is possible that Hardware Abstraction API may change in between non-major release versions.

备注: Although this document mainly focuses on hardware abstraction of peripherals (e.g., UART, SPI, I2C), certain layers of hardware abstraction extend to other aspects of hardware as well (e.g., some of the CPU's features are partially abstracted).

4.11.1 Architecture

Hardware abstraction in ESP-IDF is comprised of the following layers, ordered from low level (closer to hardware) to high level (further away from hardware) of abstraction.

- Low Level (LL) Layer
- Hardware Abstraction Layer (HAL)
- Driver Layers

The LL Layer, and HAL are entirely contained within the `hal` component. Each layer is dependent on the layer below it (i.e, driver depends on HAL, HAL depends on LL, LL depends on the register header files).

For a particular peripheral `xxx`, its hardware abstraction will generally consist of the header files described in the table below. Files that are **Target Specific** will have a separate implementation for each target (i.e., a separate copy for each chip). However, the `#include` directive will still be target-independent (i.e., will be the same for different targets) as the build system will automatically include the correct version of the header and source files.

表 2: Hardware Abstraction Header Files

Include Directive	Target Specific	Description
`#include 'soc/xxx_caps.h"`	Y	This header contains a list of C macros specifying the various capabilities of the ESP32-C2' s peripheral `xxx`. Hardware capabilities of a peripheral include things such as the number of channels, DMA support, hardware FIFO/buffer lengths, etc.
`#include "soc/xxx_struct.h"` `#include "soc/xxx_reg.h"`	Y	The two headers contain a representation of a peripheral' s registers in C structure and C macro format respectively. Users can operate a peripheral at the register level via either of these two header files.
`#include "soc/xxx_pins.h"`	Y	If certain signals of a peripheral are mapped to a particular pin of the ESP32-C2, their mappings are defined in this header as C macros.
`#include "soc/xxx_periph.h"`	N	This header is mainly used as a convenience header file to automatically include `xxx_caps.h`, `xxx_struct.h`, and `xxx_reg.h`.
`#include "hal/xxx_types.h`	N	This header contains type definitions and macros that are shared among the LL, HAL, and driver layers. Moreover, it is considered public API thus can be included by the application level. The shared types and definitions usually related to non-implementation specific concepts such as the following: • Protocol related types/macros such a frames, modes, common bus speeds, etc. • Features/characteristics of an `xxx` peripheral that are likely to be present on any implementation (implementation-independent) such as channels, operating modes, signal amplification or attenuation intensities, etc.
`#include "hal/xxx_ll.h"`	Y	This header contains the Low Level (LL) Layer of hardware abstraction. LL Layer API are primarily used to abstract away register operations into readable functions.
`#include "hal/xxx_hal.h"`	Y	The Hardware Abstraction Layer (HAL) is used to abstract away peripheral operation steps into functions (e.g., reading a buffer, starting a transmission, handling an event, etc). The HAL is built on top of the LL Layer.
`#include "driver/xxx.h"`	N	The driver layer is the highest level of ESP-IDF' s hardware abstraction. Driver layer API are meant to be called from ESP-IDF applications, and internally utilize OS primitives. Thus, driver layer API are event-driven, and can used in a multi-threaded environment.

4.11.2 LL (Low Level) Layer

The primary purpose of the LL Layer is to abstract away register field access into more easily understandable functions. LL functions essentially translate various in/out arguments into the register fields of a peripheral in the form of get/set functions. All the necessary bit shifting, masking, offsetting, and endianness of the register fields should be handled by the LL functions.

```
//Inside xxx_ll.h

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;
    // Set clock select field
    hw->clk_div_reg.divider = clock_divider >> 4;
    // Set clock divider field
```

(下页继续)

(续上页)

```
    hw->config.clk_sel = (source_clk == XXX_SCLK_APB) ? 0 : 1;
}

static inline uint32_t xxx_ll_get_rx_byte_count(xxx_dev_t *hw) {
    return hw->status_reg.rx_cnt;
}
```

The code snippet above illustrates typical LL functions for a peripheral `xxx`. LL functions typically have the following characteristics:

- All LL functions are defined as `static inline` so that there is minimal overhead when calling these functions due to compiler optimization. These functions are not guaranteed to be inlined by the compiler, so any LL function that will be called when the cache is disabled (e.g. from an IRAM ISR context) should be marked with `__attribute__((always_inline))`.
- The first argument should be a pointer to a `xxx_dev_t` type. The `xxx_dev_t` type is a structure representing the peripheral's registers, thus the first argument is always a pointer to the starting address of the peripheral's registers. Note that in some cases where the peripheral has multiple channels with identical register layouts, `xxx_dev_t *hw` may point to the registers of a particular channel instead.
- LL functions should be short and in most cases are deterministic. In other words, the worst case runtime of the LL function can be determined at compile time. Thus, any loops in LL functions should be finite bounded; however, there are currently a few exceptions to this rule.
- LL functions are not thread safe, it is the responsibility of the upper layers (driver layer) to ensure that registers or register fields are not accessed concurrently.

4.11.3 HAL (Hardware Abstraction Layer)

The HAL layer models the operational process of a peripheral as a set of general steps, where each step has an associated function. For each step, the details of a peripheral's register implementation (i.e., which registers need to be set/read) are hidden (abstracted away) by the HAL. By modeling peripheral operation as a set of functional steps, any minor hardware implementation differences of the peripheral between different targets or chip versions can be abstracted away by the HAL (i.e., handled transparently). In other words, the HAL API for a particular peripheral will remain mostly the same across multiple targets/chip versions.

The following HAL function examples are selected from the Watchdog Timer HAL as each function maps to one of the steps in a WDT's operation life cycle, thus illustrating how a HAL abstracts a peripheral's operation into functional steps.

```
// Initialize one of the WDTs
void wdt_hal_init(wdt_hal_context_t *hal, wdt_inst_t wdt_inst, uint32_t prescaler,␣
↪bool enable_intr);

// Configure a particular timeout stage of the WDT
void wdt_hal_config_stage(wdt_hal_context_t *hal, wdt_stage_t stage, uint32_t␣
↪timeout, wdt_stage_action_t behavior);

// Start the WDT
void wdt_hal_enable(wdt_hal_context_t *hal);

// Feed (i.e., reset) the WDT
void wdt_hal_feed(wdt_hal_context_t *hal);

// Handle a WDT timeout
void wdt_hal_handle_intr(wdt_hal_context_t *hal);

// Stop the WDT
void wdt_hal_disable(wdt_hal_context_t *hal);

// De-initialize the WDT
void wdt_hal_deinit(wdt_hal_context_t *hal);
```

HAL functions will generally have the following characteristics:

- The first argument to a HAL function has the `xxx_hal_context_t *` type. The HAL context type is used to store information about a particular instance of the peripheral (i.e. the context instance). A HAL context is initialized by the `xxx_hal_init()` function and can store information such as the following:
  - The channel number of this instance
  - Pointer to the peripheral's (or channel's) registers (i.e., a `xxx_dev_t *` type)
  - Information about an ongoing transaction (e.g., pointer to DMA descriptor list in use)
  - Some configuration values for the instance (e.g., channel configurations)
  - Variables to maintain state information regarding the instance (e.g., a flag to indicate if the instance is waiting for transaction to complete)
- HAL functions should not contain any OS primitives such as queues, semaphores, mutexes, etc. All synchronization/concurrency should be handled at higher layers (e.g., the driver).
- Some peripherals may have steps that cannot be further abstracted by the HAL, thus will end up being a direct wrapper (or macro) for an LL function.
- Some HAL functions may be placed in IRAM thus may carry an `IRAM_ATTR` or be placed in a separate `xxx_hal_iram.c` source file.

4.12 JTAG 调试

本文将介绍如何安装 ESP32-C2 的 OpenOCD 调试环境，以及如何使用 GDB 来调试 ESP32-C2 的应用程序。本文结构如下：

引言 介绍本指南主旨。
工作原理 介绍 ESP32-C2、JTAG（Joint Test Action Group）接口、OpenOCD 和 GDB 如何相互连接，从而实现 ESP32-C2 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-C2* 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 如果您不熟悉 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-C2 时的注意事项和补充内容。

4.12.1 引言

乐鑫已完成 OpenOCD 移植，以支持 ESP32-C2 处理器和多核 FreeRTOS 架构（大多数 ESP32-C2 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-C2 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。

4.12.2 工作原理

通过 JTAG（Joint Test Action Group）接口使用 OpenOCD 调试 ESP32-C2 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-C2** 目标的 **JTAG 适配器**，如下图“Application Loading and Monitoring”标志所示。

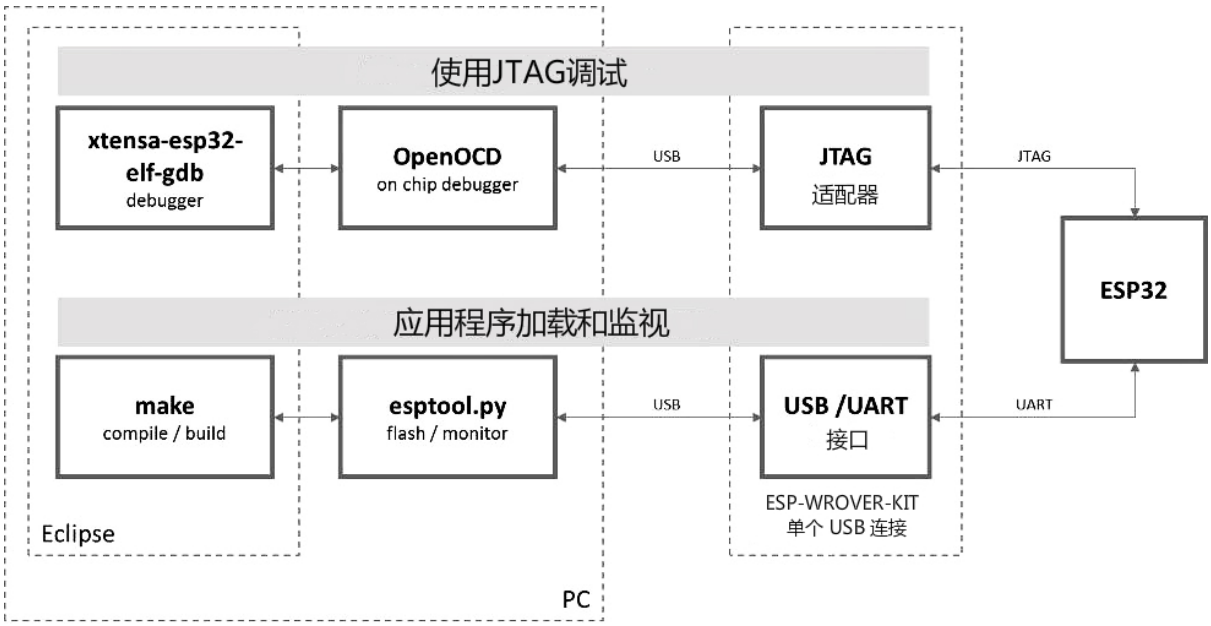


图 8: JTAG 调试 - 概述图

“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-C2 上，以及监视来自 ESP32-C2 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*。

4.12.3 选择 JTAG 适配器

如果您想使用单独的 JTAG 适配器，请确保其与 ESP32-C2 的电平电压和 OpenOCD 软件都兼容。ESP32-C2 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-C2 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-C2 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-C2 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。您也可以选择将 SRST 信号线连接到 ESP32-C2 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-C2 的 JTAG 管脚上。

4.12.4 安装 OpenOCD

如果您已经按照快速入门 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您的开发系统中。在设置开发环境 结束后，您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  v0.10.0-esp32-20190708 (2019-07-08-11:04)
Licensed under GNU GPL v2
For bug reports, read
    https://openocd.org/doc/doxygen/bugs.html
```

您还可以检查 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，则表明您已经正确安装 OpenOCD。

如果无法执行上述步骤，请再次阅读快速入门手册，参考设置安装工具 章节。

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。

4.12.5 配置 ESP32-C2 目标板

OpenOCD 安装完成后就可以配置 ESP32-C2 目标（即带 JTAG 接口的 ESP32-C2 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-C2 板，请参考以下两种情况。

配置其他 JTAG 接口

关于适配 OpenOCD 和 ESP32-C2 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。

配置硬件

1. 找到 JTAG 接口和 ESP32-C2 板上需要相互连接并建立通信的所有管脚或信号。

表 3: ESP32-C2 pins and JTAG signals

ESP32-C2 Pin	JTAG Signal
MTDO / GPIO7	TDO
MTDI / GPIO5	TDI
MTCK / GPIO6	TCK
MTMS / GPIO4	TMS

2. 检查 ESP32-C2 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-C2 和 JTAG 接口上的管脚或信号。

配置驱动 您可能还需要安装软件驱动，才能使 JTAG 在计算机上正常工作，请参阅您所使用的 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-C2 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/esp32c2-ftdi.cfg
```

备注： 上述命令中 `-f` 选项后跟的配置文件专用于 ESP32-C2 development board with ESP-Prog。基于具体使用的硬件，您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*。

现在您应该可以看到如下输出（此日志来自 ESP32-C2 development board with ESP-Prog）：

```
user-name@computer-name:~/esp/esp-idf$ openocd -f board/esp32c2-ftdi.cfg
Open On-Chip Debugger v0.11.0-esp32-20221026 (2022-10-26-14:48)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 20000 kHz

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Warn : libusb_detach_kernel_driver() failed with LIBUSB_ERROR_ACCESS, trying to␣
↪continue anyway
Info : ftdi: if you experience problems at higher adapter clocks, try the command
↪"ftdi tdo_sample_edge falling"
Info : clock speed 20000 kHz
Info : JTAG tap: esp32c2.cpu tap/device found: 0x0000cc25 (mfg: 0x612 (Espressif␣
↪Systems), part: 0x000c, ver: 0x0)
Info : datacount=2 progbufsize=16
Info : Examined RISC-V core; found 1 harts
Info :  hart 0: XLEN=32, misa=0x40101104
Info : starting gdb server for esp32c2 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/esp32c2-ftdi.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-C2 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-C2 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，您还可以使用 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/esp32c2-ftdi.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）

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。

4.12.6 启动调试器

ESP32-C2 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。

4.12.7 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-C2 目标板设置并加载 get-started/blink 至 ESP32-C2 中。

4.12.8 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。如果您想要更快速地构建，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-C2 的 OpenOCD 源码可以从乐鑫官方 GitHub 获取，网址为 https://github.com/espressif/openocd-esp32。您可以在 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。

您也可以调用 `make install`，将其复制到指定位置。

- 您可以在配置 OpenOCD 时指定这一位置，也可以在调用 `make install` 前设置 `export DESTDIR="/custom/install/dir"`。
- 如果您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。
- 如果您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，您可以将其复制到 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-C2* 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-C2 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。
- 如果您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-C2 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-C2 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。
- 如果您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-C2 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/esp32c2-ftdi.cfg
```

Windows 用户的示例如下:

```
src\openocd -f board/esp32c2-ftdi.cfg
```

4.12.9 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-C2 调试器支持 2 个硬件断点和 64 个软件断点。硬件断点是由 ESP32-C2 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面）。ESP32-C2 还支持 2 个观察点，所以可以观察 2 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-C2 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/esp32c2-ftdi.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（两个中可用的一个），这样就可以跳过进入子程序内部的细节。如果这两个断点已经用在代码的其它位置，那么 `next` 命令将不起作用。在这种情况下，请删掉一个断点以使其中一个变得可用。当两个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 2` —限制 GDB 仅使用 ESP32-C2 支持的两个硬件观察点。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。
- `flushregs` —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 `flushregs` 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-C2 可以使用的配置文件如下表所示:

表 4: OpenOCD configuration files for ESP32-C2

Name	Description
board/esp32c2-ftdi.cfg	Board configuration file for ESP32-C2 debug through an ESP-Prog compatible FTDI, includes target and adapter configuration.
target/esp32c2.cfg	ESP32-C2 target configuration file. Can be used together with one of the interface/ configuration files.
interface/ftdi/esp32_devkitj_v1.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 5: 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。

复位 ESP32-C2 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 如果除了 ESP32-C2 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-C2 JTAG 使用以下管脚:

表 6: ESP32-C2 pins and JTAG signals

ESP32-C2 Pin	JTAG Signal
MTDO / GPIO7	TDO
MTDI / GPIO5	TDI
MTCK / GPIO6	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。

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/esp32c2-ftdi.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/esp32c2-ftdi.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。

4.12.10 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： 建议您首先通过*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。

标准的 Eclipse 安装流程默认安装调试功能，另外您还可以使用插件来调试，比如 “GDB Hardware Debugging”。这个插件用起来非常方便，本指南会详细介绍该插件的使用方法。

首先，打开 Eclipse 并转到 “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。

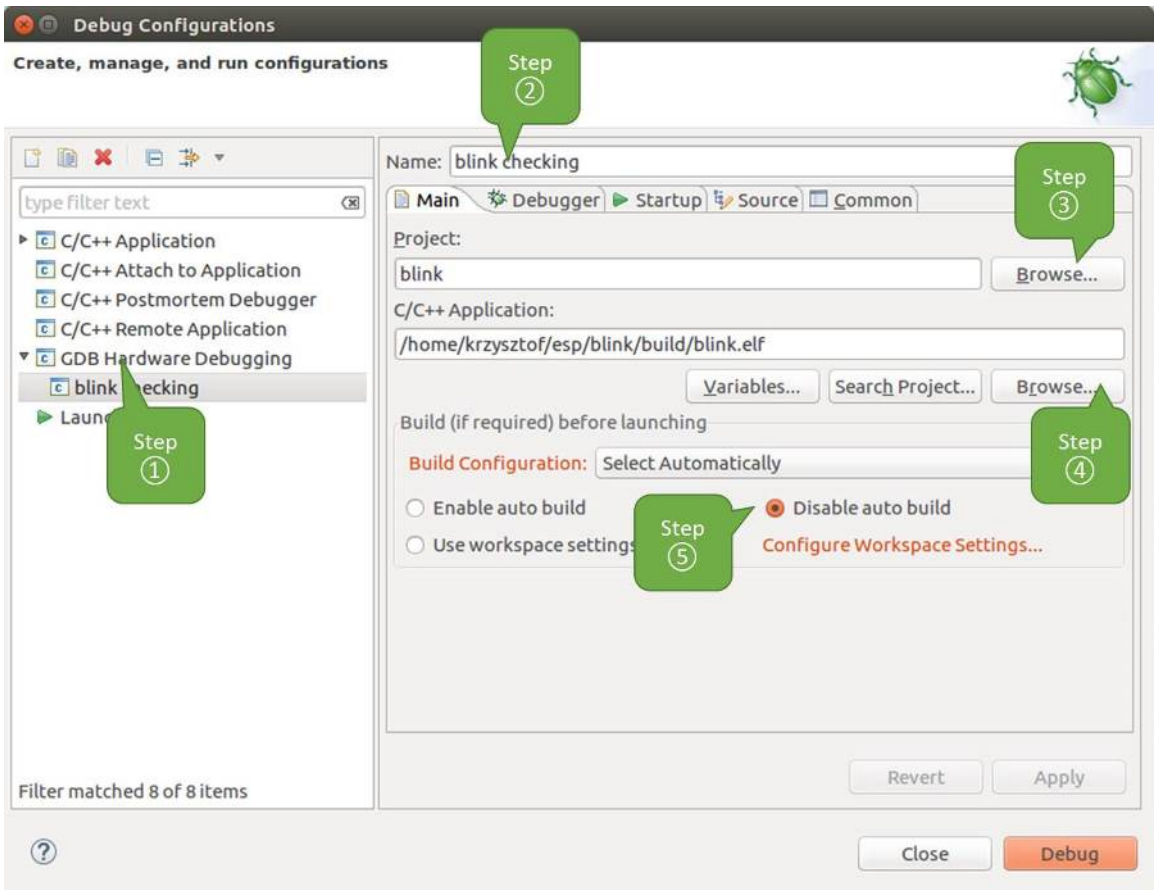


图 9: GDB 硬件调试的配置 - Main 选项卡

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。

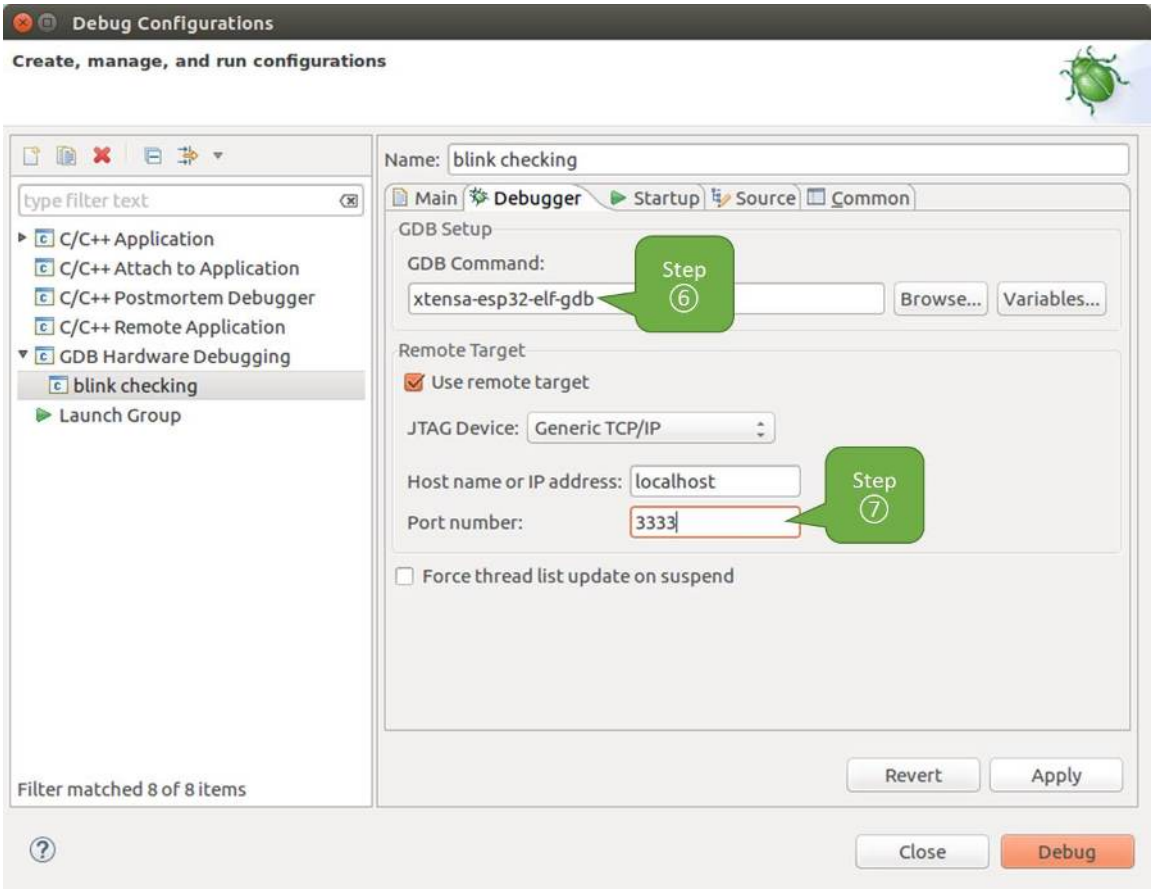


图 10: GDB 硬件调试的配置 - Debugger 选项卡

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：

```
mon reset halt
flushregs
set remote hardware-watchpoint-limit 2
```

备注： 如果您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。
    上面的启动序列看起来有些复杂，如果您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. 如果您前面已经完成配置 *ESP32-C2* 目标板 中介绍的步骤，那么目标正在运行并准备与调试器进行对话。按下“Debug”按钮就可以直接调试。否则请按下“Apply”按钮保存配置，返回配置 *ESP32-C2*

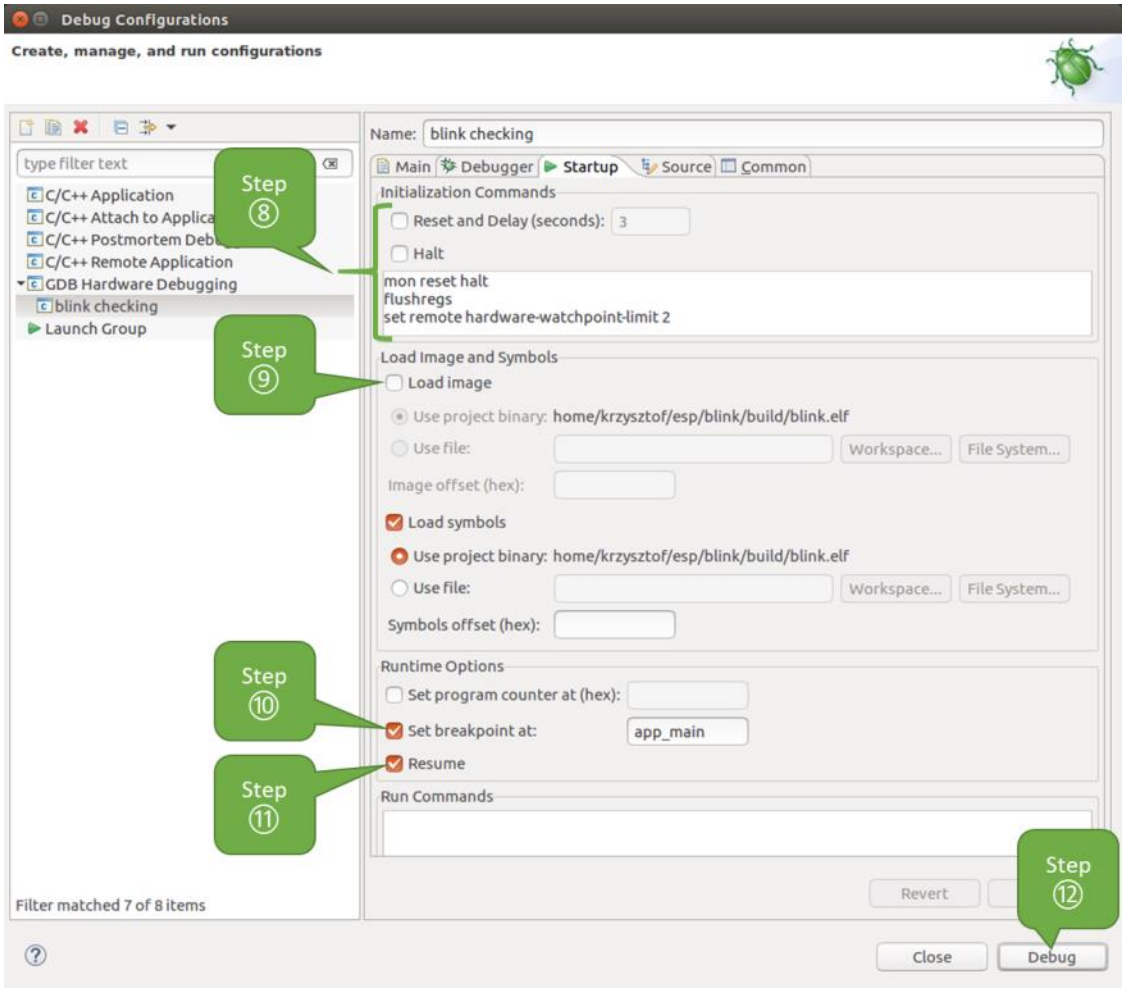


图 11: GDB 硬件调试的配置 - Startup 选项卡

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。

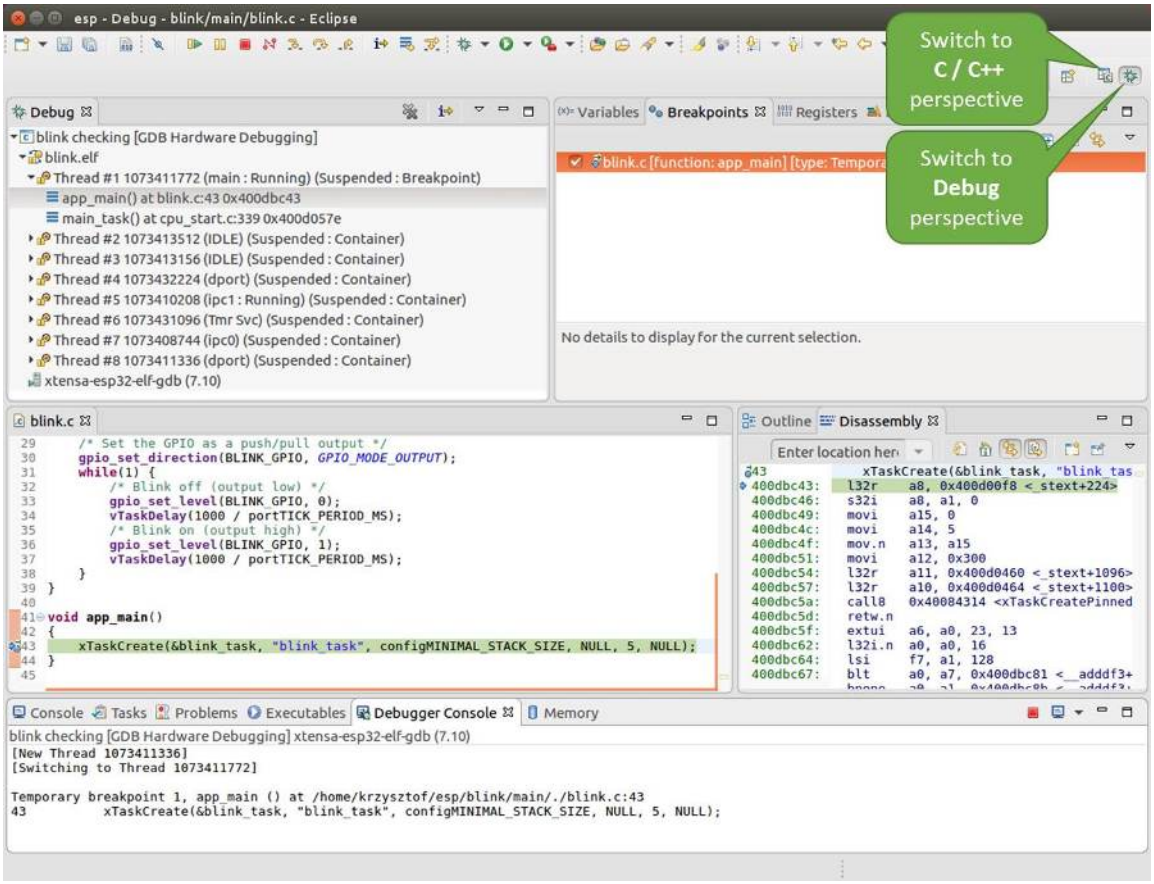


图 12: Eclipse 中的调试视图

如果您不太了解 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-C2* 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，您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
   flushregs
   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，您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/esp32c2/./freertos_hooks.c:52
52          asm("waiti 0");
JTAG tap: esp32c2.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica),␣
↪part: 0x2003, ver: 0x1)
JTAG tap: esp32c2.slave tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica),␣
↪part: 0x2003, ver: 0x1)
esp32c2: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
esp32c2: Core was reset (pwrstat=0x5F, after clear 0x0F).
esp32c2 halted. PRO_CPU: PC=0x5000004B (active)    APP_CPU: PC=0x00000000
esp32c2: target state: halted
esp32c2: Core was reset (pwrstat=0x1F, after clear 0x0F).
Target halted. PRO_CPU: PC=0x40000400 (active)    APP_CPU: PC=0x40000400
esp32c2: 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 也不会闪烁。如果这也是您看到的现象，您可以开始调试了。

如果您不太了解 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 您还可以使用 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，您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/esp32c2-ftdi.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。

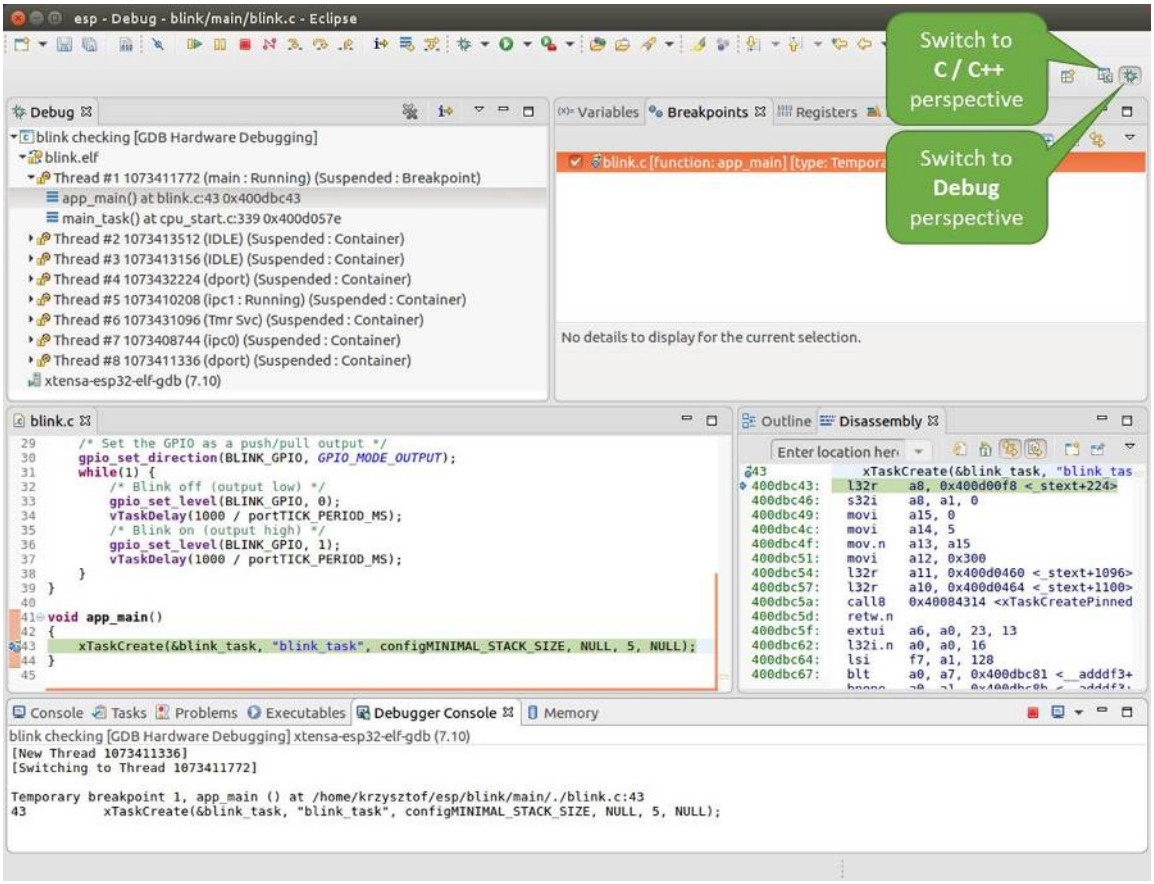


图 13: Eclipse 中的 Debug 视图

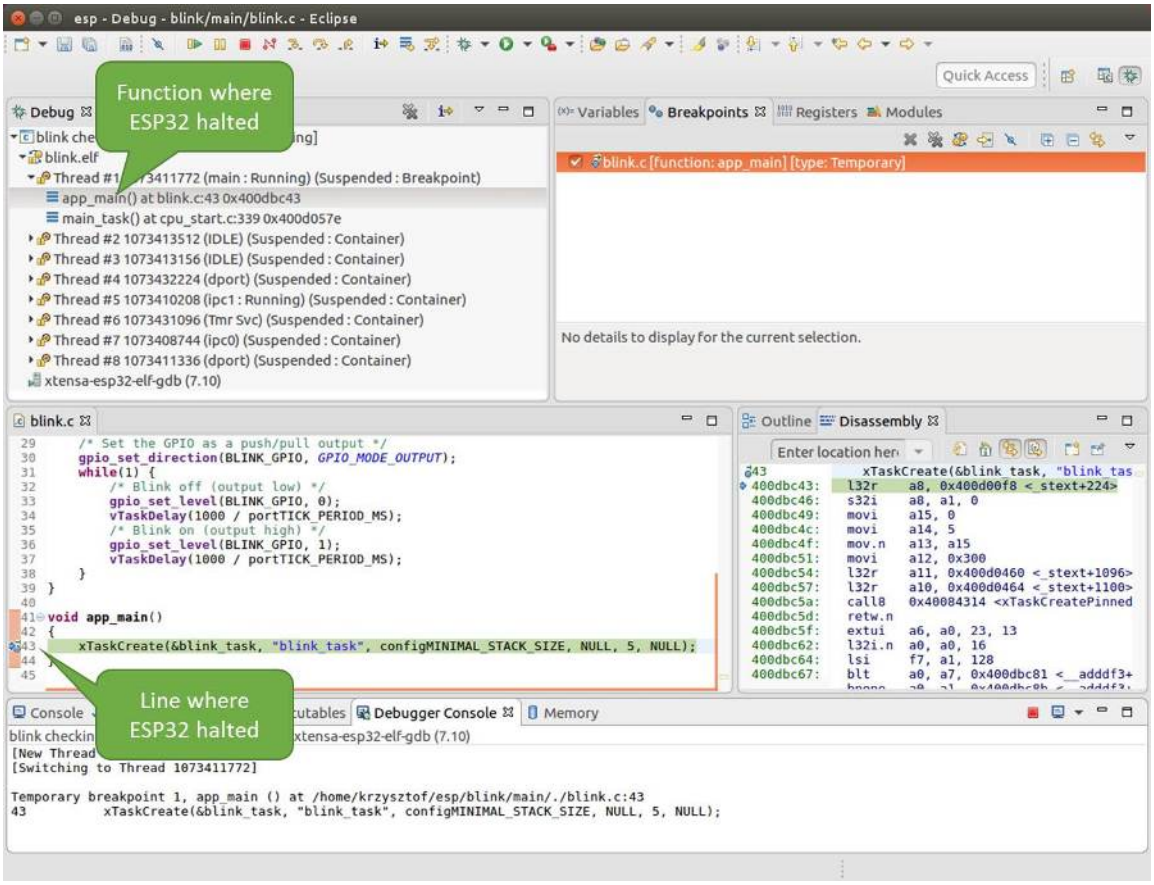

esp - Debug - blink/main/blink.c - Eclipse
Function where ESP32 halted
Line where ESP32 halted
Debug
Variables
Breakpoints
Registers
Modules
blink.elf
app_main() at blink.c:43 0x400dbc43
main_task() at cpu_start.c:339 0x400d057e
Thread #2 1073413512 (IDLE) (Suspended : Container)
Thread #3 1073413156 (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 14: 调试时目标停止

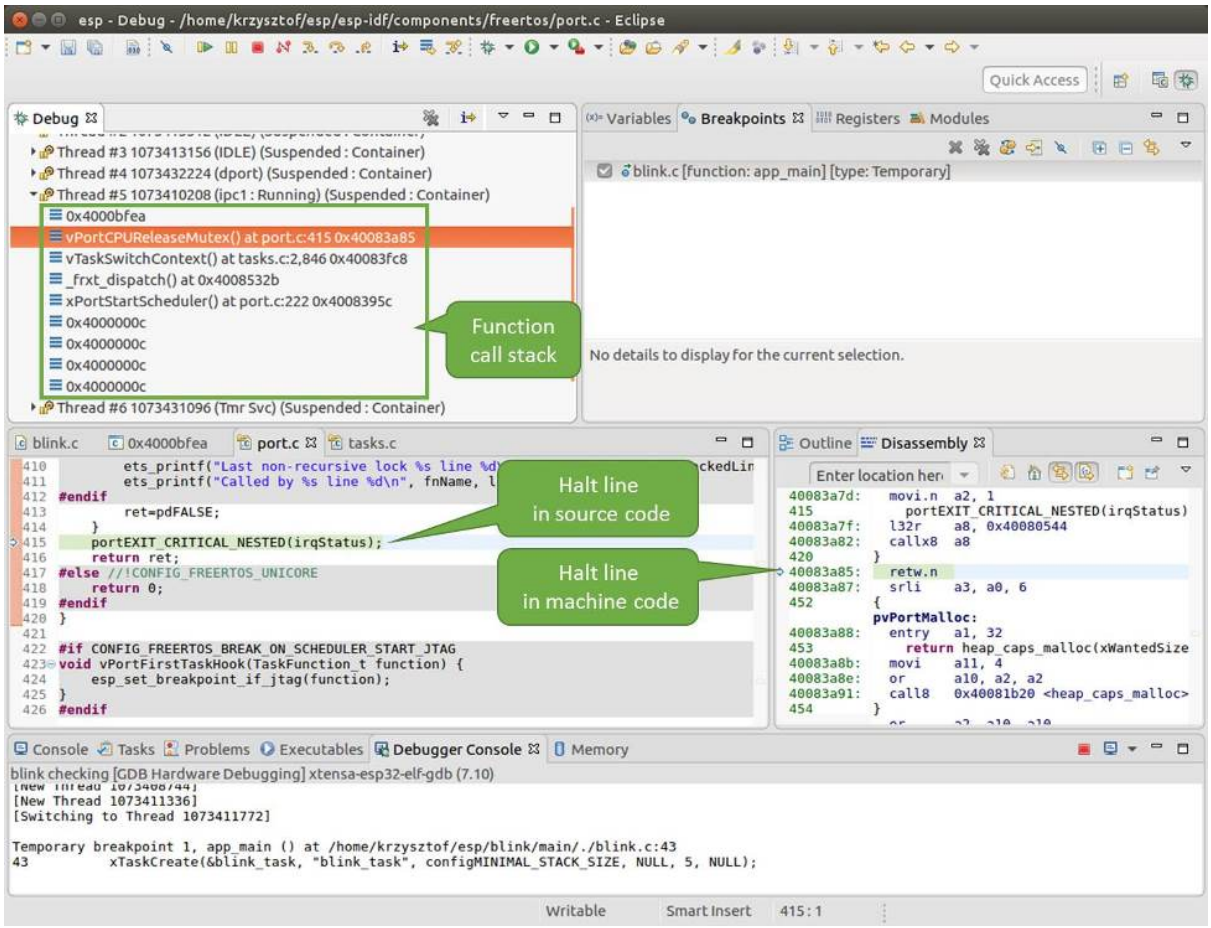


图 15: 浏览函数调用堆栈

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。

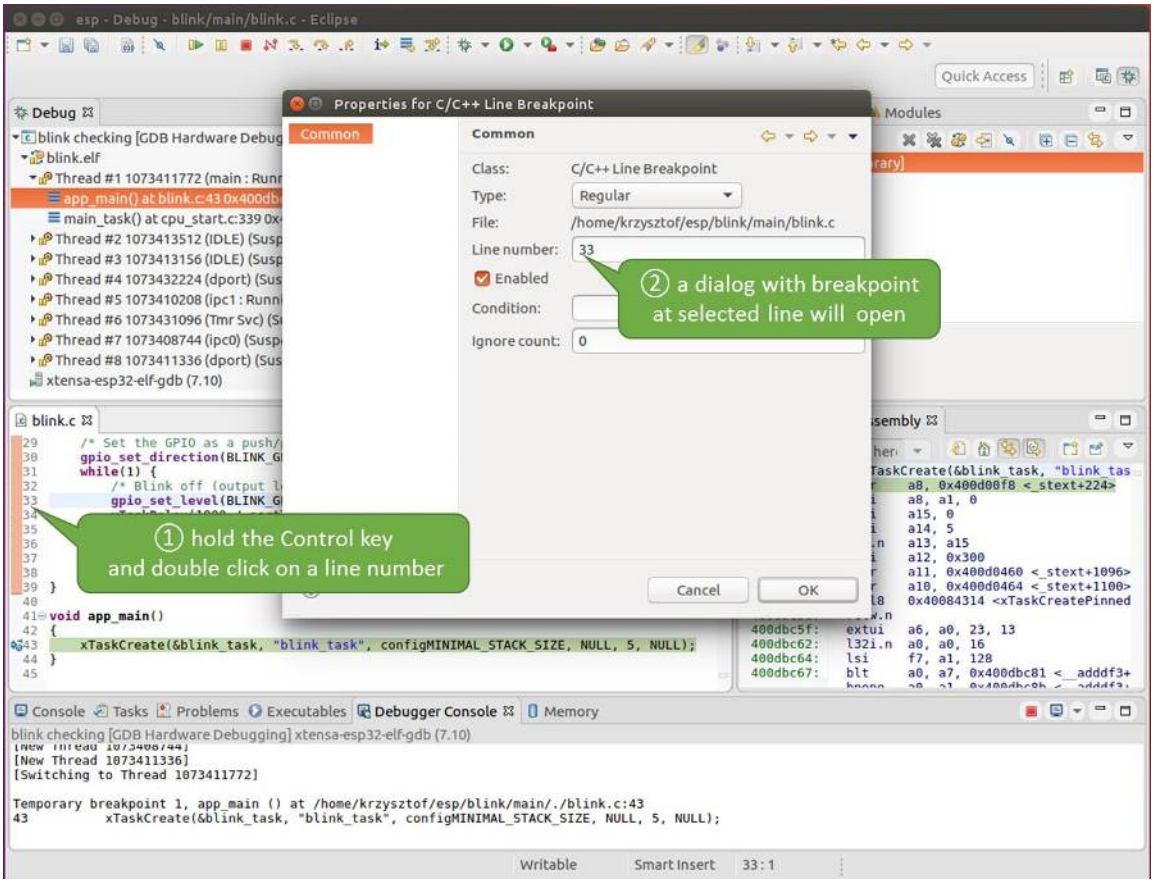


图 16: 设置断点

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。

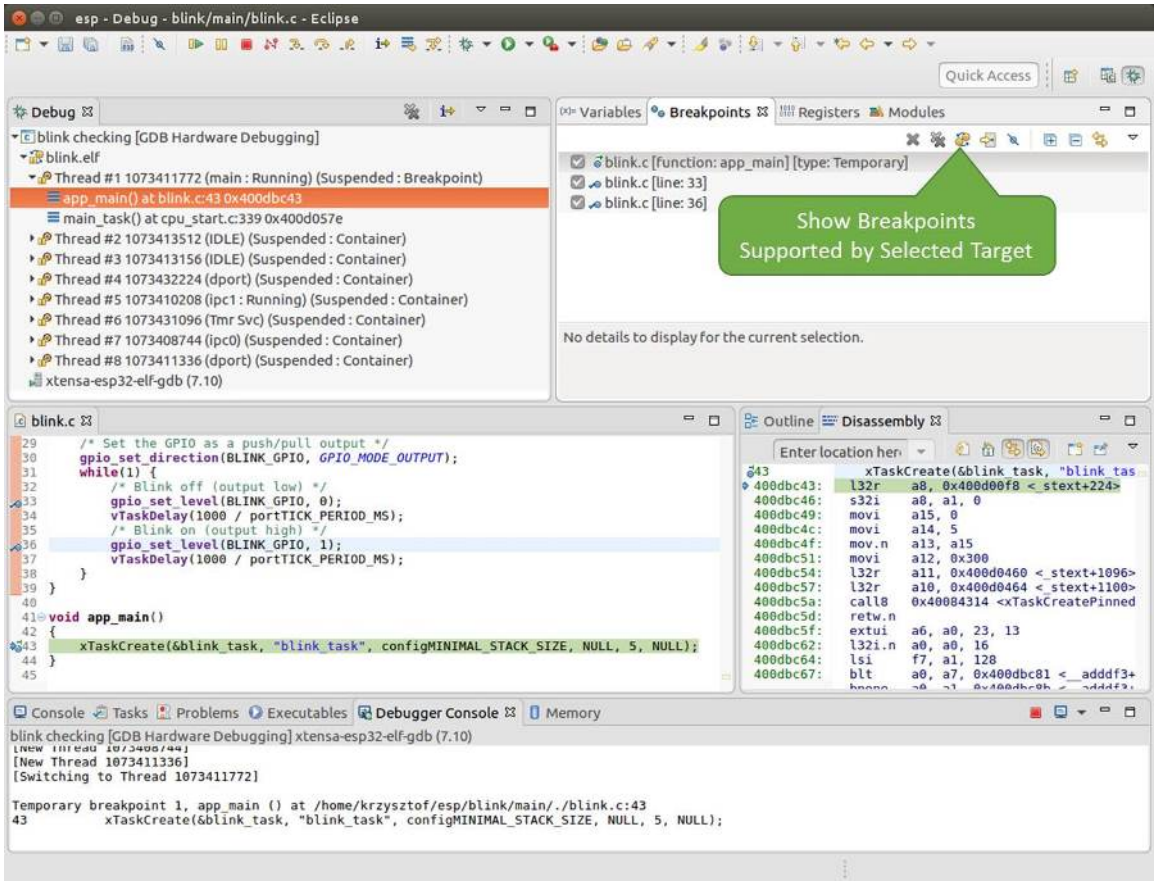


图 17: 设置了三个断点 / 最多允许两个断点

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。

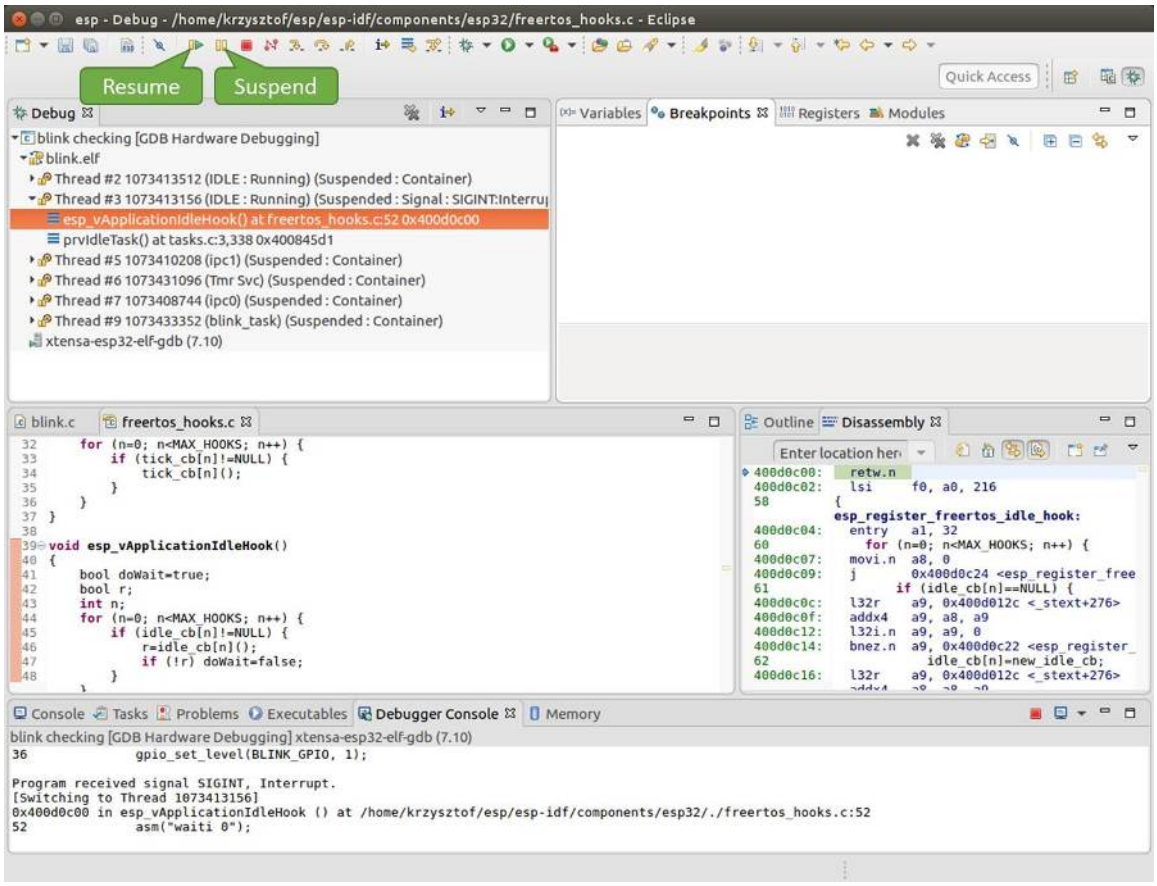


图 18: 手动暂停目标

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-C2 技术参考手册* > *IO MUX 和 GPIO Matrix (GPIO, IO_MUX)* [PDF] 章节。

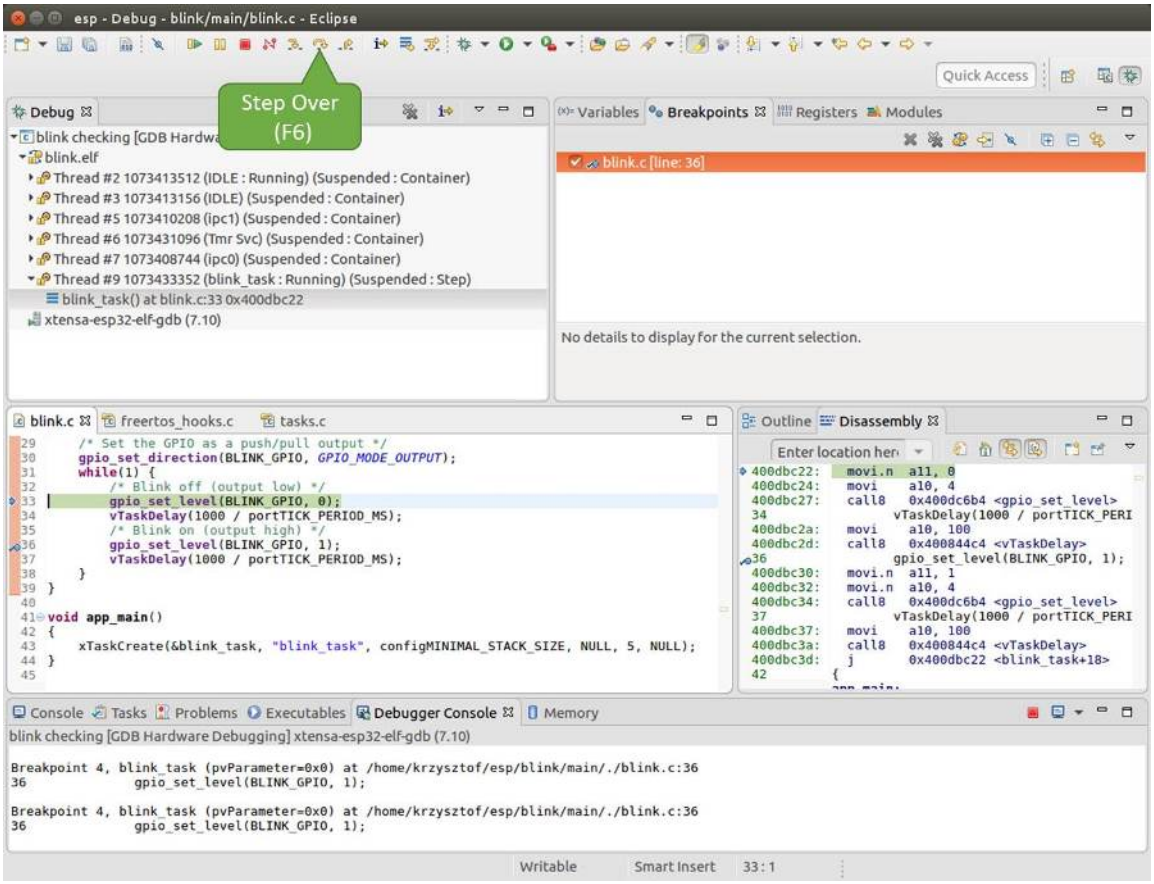


图 19: 使用“Step Over (F6)”单步执行代码

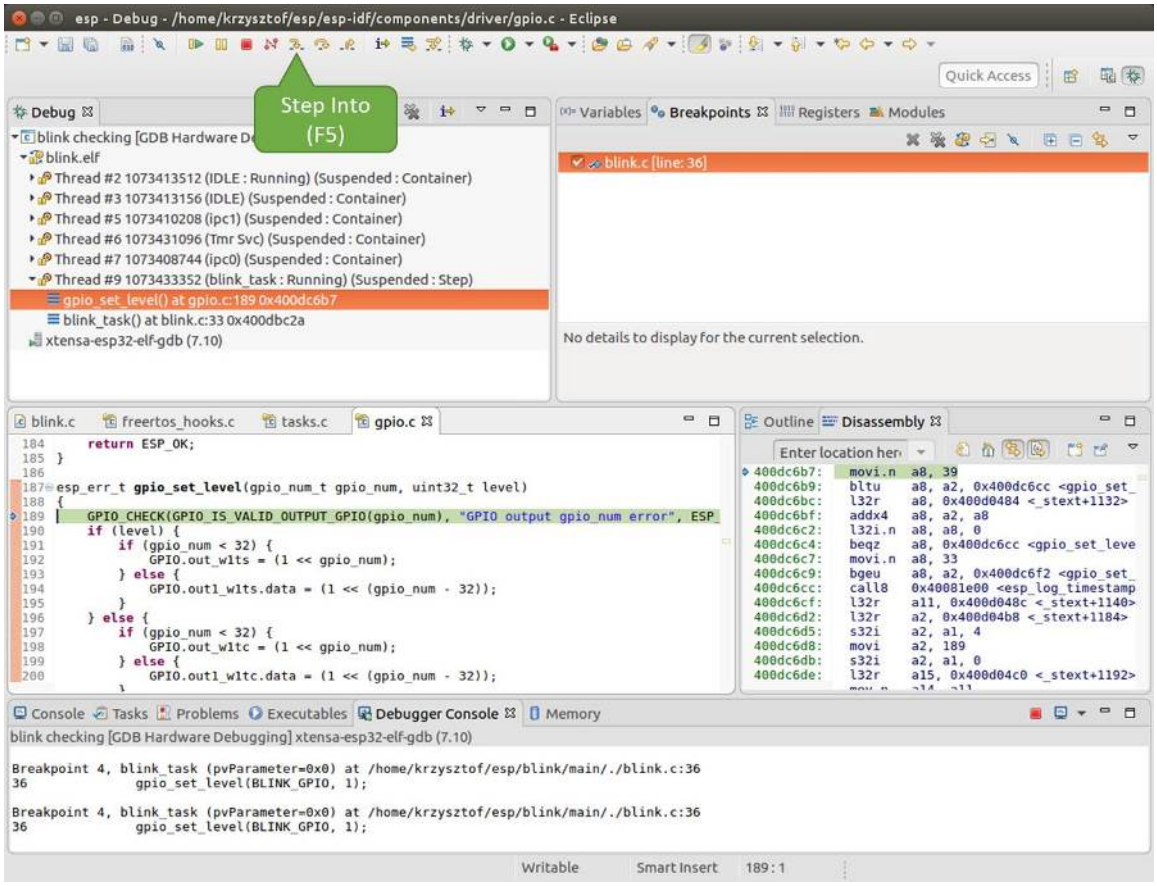


图 20: 使用“Step Into (F5)”单步执行代码

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。

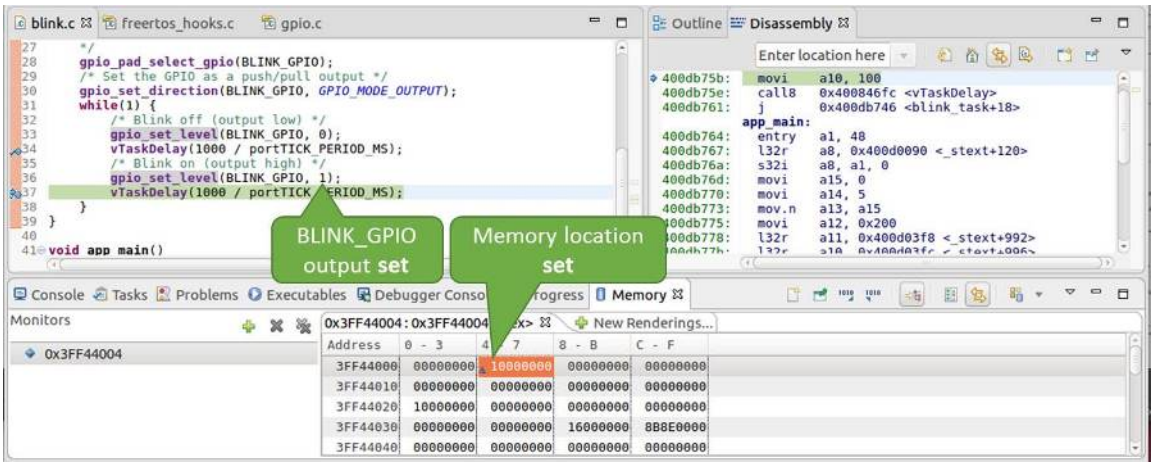


图 21: 观察内存地址 0x3FF44004 处的某个比特被置高

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）。

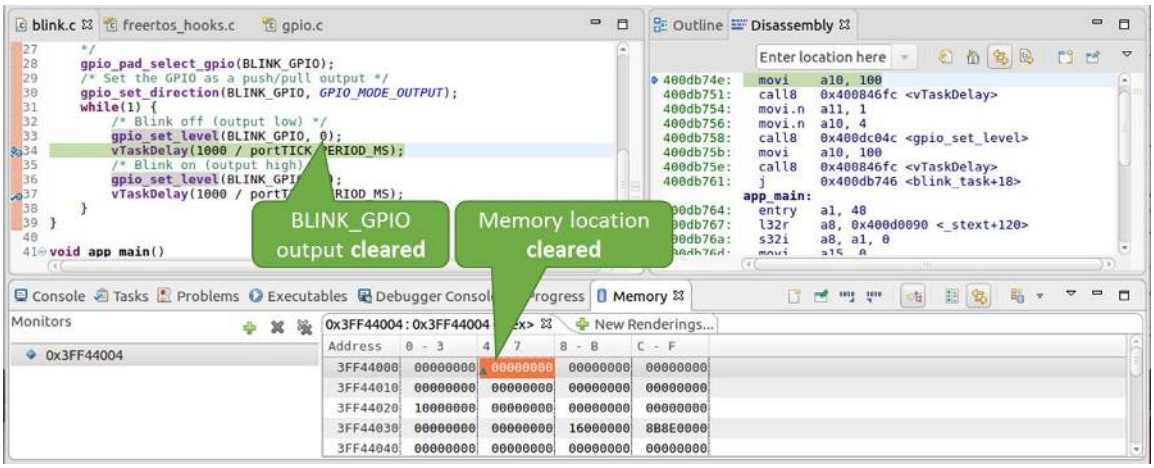


图 22: 观察内存地址 0x3FF44004 处的某个比特被置低

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-C2 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。

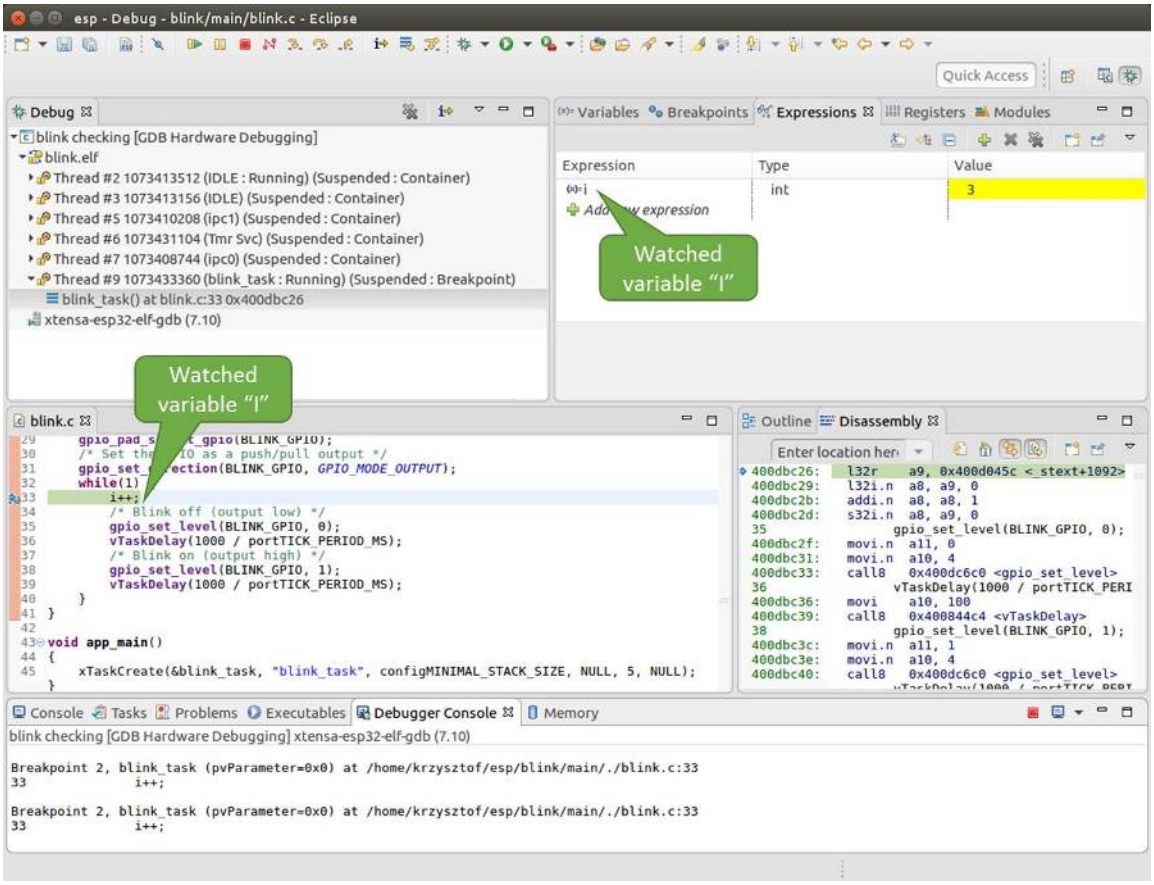

esp - Debug - blink/main/blink.c - Eclipse
Quick Access
Debug
blink checking [GDB Hardware Debugging]
blink.elf
Thread #2 1073413512 (IDLE : Running) (Suspended : Container)
Thread #3 1073413156 (IDLE) (Suspended : Container)
Thread #5 1073410208 (ipc1) (Suspended : Container)
Thread #6 1073431104 (Tmr 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 23: 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。

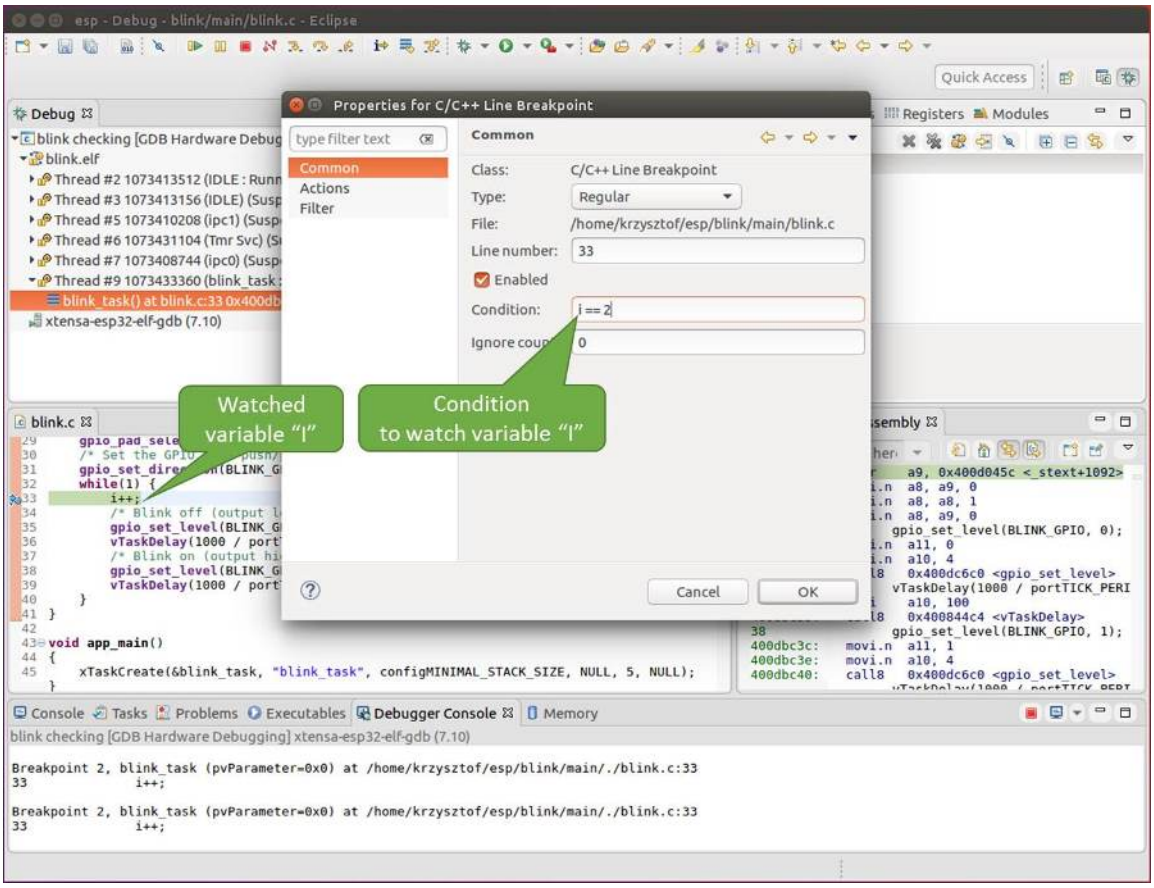


图 24: 设置条件断点

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 请检查您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/
↪esp32c2/./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/
↪esp32c2/./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/esp32c2/./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/esp32c2/./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）。

您可以通过输入 `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/esp32c2/./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，现在您可以通过输入 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/driver/gpio/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-C2 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-C2 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您调试 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，目的在于让您快速上手 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

4.13 链接器脚本生成机制

4.13.1 概述

ESP32-C2 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;

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。

4.13.2 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。

4.13.3 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 `-fdata-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：

```
mapping[target]
```

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 */
    mapping[iram0_text]
```

(下页继续)

(续上页)

```
    _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/esp32c2/sections.ld.in，生成的脚本存放在构建目录下。

将链接器脚本片段文件语法迁移至 ESP-IDF v5.0 适应版本

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

4.14 lwIP

ESP-IDF uses the open source lwIP lightweight TCP/IP stack. The ESP-IDF version of lwIP (esp-lwip) has some modifications and additions compared to the upstream project.

4.14.1 Supported APIs

ESP-IDF supports the following lwIP TCP/IP stack functions:

- *BSD Sockets API*
- *Netconn API* is enabled but not officially supported for ESP-IDF applications

Adapted APIs

> **警告：** When using any lwIP API (other than *BSD Sockets API*), please make sure that it is thread safe. To check if a given API call is safe, enable *CONFIG_LWIP_CHECK_THREAD_SAFETY* and run the application. This way lwIP asserts the TCP/IP core functionality to be correctly accessed; the execution aborts if it is not locked properly or accessed from the correct task (*lwIP FreeRTOS Task*). The general recommendation is to use *ESP-NETIF* component to interact with lwIP.

Some common lwIP "app" APIs are supported indirectly by ESP-IDF:

- DHCP Server & Client are supported indirectly via the *ESP-NETIF* functionality
- Simple Network Time Protocol (SNTP) is also supported via the *ESP-NETIF*, or directly via the lwip/include/apps/esp_sntp.h functions that provide thread-safe API to lwip/lwip/src/include/lwip/apps/sntp.h functions (see also *SNTP 时间同步*)
- ICMP Ping is supported using a variation on the lwIP ping API. See *ICMP Echo*.
- NetBIOS lookup is available using the standard lwIP API. protocols/http_server/restful_server has an option to demonstrate using NetBIOS to look up a host on the LAN.
- mDNS uses a different implementation to the lwIP default mDNS (see *mDNS 服务*), but lwIP can look up mDNS hosts using standard APIs such as `gethostbyname()` and the convention `hostname.local`, provided the *CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES* setting is enabled.
- The PPP implementation in lwIP can be used to create PPPoS (PPP over serial) interface in ESP-IDF. Please refer to the documentation of *ESP-NETIF* component to create and configure a PPP network interface, by means of the `ESP_NETIF_DEFAULT_PPP()` macro defined in esp_netif/include/esp_netif_defaults.h. Additional runtime settings are provided via the esp_netif/include/esp_netif_ppp.h. PPPoS interfaces are typically used to interact with NBIoT/GSM/LTE modems; more application level friendly API is supported by esp_modem library, which uses this PPP lwIP module behind the scenes.

4.14.2 BSD Sockets API

The BSD Sockets API is a common cross-platform TCP/IP sockets API that originated in the Berkeley Standard Distribution of UNIX but is now standardized in a section of the POSIX specification. BSD Sockets are sometimes called POSIX Sockets or Berkeley Sockets.

As implemented in ESP-IDF, lwIP supports all of the common usages of the BSD Sockets API.

References

A wide range of BSD Sockets reference material is available, including:

- Single UNIX Specification BSD Sockets page
- Berkeley Sockets Wikipedia page

Examples

A number of ESP-IDF examples show how to use the BSD Sockets APIs:

- protocols/sockets/tcp_server
- protocols/sockets/tcp_client
- protocols/sockets/udp_server
- protocols/sockets/udp_client
- protocols/sockets/udp_multicast
- protocols/http_request (Note: this is a simplified example of using a TCP socket to send an HTTP request. The *ESP HTTP 客户端* is a much better option for sending HTTP requests.)

Supported functions

The following BSD socket API functions are supported. For full details see lwip/lwip/src/include/lwip/sockets.h.

- `socket()`
- `bind()`
- `accept()`
- `shutdown()`
- `getpeername()`
- `getsockopt()` & `setsockopt()` (see *Socket Options*)
- `close()` (via *虚拟文件系统组件*)
- `read()`, `readv()`, `write()`, `writev()` (via *虚拟文件系统组件*)
- `recv()`, `recvmsg()`, `recvfrom()`
- `send()`, `sendmsg()`, `sendto()`
- `select()` (via *虚拟文件系统组件*)
- `poll()` (Note: on ESP-IDF, `poll()` is implemented by calling select internally, so using `select()` directly is recommended if a choice of methods is available.)
- `fcntl()` (see *fcntl*)

Non-standard functions:

- `ioctl()` (see *ioctls*)

备注: Some lwIP application sample code uses prefixed versions of BSD APIs, for example `lwip_socket()` instead of the standard `socket()`. Both forms can be used with ESP-IDF, but using standard names is recommended.

Socket Error Handling

BSD Socket error handling code is very important for robust socket applications. Normally the socket error handling involves the following aspects:

- Detecting the error.
- Geting the error reason code.
- Handle the error according to the reason code.

In lwIP, we have two different scenarios of handling socket errors:

- Socket API returns an error. For more information, see *Socket API Errors*.

- `select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout)` has exception descriptor indicating that the socket has an error. For more information, see *select() Errors*.

Socket API Errors

The error detection

- We can know that the socket API fails according to its return value.

Get the error reason code

- When socket API fails, the return value doesn't contain the failure reason and the application can get the error reason code by accessing errno. Different values indicate different meanings. For more information, see <*Socket Error Reason Code*>.

Example:

```
int err;
int sockfd;

if (sockfd = socket(AF_INET,SOCK_STREAM,0) < 0) {
    // the error code is obtained from errno
    err = errno;
    return err;
}
```

select() Errors

The error detection

- Socket error when `select()` has exception descriptor

Get the error reason code

- If the `select` indicates that the socket fails, we can't get the error reason code by accessing errno, instead we should call `getsockopt()` to get the failure reason code. Because `select()` has exception descriptor, the error code will not be given to errno.

备注: `getsockopt` function prototype `int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen)`. Its function is to get the current value of the option of any type, any state socket, and store the result in optval. For example, when you get the error code on a socket, you can get it by `getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &err, &optlen)`.

Example:

```
int err;

if (select(sockfd + 1, NULL, NULL, &exfds, &tval) <= 0) {
    err = errno;
    return err;
} else {
    if (FD_ISSET(sockfd, &exfds)) {
        // select() exception set using getsockopt()
        int optlen = sizeof(int);
        getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &err, &optlen);
        return err;
    }
}
```

Socket Error Reason Code Below is a list of common error codes. For more detailed list of standard POSIX/C error codes, please see newlib errno.h and the platform-specific extensions newlib/platform_include/errno.h

Error code	Description
ECONNREFUSED	Connection refused
EADDRINUSE	Address already in use
ECONNABORTED	Software caused connection abort
ENETUNREACH	Network is unreachable
ENETDOWN	Network interface is not configured
ETIMEDOUT	Connection timed out
EHOSTDOWN	Host is down
EHOSTUNREACH	Host is unreachable
EINPROGRESS	Connection already in progress
EALREADY	Socket already connected
EDESTADDRREQ	Destination address required
EPROTONOSUPPORT	Unknown protocol

Socket Options

The `getsockopt()` and `setsockopt()` functions allow getting/setting per-socket options.

Not all standard socket options are supported by lwIP in ESP-IDF. The following socket options are supported:

Common options Used with level argument `SOL_SOCKET`.

- `SO_REUSEADDR` (available if *CONFIG_LWIP_SO_REUSE* is set, behavior can be customized by setting *CONFIG_LWIP_SO_REUSE_RXTOALL*)
- `SO_KEEPALIVE`
- `SO_BROADCAST`
- `SO_ACCEPTCONN`
- `SO_RCVBUF` (available if *CONFIG_LWIP_SO_RCVBUF* is set)
- `SO_SNDTIMEO` / `SO_RCVTIMEO`
- `SO_ERROR` (this option is only used with `select()`, see *Socket Error Handling*)
- `SO_TYPE`
- `SO_NO_CHECK` (for UDP sockets only)

IP options Used with level argument `IPPROTO_IP`.

- `IP_TOS`
- `IP_TTL`
- `IP_PKTINFO` (available if *CONFIG_LWIP_NETBUF_RECVINFO* is set)

For multicast UDP sockets:

- `IP_MULTICAST_IF`
- `IP_MULTICAST_LOOP`
- `IP_MULTICAST_TTL`
- `IP_ADD_MEMBERSHIP`
- `IP_DROP_MEMBERSHIP`

TCP options TCP sockets only. Used with level argument `IPPROTO_TCP`.

- `TCP_NODELAY`

Options relating to TCP keepalive probes:

- `TCP_KEEPALIVE` (int value, TCP keepalive period in milliseconds)
- `TCP_KEEPIDLE` (same as `TCP_KEEPALIVE`, but the value is in seconds)
- `TCP_KEEPINTVL` (int value, interval between keepalive probes in seconds)
- `TCP_KEEPCNT` (int value, number of keepalive probes before timing out)

IPv6 options IPv6 sockets only. Used with level argument `IPPROTO_IPV6`

- `IPV6_CHECKSUM`
- `IPV6_V6ONLY`

For multicast IPv6 UDP sockets:

- `IPV6_JOIN_GROUP` / `IPV6_ADD_MEMBERSHIP`
- `IPV6_LEAVE_GROUP` / `IPV6_DROP_MEMBERSHIP`
- `IPV6_MULTICAST_IF`
- `IPV6_MULTICAST_HOPS`
- `IPV6_MULTICAST_LOOP`

fcntl

The `fcntl()` function is a standard API for manipulating options related to a file descriptor. In ESP-IDF, the 虚拟文件系统组件 layer is used to implement this function.

When the file descriptor is a socket, only the following `fcntl()` values are supported:

- `O_NONBLOCK` to set/clear non-blocking I/O mode. Also supports `O_NDELAY`, which is identical to `O_NONBLOCK`.
- `O_RDONLY`, `O_WRONLY`, `O_RDWR` flags for different read/write modes. These can read via `F_GETFL` only, they cannot be set using `F_SETFL`. A TCP socket will return a different mode depending on whether the connection has been closed at either end or is still open at both ends. UDP sockets always return `O_RDWR`.

ioctls

The `ioctl()` function provides a semi-standard way to access some internal features of the TCP/IP stack. In ESP-IDF, the 虚拟文件系统组件 layer is used to implement this function.

When the file descriptor is a socket, only the following `ioctl()` values are supported:

- `FIONREAD` returns the number of bytes of pending data already received in the socket's network buffer.
- `FIONBIO` is an alternative way to set/clear non-blocking I/O status for a socket, equivalent to `fcntl(fd, F_SETFL, O_NONBLOCK, ...)`.

4.14.3 Netconn API

lwIP supports two lower level APIs as well as the BSD Sockets API: the Netconn API and the Raw API.

The lwIP Raw API is designed for single threaded devices and is not supported in ESP-IDF.

The Netconn API is used to implement the BSD Sockets API inside lwIP, and it can also be called directly from ESP-IDF apps. This API has lower resource usage than the BSD Sockets API, in particular it can send and receive data without needing to first copy it into internal lwIP buffers.

重要： Espressif does not test the Netconn API in ESP-IDF. As such, this functionality is *enabled but not supported*. Some functionality may only work correctly when used from the BSD Sockets API.

For more information about the Netconn API, consult lwip/lwip/src/include/lwip/api.h and this wiki page which is part of the unofficial lwIP Application Developers Manual.

4.14.4 lwIP FreeRTOS Task

lwIP creates a dedicated TCP/IP FreeRTOS task to handle socket API requests from other tasks.

A number of configuration items are available to modify the task and the queues ("mailboxes") used to send data to/from the TCP/IP task:

- *CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*
- *CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*
- *CONFIG_LWIP_TCPIP_TASK_AFFINITY*

4.14.5 IPv6 Support

Both IPv4 and IPv6 are supported as a dual stack and are enabled by default. Both IPv6 and IPv4 may be disabled separately if they are not needed (see *Minimum RAM usage*). IPv6 support is limited to *Stateless Autoconfiguration* only, *Stateful configuration* is not supported in ESP-IDF (not in upstream lwip). IPv6 Address configuration is defined by means of these protocols or services:

- **SLAAC** IPv6 Stateless Address Autoconfiguration (RFC-2462)
- **DHCPv6** Dynamic Host Configuration Protocol for IPv6 (RFC-8415)

None of these two types of address configuration is enabled by default, so the device uses only Link Local addresses or statically defined addresses.

Stateless Autoconfiguration Process

To enable address autoconfiguration using Router Advertisement protocol please enable:

- *CONFIG_LWIP_IPV6_AUTOCONFIG*

This configuration option enables IPv6 autoconfiguration for all network interfaces (in contrast to the upstream lwIP, where the autoconfiguration needs to be explicitly enabled for each netif with `netif->ip6_autoconfig_enabled=1`

DHCPv6

DHCPv6 in lwIP is very simple and support only stateless configuration. It could be enabled using:

- *CONFIG_LWIP_IPV6_DHCP6*

Since the DHCPv6 works only in its stateless configuration, the *Stateless Autoconfiguration Process* has to be enabled, too, by means of *CONFIG_LWIP_IPV6_AUTOCONFIG*. Moreover, the DHCPv6 needs to be explicitly enabled form the application code using

```
dhcp6_enable_stateless(netif);
```

DNS servers in IPv6 autoconfiguration

In order to autoconfigure DNS server(s), especially in IPv6 only networks, we have these two options

- Recursive domain name system –this belongs to the Neighbor Discovery Protocol (NDP), uses *Stateless Autoconfiguration Process*. Number of servers must be set *CONFIG_LWIP_IPV6_RDNSS_MAX_DNS_SERVERS*, this is option is disabled (set to 0) by default.
- DHCPv6 stateless configuration –uses *DHCPv6* to configure DNS servers. Note that the this configuration assumes IPv6 Router Advertisement Flags (RFC-5175) to be set to
  - Managed Address Configuration Flag = 0
  - Other Configuration Flag = 1

4.14.6 esp-lwip custom modifications

Additions

The following code is added which is not present in the upstream lwIP release:

Thread-safe sockets It is possible to `close()` a socket from a different thread to the one that created it. The `close()` call will block until any function calls currently using that socket from other tasks have returned.

It is, however, not possible to delete a task while it is actively waiting on `select()` or `poll()` APIs. It is always necessary that these APIs exit before destroying the task, as this might corrupt internal structures and cause subsequent crashes of the lwIP. (These APIs allocate globally referenced callback pointers on stack, so that when the task gets destroyed before unrolling the stack, the lwIP would still hold pointers to the deleted stack)

On demand timers lwIP IGMP and MLD6 features both initialize a timer in order to trigger timeout events at certain times.

The default lwIP implementation is to have these timers enabled all the time, even if no timeout events are active. This increases CPU usage and power consumption when using automatic light sleep mode. `esp-lwip` default behaviour is to set each timer "on demand" so it is only enabled when an event is pending.

To return to the default lwIP behaviour (always-on timers), disable *CONFIG_LWIP_TIMERS_ONDEMAND*.

Lwip timers API When users are not using WiFi, these APIs provide users with the ability to turn off LwIP timer to reduce power consumption.

The following API functions are supported. For full details see lwip/lwip/src/include/lwip/timeouts.h.

- `sys_timeouts_init()`
- `sys_timeouts_deinit()`

Additional Socket Options

- Some standard IPV4 and IPV6 multicast socket options are implemented (see *Socket Options*).
- Possible to set IPV6-only UDP and TCP sockets with `IPV6_V6ONLY` socket option (normal lwIP is TCP only).

IP layer features

- IPV4 source based routing implementation is different.
- IPV4 mapped IPV6 addresses are supported.

Customized lwIP hooks The original lwIP supports implementing custom compile-time modifications via `LWIP_HOOK_FILENAME`. This file is already used by the IDF port layer, but IDF users could still include and implement any custom additions via a header file defined by the macro `ESP_IDF_LWIP_HOOK_FILENAME`. Here is an exmaple of adding a custom hook file to the build process (the hook is called `my_hook.h` and located in the project's `main` folder):

```
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\"")
```

Limitations

Calling `send()` or `sendto()` repeatedly on a UDP socket may eventually fail with `errno` equal to `ENOMEM`. This is a limitation of buffer sizes in the lower layer network interface drivers. If all driver transmit buffers are full then UDP transmission will fail. Applications sending a high volume of UDP datagrams who don't wish for any to be dropped by the sender should check for this error code and re-send the datagram after a short delay.

Increasing the number of TX buffers in the *Wi-Fi* project configuration may also help.

4.14.7 Performance Optimization

TCP/IP performance is a complex subject, and performance can be optimized towards multiple goals. The default settings of ESP-IDF are tuned for a compromise between throughput, latency, and moderate memory usage.

Maximum throughput

Espressif tests ESP-IDF TCP/IP throughput using the wifi/iperf example in an RF sealed enclosure.

The wifi/iperf/sdkconfig.defaults file for the iperf example contains settings known to maximize TCP/IP throughput, usually at the expense of higher RAM usage. To get maximum TCP/IP throughput in an application at the expense of other factors then suggest applying settings from this file into the project sdkconfig.

重要： Suggest applying changes a few at a time and checking the performance each time with a particular application workload.

- If a lot of tasks are competing for CPU time on the system, consider that the lwIP task has configurable CPU affinity (*CONFIG_LWIP_TCPIP_TASK_AFFINITY*) and runs at fixed priority `ESP_TASK_TCPIP_PRIO` (18). Configure competing tasks to be pinned to a different core, or to run at a lower priority. See also *Built-In Task Priorities*.
- If using `select()` function with socket arguments only, disabling *CONFIG_VFS_SUPPORT_SELECT* will make `select()` calls faster.
- If there is enough free IRAM, select *CONFIG_LWIP_IRAM_OPTIMIZATION* to improve TX/RX throughput

If using a Wi-Fi network interface, please also refer to *Wi-Fi 缓冲区使用情况*.

Minimum latency

Except for increasing buffer sizes, most changes which increase throughput will also decrease latency by reducing the amount of CPU time spent in lwIP functions.

- For TCP sockets, lwIP supports setting the standard `TCP_NODELAY` flag to disable Nagle’s algorithm.

Minimum RAM usage

Most lwIP RAM usage is on-demand, as RAM is allocated from the heap as needed. Therefore, changing lwIP settings to reduce RAM usage may not change RAM usage at idle but can change it at peak.

- Reducing *CONFIG_LWIP_MAX_SOCKETS* reduces the maximum number of sockets in the system. This will also cause TCP sockets in the `WAIT_CLOSE` state to be closed and recycled more rapidly (if needed to open a new socket), further reducing peak RAM usage.
- Reducing *CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*, *CONFIG_LWIP_TCP_RECVMBOX_SIZE* and *CONFIG_LWIP_UDP_RECVMBOX_SIZE* reduce memory usage at the expense of throughput, depending on usage.
- Reducing *CONFIG_LWIP_TCP_MSL*, *CONFIG_LWIP_TCP_FIN_WAIT_TIMEOUT* reduces the maximum segment lifetime in the system. This will also cause TCP sockets in the `TIME_WAIT`, `FIN_WAIT_2` state to be closed and recycled more rapidly
- Disabling *CONFIG_LWIP_IPV6* can save about 39 KB for firmware size and 2KB RAM when the system is powered up and 7KB RAM when the TCPIP stack is running. If there is no requirement for supporting IPV6 then it can be disabled to save flash and RAM footprint.
- Disabling *CONFIG_LWIP_IPV4* can save about 26 KB of firmware size and 600B RAM on power up and 6 KB RAM when the TCP/IP stack is running. If the local network supports IPv6-only configuration then IPv4 can be disabled to save flash and RAM footprint.

If using Wi-Fi, please also refer to *Wi-Fi 缓冲区使用情况*.

Peak Buffer Usage The peak heap memory that lwIP consumes is the **theoretically-maximum memory** that the lwIP driver consumes. Generally, the peak heap memory that lwIP consumes depends on:

- the memory required to create a UDP connection: lwip_udp_conn
- the memory required to create a TCP connection: lwip_tcp_conn
- the number of UDP connections that the application has: lwip_udp_con_num
- the number of TCP connections that the application has: lwip_tcp_con_num
- the TCP TX window size: lwip_tcp_tx_win_size
- the TCP RX window size: lwip_tcp_rx_win_size

So, the peak heap memory that the LwIP consumes can be calculated with the following formula:

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))

Some TCP-based applications need only one TCP connection. However, they may choose to close this TCP connection and create a new one when an error (such as a sending failure) occurs. This may result in multiple TCP connections existing in the system simultaneously, because it may take a long time for a TCP connection to close, according to the TCP state machine (refer to RFC793).

4.15 存储器类型

ESP32-C2 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-C2 技术参考手册* > *系统和存储器* [PDF]。

4.15.1 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。

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。

示例:

```
__NOINIT_ATTR uint32_t noinit_data;
```

4.15.2 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-C2 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。

4.15.3 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。

4.15.4 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 DRAM 放入*DRAM（数据 RAM）* 部分（见上文）。

4.15.5 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。

4.15.6 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:

- 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
}
```

4.16 OpenThread

OpenThread is a IP stack running on the 802.15.4 MAC layer which features mesh network and low power consumption.

4.16.1 Mode of the OpenThread stack

OpenThread can run under the following modes on Espressif chips:

Standalone node

The full OpenThread stack and the application layer runs on the same chip. This mode is available on chips with 15.4 radio such as {IDF_TARGET}.

Radio Co-Processor (RCP)

The chip will be connected to another host running the OpenThread IP stack. It will send and received 15.4 packets on behalf of the host. This mode is available on chips with 15.4 radio such as {IDF_TARGET}. The underlying transport between the chip and the host can be SPI or UART. For sake of latency, we recommend to use SPI as the underlying transport.

OpenThread host

For chips without 15.4 radio, it can be connected to an RCP and run OpenThread under host mode. This mode enables OpenThread on Wi-Fi chips such as ESP32, ESP32-S2, ESP32-S3 and ESP32-C3. The following diagram shows how devices work under different modes:

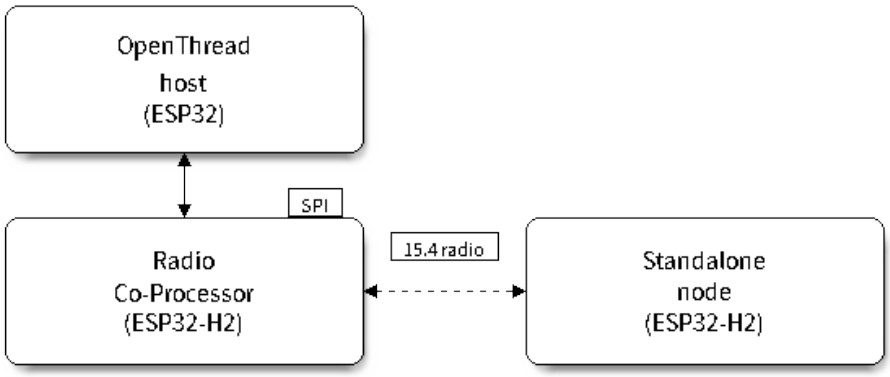


图 25: OpenThread device modes

4.16.2 How To Write an OpenThread Application

The OpenThread openthread/ot_cli example will be a good place to start at. It demonstrates basic OpenThread initialization and simple socket-based server and client.

Before OpenThread initialization

- s1.1 The main task calls `esp_vfs_eventfd_register()` to initialize the eventfd virtual filesystem. The eventfd file system is used for task notification in the OpenThread driver.
- s1.2 The main task calls `nvs_flash_init()` to initialize the NVS where the Thread network data is stored.
- s1.3 **Optional**, The main task calls `esp_netif_init()` only when it wants to create the network interface for Thread.
- s1.4: The main task calls `esp_event_loop_create()` to create the system Event task and initialize an application event' s callback function.

OpenThread stack initialization

- s2.1: Call `esp_openthread_init()` to initialize the OpenThread stack.

OpenThread network interface initialization

The whole stage is **optional** and only required if the application wants to create the network interface for Thread. - s3.1: Call `esp_netif_new()` with *ESP_NETIF_DEFAULT_OPENTHREAD* to create the interface. - s3.2: Call `esp_openthread_netif_glue_init()` to create the OpenThread interface handlers. - s3.3: Call `esp_netif_attach()` to attach the handlers to the interface.

The OpenThread main loop

- s4.3: Call `esp_openthread_launch_mainloop()` to launch the OpenThread main loop. Note that this is a busy loop and will not return until the OpenThread stack is terminated.

Calling OpenThread APIs

The OpenThread APIs are not thread-safe. When calling OpenThread APIs from other tasks, make sure to hold the lock with `esp_openthread_lock_acquire()` and release the lock with `esp_openthread_lock_release()` afterwards.

Deinitialization

The following steps are required to deintialize the OpenThread stack: - Call `esp_netif_destroy()` and `esp_openthread_netif_glue_deinit()` to deintialize the OpenThread network interface if you have created one. - Call `esp_openthread_deinit()` to deintialize the OpenThread stack.

4.16.3 The OpenThread border router

The OpenThread border router connects the Thread network with other IP networks. It will provide IPv6 connectivity, service registration and commission functionality. To launch an OpenThread border router on a ESP chip, you need to connect an RCP to a Wi-Fi capable chip such as ESP32. Call `esp_openthread_border_router_init()` during the initialization will launch all the border routing functionalities.

You may refer to the openthread/ot_br example and the README for further border router details.

4.17 分区表

4.17.1 概述

每片 ESP32-C2 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。

4.17.2 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。

4.17.3 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-C2 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）。

如果您的应用程序需要以 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。
    - 如果您希望在 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*CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION* 选项。此时，您还需要手动将 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）。
    * 用于存储 Wi-Fi 数据（如果使用了*esp_wifi_set_storage(WIFI_STORAGE_FLASH)* 初始化函数）。
    * NVS API 还可以用于其他应用程序数据。
    * 强烈建议您应为 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 还支持其它预定义的子类型用于数据存储，包括*FAT 文件系统* (*ESP_PARTITION_SUBTYPE_DATA_FAT*)，*SPIFFS* (*ESP_PARTITION_SUBTYPE_DATA_SPIFFS*) 等。

  其它数据子类型已预留给 esp-idf 未来使用。
- 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 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。

分区若偏移地址为空，则会紧跟着前一个分区之后开始；若为首个分区，则将紧跟着分区表开始。

app 分区的偏移地址必须要与 0x10000 (64 K) 对齐，如果将偏移字段留空，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 K) 对齐，则该工具会报错。

app 分区的大小和偏移地址可以采用十进制数、以 0x 为前缀的十六进制数，且支持 K 或 M 的倍数单位（分别代表 1024 和 1024*1024 字节）。

如果您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，如果您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，如果您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

当前仅支持 `encrypted` 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： `app` 分区始终会被加密，不管 Flags 字段是否设置。

4.17.4 生成二进制分区表

烧写到 ESP32-C2 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。

如果您在项目配置菜单（`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
```

4.17.5 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。

4.17.6 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，您可以使用 `idf.py erase-flash` 命令或者 `esptool.py erase_flash` 命令来擦除 flash 中的所有内容。

4.17.7 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 partool.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
```

4.18 Performance

ESP-IDF ships with default settings that are designed for a trade-off between performance, resource usage, and available functionality.

These guides describe how to optimize a firmware application for a particular aspect of performance. Usually this involves some trade-off in terms of limiting available functions, or swapping one aspect of performance (such as execution speed) for another (such as RAM usage).

4.18.1 How to Optimize Performance

1. Decide what the performance-critical aspects of your application are (for example: a particular response time to a certain network operation, a particular startup time limit, particular peripheral data throughput, etc.).
2. Find a way to measure this performance (some methods are outlined in the guides below).
3. Modify the code and project configuration and compare the new measurement to the old measurement.
4. Repeat step 3 until the performance meets the requirements set out in step 1.

4.18.2 Guides

Maximizing Execution Speed

Overview Optimizing execution speed is a key element of software performance. Code that executes faster can also have other positive effects, like reducing overall power consumption. However, improving execution speed may have trade-offs with other aspects of performance such as *Minimizing Binary Size*.

Choose What To Optimize If a function in the application firmware is executed once per week in the background, it may not matter if that function takes 10 ms or 100 ms to execute. If a function is executed constantly at 10 Hz, it matters greatly if it takes 10 ms or 100 ms to execute.

Most application firmwares will only have a small set of functions which require optimal performance. Perhaps those functions are executed very often, or have to meet some application requirements for latency or throughput. Optimization efforts should be targeted at these particular functions.

Measuring Performance The first step to improving something is to measure it.

Basic Performance Measurements If measuring performance relative to an external interaction with the world, you may be able to measure this directly (for example see the examples wifi/iperf and ethernet/iperf for measuring general network performance, or you can use an oscilloscope or logic analyzer to measure timing of an interaction with a device peripheral.)

Otherwise, one way to measure performance is to augment the code to take timing measurements:

```
#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(); // This is the thing you need to measure
    }

    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);
}
```

Executing the target multiple times can help average out factors like RTOS context switches, overhead of measurements, etc.

- Using *esp_timer_get_time()* generates "wall clock" timestamps with microsecond precision, but has moderate overhead each time the timing functions are called.
- It's also possible to use the standard Unix `gettimeofday()` and `utime()` functions, although the overhead is slightly higher.
- Otherwise, including `hal/cpu_hal.h` and calling the HAL function `cpu_hal_get_cycle_count()` will return the number of CPU cycles executed. This function has lower overhead than the others. It is good for measuring very short execution times with high precision.
- While performing "microbenchmarks" (i.e. benchmarking only a very small routine of code that runs in less than 1-2 milliseconds), the flash cache performance can sometimes cause big variations in timing measurements depending on the binary. This happens because binary layout can cause different patterns of cache misses in a particular sequence of execution. If the test code is larger then this effect usually averages out. Executing a small function multiple times when benchmarking can help reduce the impact of flash cache misses. Alternatively, move this code to IRAM (see *Targeted Optimizations*).

External Tracing The *应用层跟踪库* allows measuring code execution with minimal impact on the code itself.

Tasks If the option *CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS* is enabled then the FreeRTOS API *vTaskGetRunTimeStats()* can be used to retrieve runtime information about the processor time used by each FreeRTOS task.

SEGGER SystemView is an excellent tool for visualizing task execution and looking for performance issues or improvements in the system as a whole.

Improving Overall Speed The following optimizations will improve the execution of nearly all code - including boot times, throughput, latency, etc:

- Set *CONFIG_ESPTOOLPY_FLASHMODE* to QIO or QOUT mode (Quad I/O). Both will almost double the speed at which code is loaded or executed from flash compared to the default DIO mode. QIO is slightly faster than QOUT if both are supported. Note that both the flash chip model and the electrical connections between the ESP32-C2 and the flash chip must support quad I/O modes or the SoC will not work correctly.
- Set *CONFIG_COMPILER_OPTIMIZATION* to "Optimize for performance (-O2)". This may slightly increase binary size compared to the default setting, but will almost certainly increase performance of some code. Note that if your code contains C or C++ Undefined Behaviour then increasing the compiler optimization level may expose bugs that otherwise are not seen.
- Avoid using floating point arithmetic (`float`). On ESP32-C2 these calculations are emulated in software and are very slow. If possible then use fixed point representations, a different method of integer representation, or convert part of the calculation to be integer only before switching to floating point.
- Avoid using double precision floating point arithmetic (`double`). These calculations are emulated in software and are very slow. If possible then use an integer-based representation, or single-precision floating point.

Reduce Logging Overhead Although standard output is buffered, it's possible for an application to be limited by the rate at which it can print data to log output once buffers are full. This is particularly relevant for startup time if a lot of output is logged, but can happen at other times as well. There are multiple ways to solve this problem:

- Reduce the volume of log output by lowering the app *CONFIG_LOG_DEFAULT_LEVEL* (the equivalent bootloader setting is *CONFIG_BOOTLOADER_LOG_LEVEL*). This also reduces the binary size, and saves some CPU time spent on string formatting.
- Increase the speed of logging output by increasing the *CONFIG_ESP_CONSOLE_UART_BAUDRATE*

Not Recommended The following options will also increase execution speed, but are not recommended as they also reduce the debuggability of the firmware application and may increase the severity of any bugs.

- Set *CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL* to disabled. This also reduces firmware binary size by a small amount. However, it may increase the severity of bugs in the firmware including security-related bugs. If necessary to do this to optimize a particular function, consider adding `#define NDEBUG` in the top of that single source file instead.

Targeted Optimizations The following changes will increase the speed of a chosen part of the firmware application:

- Move frequently executed code to IRAM. By default, all code in the app is executed from flash cache. This means that it's possible for the CPU to have to wait on a "cache miss" while the next instructions are loaded from flash. Functions which are copied into IRAM are loaded once at boot time, and then will always execute at full speed.
  IRAM is a limited resource, and using more IRAM may reduce available DRAM, so a strategic approach is needed when moving code to IRAM. See *IRAM（指令 RAM）* for more information.
- Jump table optimizations can be re-enabled for individual source files that don't need to be placed in IRAM. For hot paths in large switch cases this will improve performance. For instructions on how to add the -fjump-tables -ftree-switch-conversion options when compiling individual source files, see 组件编译控制

Improving Startup Time In addition to the overall performance improvements shown above, the following options can be tweaked to specifically reduce startup time:

- Minimizing the *CONFIG_LOG_DEFAULT_LEVEL* and *CONFIG_BOOTLOADER_LOG_LEVEL* has a large impact on startup time. To enable more logging after the app starts up, set the *CONFIG_LOG_MAXIMUM_LEVEL* as well and then call `esp_log_level_set()` to restore higher level logs. The system/startup_time main function shows how to do this.
- Setting *CONFIG_BOOTLOADER_SKIP_VALIDATE_ON_POWER_ON* will skip verifying the binary on every boot from power-on reset. How much time this saves depends on the binary size and the flash settings. Note that this setting carries some risk if the flash becomes corrupt unexpectedly. Read the help text of the *config item* for an explanation and recommendations if using this option.
- It's possible to save a small amount of time during boot by disabling RTC slow clock calibration. To do so, set *CONFIG_RTC_CLK_CAL_CYCLES* to 0. Any part of the firmware that uses RTC slow clock as a timing source will be less accurate as a result.

The example project system/startup_time is pre-configured to optimize startup time. The file system/startup_time/sdkconfig.defaults contain all of these settings. You can append these to the end of your project's own `sdkconfig` file to merge the settings, but please read the documentation for each setting first.

Task Priorities As ESP-IDF FreeRTOS is a real-time operating system, it's necessary to ensure that high throughput or low latency tasks are granted a high priority in order to run immediately. Priority is set when calling `xTaskCreate()` or `xTaskCreatePinnedToCore()` and can be changed at runtime by calling `vTaskPrioritySet()`.

It's also necessary to ensure that tasks yield CPU (by calling `vTaskDelay()`, `sleep()`, or by blocking on semaphores, queues, task notifications, etc) in order to not starve lower priority tasks and cause problems for the overall system. The *Task Watchdog Timer (TWDT)* provides a mechanism to automatically detect if task starvation happens, however note that a Task WDT timeout does not always indicate a problem (sometimes the correct operation of the firmware requires some long-running computation). In these cases tweaking the Task WDT timeout or even disabling the Task WDT may be necessary.

Built-In Task Priorities ESP-IDF starts a number of system tasks at fixed priority levels. Some are automatically started during the boot process, some are started only if the application firmware initializes a particular feature. To optimize performance, structure application task priorities so that they are not delayed by system tasks, while also not starving system tasks and impacting other functions of the system.

This may require splitting up a particular task. For example, perform a time-critical operation in a high priority task or an interrupt handler and do the non-time-critical part in a lower priority task.

Header components/esp_system/include/esp_task.h contains macros for the priority levels used for built-in ESP-IDF tasks system. See *Background Tasks* for more details about the system tasks.

Common priorities are:

- *Main task that executes app_main function* has minimum priority (1).
- *High Resolution Timer (ESP Timer)* system task to manage timer events and execute callbacks has high priority (22, `ESP_TASK_TIMER_PRIO`)
- FreeRTOS Timer Task to handle FreeRTOS timer callbacks is created when the scheduler initializes and has minimum task priority (1, *configurable*).
- *Event Loop Library* system task to manage the default system event loop and execute callbacks has high priority (20, `ESP_TASK_EVENT_PRIO`). This configuration is only used if the application calls `esp_event_loop_create_default()`, it's possible to call `esp_event_loop_create()` with a custom task configuration instead.
- *lwIP* TCP/IP task has high priority (18, `ESP_TASK_TCPIP_PRIO`).
- *Wi-Fi Driver* task has high priority (23).
- Wi-Fi wpa_supplicant component may create dedicated tasks while the Wi-Fi Protected Setup (WPS), WPA2 EAP-TLS, Device Provisioning Protocol (DPP) or BSS Transition Management (BTM) features are in use. These tasks all have low priority (2).
- *Bluetooth Controller* task has high priority (23, `ESP_TASK_BT_CONTROLLER_PRIO`). The Bluetooth Controller needs to respond to requests with low latency, so it should always be close to the highest priority task in the system.

- *NimBLE Bluetooth Host* host task has high priority (21).
- The Ethernet driver creates a task for the MAC to receive Ethernet frames. If using the default config `ETH_MAC_DEFAULT_CONFIG` then the priority is medium-high (15). This setting can be changed by passing a custom `eth_mac_config_t` struct when initializing the Ethernet MAC.
- If using the *MQTT* component, it creates a task with default priority 5 (*configurable*, depends on *CONFIG_MQTT_USE_CUSTOM_CONFIG* (also configurable runtime by `task_prio` field in the `esp_mqtt_client_config_t`)
- To see what is the task priority for `mDNS` service, please check Performance Optimization.

Choosing application task priorities In general, it's not recommended to set task priorities higher than the built-in Wi-Fi/BT operations as starving them of CPU may make the system unstable. For very short timing-critical operations that don't use the network, use an ISR or a very restricted task (very short bursts of runtime only) at highest priority (24). Choosing priority 19 will allow lower layer Wi-Fi/BT functionality to run without delays, but still preempts the lwIP TCP/IP stack and other less time-critical internal functionality - this is the best option for time-critical tasks that don't perform network operations. Any task that does TCP/IP network operations should run at lower priority than the lwIP TCP/IP task (18) to avoid priority inversion issues.

备注： Task execution is always completely suspended when writing to the built-in SPI flash chip. Only *IRAM 安全中断处理程序* will continue executing.

Improving Interrupt Performance ESP-IDF supports dynamic *Interrupt allocation* with interrupt preemption. Each interrupt in the system has a priority, and higher priority interrupts will preempt lower priority ones.

Interrupt handlers will execute in preference to any task (provided the task is not inside a critical section). For this reason, it's important to minimize the amount of time spent executing in an interrupt handler.

To obtain the best performance for a particular interrupt handler:

- Assign more important interrupts a higher priority using a flag such as `ESP_INTR_FLAG_LEVEL2` or `ESP_INTR_FLAG_LEVEL3` when calling `esp_intr_alloc()`.
- If you're sure the entire interrupt handler can run from IRAM (see *IRAM 安全中断处理程序*) then set the `ESP_INTR_FLAG_IRAM` flag when calling `esp_intr_alloc()` to assign the interrupt. This prevents it being temporarily disabled if the application firmware writes to the internal SPI flash.
- Even if the interrupt handler is not IRAM safe, if it is going to be executed frequently then consider moving the handler function to IRAM anyhow. This minimizes the chance of a flash cache miss when the interrupt code is executed (see *Targeted Optimizations*). It's possible to do this without adding the `ESP_INTR_FLAG_IRAM` flag to mark the interrupt as IRAM-safe, if only part of the handler is guaranteed to be in IRAM.

Improving Network Speed

- For Wi-Fi, see *如何提高 Wi-Fi 性能* and *Wi-Fi 缓冲区使用情况*
- For lwIP TCP/IP (Wi-Fi and Ethernet), see *Performance Optimization*
- The wifi/iperf example contains a configuration that is heavily optimized for Wi-Fi TCP/IP throughput. Append the contents of the files wifi/iperf/sdkconfig.defaults, wifi/iperf/sdkconfig.defaults.esp32c2 and wifi/iperf/sdkconfig.ci.99 to your project `sdkconfig` file in order to add all of these options. Note that some of these options may have trade-offs in terms of reduced debuggability, increased firmware size, increased memory usage, or reduced performance of other features. To get the best result, read the documentation pages linked above and use this information to determine exactly which options are best suited for your app.

Improving I/O performance Using standard C library functions like `fread` and `fwrite` instead of platform specific unbuffered syscalls such as `read` and `write` can be slow. These functions are designed to be portable, so they are not necessarily optimized for speed, have a certain overhead and are buffered.

FatFS specific information and tips:

- Maximum size of the R/W request == FatFS cluster size (allocation unit size)
- Use `read` and `write` instead of `fread` and `fwrite`
- To increase speed of buffered reading functions like `fread` and `fgets`, you can increase a size of the file buffer (Newlib's default is 128 bytes) to a higher number like 4096, 8192 or 16384. This can be done locally via `setvbuf` function used on a certain file pointer or globally applied to all files via modifying *CONFIG_FATFS_VFS_FSTAT_BLKSIZE*.

> **备注：** Setting a bigger buffer size will also increase the heap memory usage.

Minimizing Binary Size

The ESP-IDF build system compiles all source files in the project and ESP-IDF, but only functions and variables that are actually referenced by the program are linked into the final binary. In some cases, it is necessary to reduce the total size of the firmware binary (for example, in order to fit it into the available flash partition size).

The first step to reducing the total firmware binary size is measuring what is causing the size to increase.

Measuring Static Sizes To optimize both firmware binary size and memory usage it's necessary to measure statically allocated RAM ("data", "bss"), code ("text") and read-only data ("rodata") in your project.

Using the *idf.py* sub-commands `size`, `size-components` and `size-files` provides a summary of memory used by the project:

> **备注：** It is possible to add `-DOUTPUT_FORMAT=csv` or `-DOUTPUT_FORMAT=json` to get the output in CSV or JSON format.

Size Summary (idf.py size)

```
$ idf.py size
[...]
Total sizes:
DRAM .data size:   11584 bytes
DRAM .bss  size:   19624 bytes
Used static DRAM:       0 bytes (      0 available, nan% used)
Used static IRAM:       0 bytes (      0 available, nan% used)
Used stat D/IRAM:  136276 bytes ( 519084 available, 20.8% used)
      Flash code:  630508 bytes
    Flash rodata:  177048 bytes
Total image size:~ 924208 bytes (.bin may be padded larger)
```

This output breaks down the size of all static memory regions in the firmware binary:

- `DRAM .data size` is statically allocated RAM that is assigned to non-zero values at startup. This uses RAM (DRAM) at runtime and also uses space in the binary file.
- `DRAM .bss size` is statically allocated RAM that is assigned zero at startup. This uses RAM (DRAM) at runtime but doesn't use any space in the binary file.
- `Used static DRAM`, `Used static IRAM` - these options are kept for compatibility with ESP32 target, and currently read 0.
- `Used stat D/IRAM` - This is total internal RAM usage, the sum of static DRAM .data + .bss, and also static *IRAM（指令 RAM）* used by the application for executable code. The `available` size is the estimated amount of DRAM which will be available as heap memory at runtime (due to metadata overhead and implementation constraints, and heap allocations done by ESP-IDF during startup, the actual free heap at startup will be lower than this).

- `Flash code` is the total size of executable code executed from flash cache (*IROM*). This uses space in the binary file.
- `Flash rodata` is the total size of read-only data loaded from flash cache (*DROM*). This uses space in the binary file.
- `Total image size` is the estimated total binary file size, which is the total of all the used memory types except for .bss.

Component Usage Summary (idf.py size-components) The summary output provided by `idf.py size` does not give enough detail to find the main contributor to excessive binary size. To analyze in more detail, use `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
```

The first lines of output from `idf.py size-components` are the same as `idf.py size`. After this a table is printed of "per-archive contributions to ELF file". This means how much each static library archive has contributed to the final binary size.

Generally, one static library archive is built per component, although some are binary libraries included by a particular component (for example, `libnet80211.a` is included by `esp_wifi` component). There are also toolchain libraries such as `libc.a` and `libgcc.a` listed here, these provide Standard C/C++ Library and toolchain built-in functionality.

If your project is simple and only has a "main" component, then all of the project's code will be shown under `libmain.a`. If your project includes its own components (see 构建系统), then they will each be shown on a separate line.

The table is sorted in descending order of the total contribution to the binary size.

The columns are as follows:

- `DRAM .data & .bss & other` - .data and .bss are the same as for the totals shown above (static variables, these both reduce total available RAM at runtime but .bss doesn't contribute to the binary file size). "other" is a column for any custom section types that also contribute to RAM size (usually this value is 0).
- `IRAM` - is the same as for the totals shown above (code linked to execute from IRAM, uses space in the binary file and also reduces DRAM available as heap at runtime.
- `Flash code & rodata` - these are the same as the totals above, IROM and DROM space accessed from flash cache that contribute to the binary size.

Source File Usage Summary (idf.py size-files) For even more detail, run `idf.py size-files` to get a summary of the contribution each object file has made to the final binary size. Each object file corresponds to a single source file.

```
$ 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
```

(下页继续)

(续上页)

```
        lib_a-vfprintf.o           0       0       0       0       0   13829       
↪752   14581
       lib_a-svfprintf.o           0       0       0       0       0   13251       
↪752   14003
            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 ...]
```

After the summary of total sizes, a table of "Per-file contributions to ELF file" is printed.

The columns are the same as shown above for `idy.py size-components`, but this time the granularity is the contribution of each individual object file to the binary size.

For example, we can see that the file `x509_crt_bundle.S.o` contributed 64212 bytes to the total firmware size, all as `.rodata` in flash. Therefore we can guess that this application is using the *ESP x509 Certificate Bundle* feature and not using this feature would save at last this many bytes from the firmware size.

Some of the object files are linked from binary libraries and therefore you won't find a corresponding source file. To locate which component a source file belongs to, it's generally possible to search in the ESP-IDF source tree or look in the *Linker Map File* for the full path.

Comparing Two Binaries If making some changes that affect binary size, it's possible to use an ESP-IDF tool to break down the exact differences in size.

This operation isn't part of `idf.py`, it's necessary to run the esp_idf_size Python tool directly.

To do so, first locate the linker map file in the build directory. It will have the name `PROJECTNAME.map`. The `esp_idf_size` tool performs its analysis based on the output of the linker map file.

To compare with another binary, you will also need its corresponding `.map` file saved from the build directory.

For example, to compare two builds: one with the default *CONFIG_COMPILER_OPTIMIZATION* setting "Debug (-Og)" configuration and one with "Optimize for size (-Os)":

```
$ 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
```

We can see from the "Difference" column that changing this one setting caused the whole binary to be over 60 KB smaller and over 5 KB more RAM is available.

It's also possible to use the "diff" mode to output a table of component-level (static library archive) differences:

备注: To get the output in JSON or CSV format using `esp_idf_size` it is possible to use the `--format` option.

```
python -m esp_idf_size --archives --diff build_Og/https_request.map build_Oshttps_
↪request.map
```

Also at the individual source file level:

```
python -m esp_idf_size --files --diff build_Og/https_request.map build_Oshttps_
↪request.map
```

Other options (like writing the output to a file) are available, pass `--help` to see the full list.

Showing Size When Linker Fails If too much static memory is used, then the linker will fail with an error such as `DRAM segment data does not fit, region `iram0_0_seg' overflowed by 44 bytes`, or similar.

In these cases, `idf.py size` will not succeed either. However it is possible to run `esp_idf_size` manually in order to view the *partial static memory usage* (the memory usage will miss the variables which could not be linked, so there still appears to be some free space.)

The map file argument is `<projectname>.map` in the build directory

```
python -m esp_idf_size build/project_name.map
```

It is also possible to view the equivalent of `size-components` or `size-files` output:

```
python -m esp_idf_size --archives build/project_name.map
python -m esp_idf_size --files build/project_name.map
```

Linker Map File *This is an advanced analysis method, but it can be very useful. Feel free to skip ahead to :ref:'reducing-overall-size' and possibly come back to this later.*

The `idf.py size` analysis tools all work by parsing the GNU binutils "linker map file", which is a summary of everything the linker did when it created ("linked") the final firmware binary file

Linker map files themselves are plain text files, so it's possible to read them and find out exactly what the linker did. However, they are also very complex and long - often 100,000 or more lines!

The map file itself is broken into parts and each part has a heading. The parts are:

- `Archive member included to satisfy reference by file (symbol).` This shows you: for each object file included in the link, what symbol (function or variable) was the linker searching for when it included that object file. If you're wondering why some object file in particular was included in the binary, this part may give a clue. This part can be used in conjunction with the `Cross Reference Table` at the end of the file. Note that not every object file shown in this list ends up included in the final binary, some end up in the `Discarded input sections` list instead.
- `Allocating common symbols` - This is a list of (some) global variables along with their sizes. Common symbols have a particular meaning in ELF binary files, but ESP-IDF doesn't make much use of them.
- `Discarded input sections` - These sections were read by the linker as part of an object file to be linked into the final binary, but then nothing else referred to them so they were discarded from the final binary. For ESP-IDF this list can be very long, as we compile each function and static variable to a unique section in order to minimize the final binary size (specifically ESP-IDF uses compiler options `-ffunction-sections -fdata-sections` and linker option `--gc-sections`). Items mentioned in this list *do not* contribute to the final binary.
- `Memory Configuration`, `Linker script and memory map` These two parts go together. Some of the output comes directly from the linker command line and the Linker Script, both provided by the 构建系统. The linker script is partially generated from the ESP-IDF project using the 链接器脚本生成机制 feature. As the output of the `Linker script and memory map` part of the map unfolds, you can see each symbol (function or static variable) linked into the final binary along with its address (as a 16 digit hex number), its length (also in hex), and the library and object file it was linked from (which can be used to determine the component and the source file).
  Following all of the output sections that take up space in the final `.bin` file, the `memory map` also includes some sections in the ELF file that are only used for debugging (ELF sections `.debug_*`, etc.). These don't contribute to the final binary size. You'll notice the address of these symbols is a very low number (starting from 0x0000000000000000 and counting up).
- `Cross Reference Table`. This table shows for each symbol (function or static variable), the list of object file(s) that referred to it. If you're wondering why a particular thing is included in the binary, this will help determine what included it.

  备注: Unfortunately, the `Cross Reference Table` doesn't only include symbols that made it into the final binary. It also includes symbols in discarded sections. Therefore, just because something is shown here doesn't mean that it was included in the final binary - this needs to be checked separately.

备注: Linker map files are generated by the GNU binutils linker "ld", not ESP-IDF. You can find additional information online about the linker map file format. This quick summary is written from the perspective of ESP-IDF build system in particular.

Reducing Overall Size The following configuration options will reduce the final binary size of almost any ESP-IDF project:

- Set *CONFIG_COMPILER_OPTIMIZATION* to "Optimize for size (-Os)". In some cases, "Optimize for performance (-O2)" will also reduce the binary size compared to the default. Note that if your code contains C or C++ Undefined Behaviour then increasing the compiler optimization level may expose bugs that otherwise don't happen.
- Reduce the compiled-in log output by lowering the app *CONFIG_LOG_DEFAULT_LEVEL*. If the *CONFIG_LOG_MAXIMUM_LEVEL* is changed from the default then this setting controls the binary size instead. Reducing compiled-in logging reduces the number of strings in the binary, and also the code size of the calls to logging functions.
- Set the *CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL* to "Silent". This avoids compiling in a dedicated assertion string and source file name for each assert that may fail. It's still possible to find the failed assert in the code by looking at the memory address where the assertion failed.
- Besides the *CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL*, you can disable or silent the assertion for HAL component separately by setting *CONFIG_HAL_DEFAULT_ASSERTION_LEVEL*. It is to notice that ESP-IDF lowers HAL assertion level in bootloader to be silent even if *CONFIG_HAL_DEFAULT_ASSERTION_LEVEL* is set to full-assertion level. This is to reduce the bootloader size.
- Set *CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT*. This removes specific error messages for particular internal ESP-IDF error check macros. This may make it harder to debug some error conditions by reading the log output.
- Don't enable *CONFIG_COMPILER_CXX_EXCEPTIONS*, *CONFIG_COMPILER_CXX_RTTI*, or set the *CONFIG_COMPILER_STACK_CHECK_MODE* to Overall. All of these options are already disabled by default, but they have a large impact on binary size.
- Disabling *CONFIG_ESP_ERR_TO_NAME_LOOKUP* will remove the lookup table to translate user-friendly names for error values (see 错误处理) in error logs, etc. This saves some binary size, but error values will be printed as integers only.
- Setting *CONFIG_ESP_SYSTEM_PANIC* to "Silent reboot" will save a small amount of binary size, however this is *only* recommended if no one will use UART output to debug the device.
- Set *CONFIG_COMPILER_SAVE_RESTORE_LIBCALLS* to reduce binary size by replacing inlined prologues/epilogues with library calls.
- If the application binary uses only one of the security versions of the protocomm component, then the support for others can be disabled to save some code size. The support can be disabled through *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0*, *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1* or *CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2* respectively.

备注: In addition to the many configuration items shown here, there are a number of configuration options where changing the option from the default will increase binary size. These are not noted here. Where the increase is significant, this is usually noted in the configuration item help text.

Targeted Optimizations The following binary size optimizations apply to a particular component or a function:

Wi-Fi

- Disabling *CONFIG_ESP_WIFI_ENABLE_WPA3_SAE* will save some Wi-Fi binary size if WPA3 support is not needed. (Note that WPA3 is mandatory for new Wi-Fi device certifications.)
- Disabling *CONFIG_ESP_WIFI_SOFTAP_SUPPORT* will save some Wi-Fi binary size if soft-AP support is not needed.

Bluetooth NimBLE If using *NimBLE Bluetooth Host* then the following modifications can reduce binary size:

- *CONFIG_BT_NIMBLE_MAX_CONNECTIONS* to 1 if only one BLE connection is needed.

- Disable either *CONFIG_BT_NIMBLE_ROLE_CENTRAL* or *CONFIG_BT_NIMBLE_ROLE_OBSERVER* if these roles are not needed.
- Reducing *CONFIG_BT_NIMBLE_LOG_LEVEL* can reduce binary size. Note that if the overall log level has been reduced as described above in *Reducing Overall Size* then this also reduces the NimBLE log level.

lwIP IPv6

- Setting *CONFIG_LWIP_IPV6* to false will reduce the size of the lwIP TCP/IP stack, at the cost of only supporting IPv4.

备注： IPv6 is required by some components such as `coap` and *ASIO port*, These components will not be available if IPV6 is disabled.

---

lwIP IPv4

- If IPv4 connectivity is not required, setting *CONFIG_LWIP_IPV4* to false will reduce the size of the lwIP, supporting IPv6 only TCP/IP stack.

备注： Before disabling IPv4 support, please note that IPv6 only network environments are not ubiquitous and must be supported in the local network, e.g. by your internet service provider or using constrained local network settings.

Newlib nano formatting By default, ESP-IDF uses newlib "full" formating for I/O (printf, scanf, etc.)

Enabling the config option *CONFIG_NEWLIB_NANO_FORMAT* will switch newlib to the "nano" formatting mode. This both smaller in code size and a large part of the implementation is compiled into the ESP32-C2 ROM, so it doesn't need to be included in the binary at all.

The exact difference in binary size depends on which features the firmware uses, but 25 KB ~ 50 KB is typical.

Enabling Nano formatting reduces the stack usage of each function that calls printf() or another string formatting function, see *Reducing Stack Sizes*.

"Nano" formatting doesn't support 64-bit integers, or C99 formatting features. For a full list of restrictions, search for `--enable-newlib-nano-formatted-io` in the Newlib README file.

备注： *CONFIG_NEWLIB_NANO_FORMAT* is enabled by default on ESP32-C2

---

mbedTLS features Under *Component Config -> mbedTLS* there are multiple mbedTLS features which are enabled by default but can be disabled if not needed to save code size.

These include:

- *CONFIG_MBEDTLS_HAVE_TIME*
- *CONFIG_MBEDTLS_ECDSA_DETERMINISTIC*
- *CONFIG_MBEDTLS_SHA512_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* (Alternatively: Leave this option enabled but disable some of the elliptic curves listed in the sub-menu.)
- Change *CONFIG_MBEDTLS_TLS_MODE* if both server & client functionalities are not needed

- Consider disabling some ciphersuites listed in the "TLS Key Exchange Methods" sub-menu (i.e. *CONFIG_MBEDTLS_KEY_EXCHANGE_RSA*)

The help text for each option has some more information.

重要： It is **strongly not recommended to disable all these mbedTLS options**. Only disable options where you understand the functionality and are certain that it is not needed in the application. In particular:

- Ensure that any TLS server(s) the device connects to can still be used. If the server is controlled by a third party or a cloud service, recommend ensuring that the firmware supports at least two of the supported cipher suites in case one is disabled in a future update.
- Ensure that any TLS client(s) that connect to the device can still connect with supported/recommended cipher suites. Note that future versions of client operating systems may remove support for some features, so it is recommended to enable multiple supported cipher suites or algorithms for redundancy.

If depending on third party clients or servers, always pay attention to announcements about future changes to supported TLS features. If not, the ESP32-C2 device may become inaccessible if support changes.

备注： Not every combination of mbedTLS compile-time config is tested in ESP-IDF. If you find a combination that fails to compile or function as expected, please report the details on GitHub.

VFS *Virtual filesystem* feature in ESP-IDF allows multiple filesystem drivers and file-like peripheral drivers to be accessed using standard I/O functions (`open`, `read`, `write`, etc.) and C library functions (`fopen`, `fread`, `fwrite`, etc.). When filesystem or file-like peripheral driver functionality is not used in the application this feature can be fully or partially disabled. VFS component provides the following configuration options:

- *CONFIG_VFS_SUPPORT_TERMIOS* —can be disabled if the application doesn't use `termios` family of functions. Currently, these functions are implemented only for UART VFS driver. Most applications can disable this option. Disabling this option reduces the code size by about 1.8 kB.
- *CONFIG_VFS_SUPPORT_SELECT* — can be disabled if the application doesn't use `select` function with file descriptors. Currently, only the UART and eventfd VFS drivers implement `select` support. Note that when this option is disabled, `select` can still be used for socket file descriptors. Disabling this option reduces the code size by about 2.7 kB.
- *CONFIG_VFS_SUPPORT_DIR* —can be disabled if the application doesn't use directory related functions, such as `readdir` (see the description of this option for the complete list). Applications which only open, read and write specific files and don't need to enumerate or create directories can disable this option, reducing the code size by 0.5 kB or more, depending on the filesystem drivers in use.
- *CONFIG_VFS_SUPPORT_IO* —can be disabled if the application doesn't use filesystems or file-like peripheral drivers. This disables all VFS functionality, including the three options mentioned above. When this option is disabled, *console* can't be used. Note that the application can still use standard I/O functions with socket file descriptors when this option is disabled. Compared to the default configuration, disabling this option reduces code size by about 9.4 kB.

HAL

- Enabling *CONFIG_HAL_SYSTIMER_USE_ROM_IMPL* can reduce the IRAM usage and binary size by linking in the systimer HAL driver of ROM implementation.
- Enabling *CONFIG_HAL_WDT_USE_ROM_IMPL* can reduce the IRAM usage and binary size by linking in the watchdog HAL driver of ROM implementation.

Heap

- Enabling *CONFIG_HEAP_PLACE_FUNCTION_INTO_FLASH* can reduce the IRAM usage and binary size by placing the entirety of the heap functionalities in flash memory.
- Enabling *CONFIG_HEAP_TLSF_USE_ROM_IMPL* can reduce the IRAM usage and binary size by linking in the TLSF library of ROM implementation.

Bootloader Size This document deals with the size of an ESP-IDF app binary only, and not the ESP-IDF 二级引导程序.

For a discussion of ESP-IDF bootloader binary size, see 引导加载程序大小.

IRAM Binary Size If the IRAM section of a binary is too large, this issue can be resolved by reducing IRAM memory usage. See *Optimizing IRAM Usage*.

Minimizing RAM Usage

In some cases, a firmware application's available RAM may run low or run out entirely. In these cases, it's necessary to tune the memory usage of the firmware application.

In general, firmware should aim to leave some "headroom" of free internal RAM in order to deal with extraordinary situations or changes in RAM usage in future updates.

Background Before optimizing ESP-IDF RAM usage, it's necessary to understand the basics of ESP32-C2 memory types, the difference between static and dynamic memory usage in C, and the way ESP-IDF uses stack and heap. This information can all be found in *Heap Memory Allocation*.

Measuring Static Memory Usage The *idf.py* tool can be used to generate reports about the static memory usage of an application. Refer to *the Binary Size chapter for more information*.

Measuring Dynamic Memory Usage ESP-IDF contains a range of heap APIs for measuring free heap at runtime. See *Heap Memory Debugging*.

备注: In embedded systems, heap fragmentation can be a significant issue alongside total RAM usage. The heap measurement APIs provide ways to measure the "largest free block". Monitoring this value along with the total number of free bytes can give a quick indication of whether heap fragmentation is becoming an issue.

Reducing Static Memory Usage

- Reducing the static memory usage of the application increases the amount of RAM available for heap at runtime, and vice versa.
- Generally speaking, minimizing static memory usage requires monitoring the .data and .bss sizes. For tools to do this, see *Measuring Static Sizes*.
- Internal ESP-IDF functions do not make heavy use of static RAM allocation in C. In many instances (including: Wi-Fi library, Bluetooth controller) "static" buffers are still allocated from heap, but the allocation is done once when the feature is initialized and will be freed if the feature is deinitialized. This is done in order to maximize the amount of free memory at different points in the application life-cycle.

To minimize static memory use:

- Declare structures, buffers, or other variables `const` whenever possible. Constant data can be stored in flash not RAM. This may require changing functions in the firmware to take `const *` arguments instead of mutable pointer arguments. These changes can also reduce the stack usage of some functions.
- If using Bluedroid, setting the option *CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY* will cause Bluedroid to allocate memory on initialization and free it on deinitialization. This doesn't necessarily reduce the peak memory usage, but changes it from static memory usage to runtime memory usage.

Reducing Stack Sizes In FreeRTOS, task stacks are usually allocated from the heap. The stack size for each task is fixed (passed as an argument to *xTaskCreate()*). Each task can use up to its allocated stack size, but using more than this will cause an otherwise valid program to crash with a stack overflow or heap corruption.

Therefore, determining the optimum sizes of each task stack can substantially reduce RAM usage.

To determine optimum task stack sizes:

- Combine tasks. The best task stack size is 0 bytes, achieved by combining a task with another existing task. Anywhere that the firmware can be structured to perform multiple functions sequentially in a single task will increase free memory. In some cases, using a "worker task" pattern where jobs are serialized into a FreeRTOS queue (or similar) and then processed by generic worker tasks may help.
- Consolidate task functions. String formatting functions (like `printf`) are particularly heavy users of stack, so any task which doesn't ever call these can usually have its stack size reduced.
- Enabling *Newlib nano formatting* will reduce the stack usage of any task that calls `printf()` or other C string formatting functions.
- Avoid allocating large variables on the stack. In C, any large struct or array allocated as an "automatic" variable (i.e. default scope of a C declaration) will use space on the stack. Minimize the sizes of these, allocate them statically and/or see if you can save memory by allocating them from the heap only when they are needed.
- Avoid deep recursive function calls. Individual recursive function calls don't always add a lot of stack usage each time they are called, but if each function includes large stack-based variables then the overhead can get quite high.
- At runtime, call the function *uxTaskGetStackHighWaterMark()* with the handle of any task where you think there is unused stack memory. This function returns the minimum lifetime free stack memory in bytes. The easiest time to call this is from the task itself: call `uxTaskGetStackHighWaterMark(NULL)` to get the current task's high water mark after the time that the task has achieved its peak stack usage (i.e. if there is a main loop, execute the main loop a number of times with all possible states and then call *uxTaskGetStackHighWaterMark()*). Often, it's possible to subtract almost the entire value returned here from the total stack size of a task, but allow some safety margin to account for unexpected small increases in stack usage at runtime.
- Call *uxTaskGetSystemState()* at runtime to get a summary of all tasks in the system. This includes their individual stack "high watermark" values.
- When debugger watchpoints are not being used, set the *CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK* option to trigger an immediate panic if a task writes the word at the end of its assigned stack. This is slightly more reliable than the default *CONFIG_FREERTOS_CHECK_STACKOVERFLOW* option of "Check using canary bytes", because the panic happens immediately, not on the next RTOS context switch. Neither option is perfect, it's possible in some cases for stack pointer to skip the watchpoint or canary bytes and corrupt another region of RAM, instead.

Internal Stack Sizes ESP-IDF allocates a number of internal tasks for housekeeping purposes or operating system functions. Some are created during the startup process, and some are created at runtime when particular features are initialized.

The default stack sizes for these tasks are usually set conservatively high, to allow all common usage patterns. Many of the stack sizes are configurable, and it may be possible to reduce them to match the real runtime stack usage of the task.

重要： If internal task stack sizes are set too small, ESP-IDF will crash unpredictably. Even if the root cause is task stack overflow, this is not always clear when debugging. It is recommended that internal stack sizes are only reduced carefully (if at all), with close attention to "high water mark" free space under load. If reporting an issue that occurs when internal task stack sizes have been reduced, please always include this information and the specific configuration that is being used.

- *Main task that executes app_main function* has stack size *CONFIG_ESP_MAIN_TASK_STACK_SIZE*.
- *High Resolution Timer (ESP Timer)* system task which executes callbacks has stack size *CONFIG_ESP_TIMER_TASK_STACK_SIZE*.

- FreeRTOS Timer Task to handle FreeRTOS timer callbacks has stack size *CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH*.
- *Event Loop Library* system task to execute callbacks for the default system event loop has stack size *CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE*.
- *lwIP* TCP/IP task has stack size *CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*
- *Bluedroid Bluetooth Host* have task stack sizes *CONFIG_BT_BTC_TASK_STACK_SIZE*, *CONFIG_BT_BTU_TASK_STACK_SIZE*.
- *NimBLE Bluetooth Host* has task stack size *CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE*
- The Ethernet driver creates a task for the MAC to receive Ethernet frames. If using the default config `ETH_MAC_DEFAULT_CONFIG` then the task stack size is 4 KB. This setting can be changed by passing a custom `eth_mac_config_t` struct when initializing the Ethernet MAC.
- FreeRTOS idle task stack size is configured by *CONFIG_FREERTOS_IDLE_TASK_STACKSIZE*.
- If using the *MQTT* component, it creates a task with stack size configured by *CONFIG_MQTT_TASK_STACK_SIZE*. MQTT stack size can also be configured using `task_stack` field of `esp_mqtt_client_config_t`.
- To see how to optimize RAM usage when using `mDNS`, please check Performance Optimization.

备注： Aside from built-in system features such as esp-timer, if an ESP-IDF feature is not initialized by the firmware then no associated task is created. In those cases, the stack usage is zero and the stack size configuration for the task is not relevant.

Reducing Heap Usage For functions that assist in analyzing heap usage at runtime, see *Heap Memory Debugging*.

Normally, optimizing heap usage consists of analyzing the usage and removing calls to `malloc()` that aren't being used, reducing the corresponding sizes, or freeing previously allocated buffers earlier.

There are some ESP-IDF configuration options that can reduce heap usage at runtime:

- lwIP documentation has a section to configure *Minimum RAM usage*.
- *Wi-Fi 缓冲区使用情况* describes options to either reduce numbers of "static" buffers or reduce the maximum number of "dynamic" buffers in use, in order to minimize memory usage at possible cost of performance. Note that "static" Wi-Fi buffers are still allocated from heap when Wi-Fi is initialized and will be freed if Wi-Fi is deinitialized.
- Several Mbed TLS configuration options can be used to reduce heap memory usage. See the *Mbed TLS* docs for details.

备注： There are other configuration options that will increase heap usage at runtime if changed from the defaults. These are not listed here, but the help text for the configuration item will mention if there is some memory impact.

Optimizing IRAM Usage The available DRAM at runtime (for heap usage) is also reduced by the static IRAM usage. Therefore, one way to increase available DRAM is to reduce IRAM usage.

If the app allocates more static IRAM than is available then the app will fail to build and linker errors such as `section `.iram0.text' will not fit in region `iram0_0_seg', IRAM0 segment data does not fit` and `region `iram0_0_seg' overflowed by 84 bytes` will be seen. If this happens, it is necessary to find ways to reduce static IRAM usage in order to link the application.

To analyze the IRAM usage in the firmware binary, use *Measuring Static Sizes*. If the firmware failed to link, steps to analyze are shown at *Showing Size When Linker Fails*.

The following options will reduce IRAM usage of some ESP-IDF features:

- Enable *CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH*. Provided these functions are not (incorrectly) used from ISRs, this option is safe to enable in all configurations.

- Enable *CONFIG_FREERTOS_PLACE_SNAPSHOT_FUNS_INTO_FLASH*. Enabling this option will place snapshot-related functions, such as `vTaskGetSnapshot` or `uxTaskGetSnapshotAll`, in flash.
- Enable *CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH*. Provided these functions are not (incorrectly) used from ISRs, this option is safe to enable in all configurations.
- Enable *CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH*. This option is not safe to use if the ISR ringbuf functions are used from an IRAM interrupt context, e.g. if *CONFIG_UART_ISR_IN_IRAM* is enabled. For the IDF drivers where this is the case you will get an error at run-time when installing the driver in question.
- Disable Wi-Fi options *CONFIG_ESP_WIFI_IRAM_OPT* and/or *CONFIG_ESP_WIFI_RX_IRAM_OPT*. Disabling these options will free available IRAM at the cost of Wi-Fi performance.
- *CONFIG_SPI_FLASH_ROM_IMPL* enabling this option will free some IRAM but will mean that esp_flash bugfixes and new flash chip support is not available, see *SPI Flash API ESP-IDF version vs Chip-ROM version* for details.
- Disabling *CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR* prevents posting `esp_event` events from *IRAM 安全中断处理程序* but will save some IRAM.
- Disabling *CONFIG_SPI_MASTER_ISR_IN_IRAM* prevents spi_master interrupts from being serviced while writing to flash, and may otherwise reduce spi_master performance, but will save some IRAM.
- Disabling *CONFIG_SPI_SLAVE_ISR_IN_IRAM* prevents spi_slave interrupts from being serviced while writing to flash, will save some IRAM.
- Setting *CONFIG_HAL_DEFAULT_ASSERTION_LEVEL* to disable assertion for HAL component will save some IRAM especially for HAL code who calls *HAL_ASSERT* a lot and resides in IRAM.
- Refer to sdkconfig menu `Auto-detect flash chips` and you can disable flash drivers which you don't need to save some IRAM.
- Enable *CONFIG_HEAP_PLACE_FUNCTION_INTO_FLASH*. Provided that *CONFIG_SPI_MASTER_ISR_IN_IRAM* is not enabled and the heap functions are not (incorrectly) used from ISRs, this option is safe to enable in all configuration.

备注： Moving frequently-called functions from IRAM to flash may increase their execution time.

备注： Other configuration options exist that will increase IRAM usage by moving some functionality into IRAM, usually for performance, but the default option is not to do this. These are not listed here. The IRAM size impact of enabling these options is usually noted in the configuration item help text.

4.19 Reproducible Builds

4.19.1 Introduction

ESP-IDF build system has support for reproducible builds.

When reproducible builds are enabled, the application built with ESP-IDF doesn't 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

4.19.2 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.

4.19.3 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.

4.19.4 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* 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.

4.19.5 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.

4.19.6 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 Image can be used to ensure that these factors do not affect the build.

4.20 RF calibration

ESP32-C2 supports three RF calibration methods during RF initialization:

1. Partial calibration
2. Full calibration
3. No calibration

4.20.1 Partial calibration

During RF initialization, the partial calibration method is used by default for RF calibration. It is done based on the full calibration data which is stored in the NVS. To use this method, please go to `menuconfig` and enable *CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE*.

4.20.2 Full calibration

Full calibration is triggered in the follwing conditions:

1. NVS does not exist.
2. The NVS partition to store calibration data is erased.
3. Hardware MAC address is changed.
4. PHY library version is changed.
5. The RF calibration data loaded from the NVS partition is broken.

It takes about 100ms more than partial calibration. If boot duration is not critical, it is suggested to use the full calibration method. To switch to the full calibration method, go to `menuconfig` and disable *CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE*. If you use the default method of RF calibration, there are two ways to add the function of triggering full calibration as a last-resort remedy.

1. Erase the NVS partition if you don't mind all of the data stored in the NVS partition is erased. That is indeed the easiest way.
2. Call API `esp_phy_erase_cal_data_in_nvs()` before initializing WiFi and BT/BLE based on some conditions (e.g. an option provided in some diagnostic mode). In this case, only phy namespace of the NVS partition is erased.

4.20.3 No calibration

No calibration method is only used when the device wakes up from deep sleep.

4.20.4 PHY initialization data

The PHY initialization data is used for RF calibration. There are two ways to get the PHY initialization data.

One is the default initialization data which is located in the header file components/esp_phy/esp32c2/include/phy_init_data.h.

It is embedded into the application binary after compiling and then stored into read-only memory (DROM). To use the default initialization data, please go to `menuconfig` and disable *CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*.

Another is the initialization data which is stored in a partition. When using a custom partition table, make sure that PHY data partition is included (type: *data*, subtype: *phy*). With default partition table, this is done automatically. If initialization data is stored in a partition, it has to be flashed there, otherwise runtime error will occur. To switch to the initialization data stored in a partition, go to `menuconfig` and enable *CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*.

4.20.5 API Reference

Header File

- components/esp_phy/include/esp_phy_init.h

Functions

const *esp_phy_init_data_t* ***esp_phy_get_init_data**(void)

Get PHY init data.

If "Use a partition to store PHY init data" option is set in menuconfig, This function will load PHY init data from a partition. Otherwise, PHY init data will be compiled into the application itself, and this function will return a pointer to PHY init data located in read-only memory (DROM).

If "Use a partition to store PHY init data" option is enabled, this function may return NULL if the data loaded from flash is not valid.

备注: Call esp_phy_release_init_data to release the pointer obtained using this function after the call to esp_wifi_init.

返回 pointer to PHY init data structure

void **esp_phy_release_init_data**(const *esp_phy_init_data_t* *data)

Release PHY init data.

参数 **data** –pointer to PHY init data structure obtained from esp_phy_get_init_data function

esp_err_t **esp_phy_load_cal_data_from_nvs**(*esp_phy_calibration_data_t* *out_cal_data)

Function called by esp_phy_init to load PHY calibration data.

This is a convenience function which can be used to load PHY calibration data from NVS. Data can be stored to NVS using esp_phy_store_cal_data_to_nvs function.

If calibration data is not present in the NVS, or data is not valid (was obtained for a chip with a different MAC address, or obtained for a different version of software), this function will return an error.

If "Initialize PHY in startup code"option is set in menuconfig, this function will be used to load calibration data. To provide a different mechanism for loading calibration data, disable "Initialize PHY in startup code" option in menuconfig and call esp_phy_init function from the application. For an example usage of esp_phy_init and this function, see esp_phy_store_cal_data_to_nvs function in cpu_start.c

参数 **out_cal_data** –pointer to calibration data structure to be filled with loaded data.
返回 ESP_OK on success

esp_err_t **esp_phy_store_cal_data_to_nvs**(const *esp_phy_calibration_data_t* *cal_data)

Function called by esp_phy_init to store PHY calibration data.

This is a convenience function which can be used to store PHY calibration data to the NVS. Calibration data is returned by esp_phy_init function. Data saved using this function to the NVS can later be loaded using esp_phy_store_cal_data_to_nvs function.

If "Initialize PHY in startup code" option is set in menuconfig, this function will be used to store calibration data. To provide a different mechanism for storing calibration data, disable "Initialize PHY in startup code" option in menuconfig and call esp_phy_init function from the application.

参数 **cal_data** –pointer to calibration data which has to be saved.
返回 ESP_OK on success

esp_err_t **esp_phy_erase_cal_data_in_nvs**(void)

Erase PHY calibration data which is stored in the NVS.

This is a function which can be used to trigger full calibration as a last-resort remedy if partial calibration is used. It can be called in the application based on some conditions (e.g. an option provided in some diagnostic mode).

返回 ESP_OK on success
返回 others on fail. Please refer to NVS API return value error number.

void **esp_phy_enable**(void)

Enable PHY and RF module.

PHY and RF module should be enabled in order to use WiFi or BT. Now PHY and RF enabling job is done automatically when start WiFi or BT. Users should not call this API in their application.

void **esp_phy_disable**(void)

Disable PHY and RF module.

PHY module should be disabled in order to shutdown WiFi or BT. Now PHY and RF disabling job is done automatically when stop WiFi or BT. Users should not call this API in their application.

void **esp_btbb_enable**(void)

Enable BTBB module.

BTBB module should be enabled in order to use IEEE802154 or BT. Now BTBB enabling job is done automatically when start IEEE802154 or BT. Users should not call this API in their application.

void **esp_btbb_disable**(void)

Disable BTBB module.

Dsiable BTBB module, used by IEEE802154 or Bluetooth. Users should not call this API in their application.

void **esp_phy_load_cal_and_init**(void)

Load calibration data from NVS and initialize PHY and RF module.

void **esp_phy_modem_init**(void)

Initialize backup memory for Phy power up/down.

void **esp_phy_modem_deinit**(void)

Deinitialize backup memory for Phy power up/down Set phy_init_flag if all modems deinit on ESP32C3.

void **esp_phy_common_clock_enable**(void)

Enable WiFi/BT common clock.

void **esp_phy_common_clock_disable**(void)

Disable WiFi/BT common clock.

int64_t **esp_phy_rf_get_on_ts**(void)

Get the time stamp when PHY/RF was switched on.

返回 return 0 if PHY/RF is never switched on. Otherwise return time in microsecond since boot when phy/rf was last switched on

esp_err_t **esp_phy_update_country_info**(const char *country)

Update the corresponding PHY init type according to the country code of Wi-Fi.

参数 **country** –country code
返回 ESP_OK on success.
返回 esp_err_t code describing the error on fail

char ***get_phy_version_str**(void)

Get PHY lib version.

返回 PHY lib version.

Structures

struct **esp_phy_init_data_t**

Structure holding PHY init parameters.

Public Members

uint8_t **params**[128]

opaque PHY initialization parameters

struct **esp_phy_calibration_data_t**

Opaque PHY calibration data.

Public Members

uint8_t **version**[4]

PHY version

uint8_t **mac**[6]

The MAC address of the station

uint8_t **opaque**[1894]

calibration data

Enumerations

enum **esp_phy_calibration_mode_t**

PHY calibration mode.

Values:

enumerator **PHY_RF_CAL_PARTIAL**

Do part of RF calibration. This should be used after power-on reset.

enumerator **PHY_RF_CAL_NONE**

Don’ t do any RF calibration. This mode is only suggested to be used after deep sleep reset.

enumerator **PHY_RF_CAL_FULL**

Do full RF calibration. Produces best results, but also consumes a lot of time and current. Suggested to be used once.

Header File

- components/esp_phy/include/esp_phy_cert_test.h

Functions

void **esp_wifi_power_domain_on**(void)

Wifi power domain power on.

void **esp_wifi_power_domain_off**(void)

Wifi power domain power off.

void **esp_phy_rftest_config**(uint8_t conf)

Environment variable configuration.

参数 **conf** –Set to 1 to enter RF test mode.

void **esp_phy_rftest_init**(void)

RF initialization configuration.

void **esp_phy_tx_contin_en**(bool contin_en)

TX Continuous mode.

参数 **contin_en** –Set to true for continuous packet sending, which can be used for certification testing; Set to false to cancel continuous mode, which is the default mode and can be used for WLAN tester.

void **esp_phy_cbw40m_en**(bool en)

HT40/HT20 mode selection.

参数 **en** –Set to false to enter 11n HT20 mode; Set to true to enter 11n HT40 mode;

void **esp_phy_wifi_tx**(uint32_t chan, *esp_phy_wifi_rate_t* rate, int8_t backoff, uint32_t length_byte, uint32_t packet_delay, uint32_t packet_num)

Wi-Fi TX command.

参数

- **chan** –channel setting, 1~14;
- **rate** –rate setting;
- **backoff** –Transmit power attenuation, unit is 0.25dB. For example, 4 means that the power is attenuated by 1dB;
- **length_byte** –TX packet length configuration, indicating PSDU Length, unit is byte;
- **packet_delay** –TX packet interval configuration, unit is us;
- **packet_num** –The number of packets sent, 0 means sending packets continuously, other values represent the number of packets to send.

void **esp_phy_test_start_stop**(uint8_t value)

Test start/stop command, used to stop transmitting or reciving state.

参数 **value** –Value should be set to 3 before TX/RX. Set value to 0 to end TX/RX state.

void **esp_phy_wifi_rx**(uint32_t chan, *esp_phy_wifi_rate_t* rate)

Wi-Fi RX command.

参数

- **chan** –channel setting, 1~14;
- **rate** –rate setting;

void **esp_phy_wifi_tx_tone**(uint32_t start, uint32_t chan, uint32_t backoff)

Wi-Fi Carrier Wave(CW) TX command.

参数

- **start** –enable CW, 1 means transmit, 0 means stop transmitting;
- **chan** –CW channel setting, 1~14;
- **backoff** –CW power attenuation parameter, unit is 0.25dB. 4 indicates the power is attenuated by 1dB.

void **esp_phy_ble_tx**(uint32_t txpwr, uint32_t chan, uint32_t len, *esp_phy_ble_type_t* data_type, uint32_t syncw, *esp_phy_ble_rate_t* rate, uint32_t tx_num_in)

BLE TX command.

参数

- **txpwr** –Transmit power level. Tx power is about (level-8)*3 dBm, step is 3dB. Level 8 is around 0 dBm;
- **chan** –channel setting, range is 0~39, corresponding frequency = 2402+chan*2;
- **len** –Payload length setting, range is 0-255, unit is byte, 37 bytes is employed generally;
- **data_type** –Data type setting;
- **syncw** –Packet identification (need to be provided by the packet generator or instrument manufacturer), 0x71764129 is employed generally;
- **rate** –rate setting;
- **tx_num_in** –The number of packets sent, 0 means sending packets continuously, other values represent the number of packets to send.

void **esp_phy_ble_rx**(uint32_t chan, uint32_t syncw, *esp_phy_ble_rate_t* rate)

BLE RX command.

参数

- **chan** –channel selection, range is 0-39; Channels 0, 1, 2~10 correspond to 2404MHz, 2406MHz, 2408MHz~2424MHz respectively; Channels 11, 12, 13~36 correspond to 2428MHz, 2430MHz, 2432MHz~2478MHz respectively; Channel 37: 2402MHz, Channel 38: 2426MHz, Channel 39: 2480MHz;
- **syncw** –Packet identification (need to be provided by the packet generator or instrument manufacturer), 0x71764129 is employed generally;
- **rate** –rate setting;

void **esp_phy_bt_tx_tone**(uint32_t start, uint32_t chan, uint32_t power)

BLE Carrier Wave(CW) TX command.

参数

- **start** –enable CW, 1 means transmit, 0 means stop transmitting;
- **chan** –Single carrier transmission channel selection, range is 0~39, corresponding frequency freq = 2402+chan*2;
- **power** –CW power attenuation parameter, unit is 0.25dB. 4 indicates the power is attenuated by 1dB.

void **esp_phy_get_rx_result**(*esp_phy_rx_result_t* *rx_result)

Get some RX information.

参数 **rx_result** –This struct for storing RX information;

Structures

struct **esp_phy_rx_result_t**

Structure holding PHY RX result.

Public Members

uint32_t **phy_rx_correct_count**

The number of desired packets received

int **phy_rx_rssi**

Average RSSI of desired packets

uint32_t **phy_rx_total_count**

The number of total packets received

uint32_t **phy_rx_result_flag**

0 means no RX info; 1 means the lastest Wi-Fi RX info; 2 means the lastest BLE RX info.

Enumerations

enum **esp_phy_wifi_rate_t**

Values:

enumerator **PHY_RATE_1M**

enumerator **PHY_RATE_2M**

enumerator **PHY_RATE_5M5**

enumerator **PHY_RATE_11M**

enumerator **PHY_RATE_6M**

enumerator **PHY_RATE_9M**

enumerator **PHY_RATE_12M**

enumerator **PHY_RATE_18M**

enumerator **PHY_RATE_24M**

enumerator **PHY_RATE_36M**

enumerator **PHY_RATE_48M**

enumerator **PHY_RATE_54M**

enumerator **PHY_RATE_MCS0**

enumerator **PHY_RATE_MCS1**

enumerator **PHY_RATE_MCS2**

enumerator **PHY_RATE_MCS3**

enumerator **PHY_RATE_MCS4**

enumerator **PHY_RATE_MCS5**

enumerator **PHY_RATE_MCS6**

enumerator **PHY_RATE_MCS7**

enumerator **PHY_WIFI_RATE_MAX**

enum **esp_phy_ble_rate_t**

Values:

enumerator **PHY_BLE_RATE_1M**

enumerator **PHY_BLE_RATE_2M**

enumerator **PHY_BLE_RATE_125K**

enumerator **PHY_BLE_RATE_500k**

enumerator **PHY_BLE_RATE_MAX**

enum **esp_phy_ble_type_t**

Values:

enumerator **PHY_BLE_TYPE_1010**

enumerator **PHY_BLE_TYPE_00001111**

enumerator **PHY_BLE_TYPE_prbs9**

enumerator **PHY_BLE_TYPE_00111100**

enumerator **PHY_BLE_TYPE_MAX**

4.21 Security

This guide provides an overview of the overall security features available in Espressif solutions. It is highly recommended to consider this guide while designing the products with Espressif platform and ESP-IDF software stack from the **"security"** perspective.

4.21.1 Goals

High level security goals are as follows:

1. Preventing untrusted code execution
2. Protecting the identity and integrity of the code stored in the off-chip flash memory
3. Securing device identity

4. Secure storage for confidential data
5. Authenticated and encrypted communication from the device

4.21.2 Platform Security

Secure Boot

Secure Boot feature ensures that only authenticated software can execute on the device. Secure boot process forms chain of trust by verifying all *mutable* software entities involved in the *ESP-IDF boot process*. Signature verification happens during both boot-up as well as OTA updates.

Please refer to the *Secure Boot (v2) Guide* for detailed documentation about this feature.

重要： It is highly recommended that a secure boot feature be enabled on all production devices.

Secure Boot Best Practices

- Generate the signing key on a system with a quality source of entropy.
- Always keep the signing key private. 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.
- Ensure that all security eFuses have been correctly programmed, includes disabling of the debug interfaces, non-required boot mediums (e.g., UART DL mode) etc.

Flash Encryption

Flash Encryption feature helps to encrypt the contents on the off-chip flash memory and thus provides the "*confidentiality*" aspect to the software or data stored in the flash memory.

Please refer to the *Flash Encryption Guide* for detailed documentation about this feature.

Flash Encryption Best Practices

- It is recommended to use Flash Encryption release mode for the production use-cases.
- It is recommended to have a unique flash encryption key per device.
- Enable *Secure Boot* as an extra layer of protection, and to prevent an attacker from selectively corrupting any part of the flash before boot.

Memory Protection

ESP32-C2 supports "Memory Protection" scheme (either through architecture or special peripheral like PMS) which provides an ability to enforce and monitor permission attributes to memory (and peripherals in some cases). ESP-IDF application startup code configures the permissions attributes like Read/Write access on data memories and Read/Execute access on instruction memories using this peripheral. If there is any attempt made that breaks these permission attributes (e.g., a write operation to instruction memory region) then a violation interrupt is raised, and it results in system panic.

This feature depends on the config option *CONFIG_ESP_SYSTEM_MEMPROT_FEATURE* and it is kept default enabled. Please note that the API for this feature is `private` and used exclusively by ESP-IDF code only.

备注： This feature can help to prevent the possibility of remote code injection due to the existing vulnerabilities in the software.

Debug Interfaces

JTAG

- JTAG interfaces stays disabled if any of the security features are enabled, please refer to *JTAG 与 flash 加密和安全引导* for more information.
- JTAG interface can also be disabled in the absence of any other security features using *eFuse API*.

UART DL Mode In ESP32-C2, Secure UART Download mode gets activated if any of the security features are enabled.

- Secure UART Download mode can also be enabled by calling `esp_efuse_enable_rom_secure_download_mode()`.
- This mode does not allow any arbitrary code to execute if downloaded through the UART download mode.
- It also limits the available commands in Download mode 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*).
- To disable Download Mode entirely select the *CONFIG_SECURE_UART_ROM_DL_MODE* to "Permanently disable ROM Download Mode (recommended)" or call `esp_efuse_disable_rom_download_mode()` at runtime.

重要： In Secure UART Download mode, `esptool` can only work with the argument `--no-stub`.

4.21.3 Network Security

Wi-Fi

In addition to the traditional security methods (WEP/WPA-TKIP/WPA2-CCMP), Wi-Fi driver in ESP-IDF also supports additional state-of-the-art security protocols. Please refer to the *Wi-Fi Security* for detailed documentation.

TLS (Transport Layer Security)

It is recommended to use TLS (Transport Layer Security) in all external communications, e.g., cloud communication, OTA updates etc. from the ESP device. ESP-IDF supports *mbedTLS* as the official TLS stack.

TLS is default integrated in *ESP HTTP Client*, *ESP HTTPS Server* and several other components that ship with ESP-IDF.

备注： It is recommended to use ESP-IDF protocol components in their default configuration which has been ensured to be secure. Disabling HTTPS and similar security critical configurations should be avoided.

ESP-TLS Abstraction ESP-IDF provides an abstraction layer for most used TLS functionalities and hence it is recommended that an application makes use of the API exposed by *ESP-TLS*.

TLS Server verification section highlights diverse ways in which the identity of server could be established on the device side.

ESP Certificate Bundle The *ESP x509 Certificate Bundle* API provides an easy way to include a bundle of custom x509 root certificates for TLS server verification. The certificate bundle is the easiest way to verify the identity of almost all standard TLS servers.

重要： It is highly recommended to verify the identity of the server (based on X.509 certificates) to avoid establishing communication with the *fake* server.

4.21.4 Product Security

Secure Provisioning

Secure Provisioning refers to a process of secure on-boarding of the ESP device on to the Wi-Fi network. This mechanism also allows provision of additional custom configuration data during the initial provisioning phase from the provisioning entity (e.g., Smartphone).

ESP-IDF provides various security schemes to establish a secure session between ESP and the provisioning entity, they are highlighted at *Security Schemes*.

Please refer to the *Wi-Fi Provisioning* documentation for details and example code for this feature.

备注： Espressif provides Android and iOS Phone Apps along with their sources so that it could be easy to further customize them as per the product requirement.

Secure OTA (Over-the-air) Updates

- OTA Updates must happen over secure transport, e.g., HTTPS.
- ESP-IDF provides a simplified abstraction layer *ESP HTTPS OTA* for this.
- If *Secure Boot* is enabled then server should host the signed application image.
- If *Flash Encryption* is enabled then no additional steps are required on the server side, encryption shall be taken care on the device itself during flash write.
- OTA update 回滚过程 can help to switch the application as `active` only after its functionality has been verified.

Anti-Rollback Protection Anti-rollback protection feature ensures that device only executes application that meets the security version criteria as stored in its eFuse. So even though the application is trusted and signed by legitimate key it may contain some revoked security feature or credential and hence device must reject any such application.

ESP-IDF allows this feature for the application only and it's managed through 2nd stage bootloader. The security version is stored in the device eFuse and it's compared against the application image header during both bootup and over-the-air updates.

Please see more information to enable this feature in the 防回滚 guide.

Encrypted Firmware Distribution Encrypted firmware distribution during over-the-air updates ensure that the application stays encrypted **in transit** from server to the the device. This can act as an additional layer of protection on top of the TLS communication during OTA updates and protect the identity of the application.

Please see working example for this documented in *OTA Upgrades with Pre-Encrypted Firmware* section.

Secure Storage

Secure storage refers to the application specific data that can be stored in a secure manner on the device (off-chip flash memory). This is typically read-write flash partition and holds device specific configuration data e.g., Wi-Fi credentials.

ESP-IDF provides "NVS (Non-volatile Storage)" management component which allows encrypted data partitions. This feature is tied with the platform *Flash Encryption* feature described earlier.

Please refer to the *NVS Encryption* for detailed documentation on the working and instructions to enable this feature.

重要： By default, ESP-IDF components writes the device specific data into the default NVS partition (includes Wi-Fi credentials too) and it is recommended to protect this data using "NVS Encryption" feature.

Secure Device Control

ESP-IDF provides capability to control an ESP device over `Wi-Fi + HTTP` or `BLE` in a secure manner using ESP Local Control component.

Please refer to the *ESP Local Control* for detailed documentation about this feature.

4.21.5 Security Policy

ESP-IDF GitHub repository has attached Security Policy Brief.

Advisories

- Espressif publishes critical Security Advisories on the website, this includes both hardware and software related.
- ESP-IDF software components specific advisories are published through the GitHub repository.

Software Updates

Critical security issues in the ESP-IDF components, 3rd party libraries are fixed as and when we find them or when they are reported to us. Gradually, we make the fixes available in all applicable release branches in ESP-IDF.

Applicable security issues and CVEs for the ESP-IDF components, 3rd party libraries are mentioned in the ESP-IDF release notes.

重要： We recommend periodically updating to the latest bugfix version of the ESP-IDF release to have all critical security fixes available.

4.22 Secure Boot V2

重要： This document is about Secure Boot V2, supported on ESP32-C2

Secure Boot V2 uses ECDSA based app and bootloader verification. This document can also be used as a reference for signing apps using the ECDSA scheme without signing the bootloader.

4.22.1 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-C2, 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.

A ECC based Secure Boot verification scheme (Secure Boot V2) has been introduce on ESP32-C2

The Secure Boot process on the ESP32-C2 involves the following steps:

1. When the first stage bootloader loads the second stage bootloader, the second stage bootloader’ s 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 ECDSA signature is verified. If the verification is successful, the application image is executed.

4.22.2 Advantages

- The ECDSA public key is stored on the device. The corresponding ECDSA private key is kept at a secret place and is never accessed by the device.
- Only one public key can be generated and stored in the chip during manufacturing.
- 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.)

4.22.3 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.

Only one signature block can be appended to the bootloader or application image in ESP32-C2

Each signature block contains a signature of the preceding image as well as the corresponding ECDSA-256 or ECDSA-192 public key. For more details about the format, refer to *Signature Block Format*. A digest of the 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.

4.22.4 Signature Block Format

The signature block starts on a 4KB 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*).

The content of each signature block is shown in the following table:

表 7: 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.

4.22.5 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
- ESP32-C2 supports configurable flash MMU page size, it (`CONFIG_MMU_PAGE_SIZE`) gets set based on the *CONFIG_ESPTOOLPY_FLASHSIZE*
- 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 8: 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`).

4.22.6 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's invalid.

4.22.7 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 doesn't 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 don't match, the verification fails.
3. Use the public key to verify the signature of the bootloader image, using ECDSA signature verification (section 5.3.3 of RFC6090) with the image digest calculated in step (2) for comparison.

4.22.8 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`.

4.22.9 eFuse usage

- SECURE_BOOT_EN - Enables Secure Boot protection on boot.

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.

4.22.10 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 ECDSA (V2) by default.
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're 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` doesn't flash the bootloader if Secure Boot is enabled.

---

10. Reset the ESP32-C2 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-C2 to verify that Secure Boot is enabled and no errors have occurred due to the build configuration.

备注： Secure boot won't 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-C2 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).

4.22.11 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.

4.22.12 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`.

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 ECDSA key generation utilities.

For example, to generate a signing key using the openssl command line:

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.

4.22.13 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.

4.22.14 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 encryption* to prevent local readout of the flash contents.

4.22.15 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 ECDSA-256 or ECDSA-192 private signing key.

4.22.16 Secure Boot & Flash Encryption

If Secure Boot is used without *Flash Encryption*, 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.

重要： ESP32-C2 has only one eFuse key block, which is used for both keys: Secure Boot and Flash Encryption. The eFuse key block can only be burned once. Therefore these keys should be burned together at the same time. Please note that "Secure Boot" and "Flash Encryption" can not be enabled separately as subsequent writes to eFuse key block shall return an error.

4.22.17 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 isn't verified during the update process, it's 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's 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 isn't 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's necessary to enable the full Secure Boot feature instead.

备注： In general, it's 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. Ensure *App Signing Scheme* is *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.

4.22.18 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.

4.23 Thread Local Storage

4.23.1 Overview

Thread-local storage (TLS) is a mechanism by which variables are allocated such that there is one instance of the variable per extant thread. ESP-IDF provides three ways to make use of such variables:

- *FreeRTOS Native API*: ESP-IDF FreeRTOS native API.
- *Pthread API*: ESP-IDF's pthread API.
- *C11 Standard*: C11 standard introduces special keyword to declare variables as thread local.

4.23.2 FreeRTOS Native API

The ESP-IDF FreeRTOS provides the following API to manage thread local variables:

- `vTaskSetThreadLocalStoragePointer()`
- `pvTaskGetThreadLocalStoragePointer()`
- `vTaskSetThreadLocalStoragePointerAndDelCallback()`

In this case maximum number of variables that can be allocated is limited by *CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS* configuration value. Variables are kept in the task control block (TCB) and accessed by their index. Note that index 0 is reserved for ESP-IDF internal uses.

Using that API user can allocate thread local variables of an arbitrary size and assign them to any number of tasks. Different tasks can have different sets of TLS variables.

If size of the variable is more then 4 bytes then user is responsible for allocating/deallocating memory for it. Variable's deallocation is initiated by FreeRTOS when task is deleted, but user must provide function (callback) to do proper cleanup.

4.23.3 Pthread API

The ESP-IDF provides the following *pthread API* to manage thread local variables:

- `pthread_key_create()`
- `pthread_key_delete()`
- `pthread_getspecific()`
- `pthread_setspecific()`

This API has all benefits of the one above, but eliminates some its limits. The number of variables is limited only by size of available memory on the heap. Due to the dynamic nature this API introduces additional performance overhead compared to the native one.

4.23.4 C11 Standard

The ESP-IDF FreeRTOS supports thread local variables according to C11 standard (ones specified with `__thread` keyword). For details on this GCC feature please see https://gcc.gnu.org/onlinedocs/gcc-5.5.0/gcc/Thread-Local.html#Thread-Local. Storage for that kind of variables is allocated on the task's stack. Note that area for all such variables in the program will be allocated on the stack of every task in the system even if that task does not use such variables at all. For example ESP-IDF system tasks (like `ipc`, `timer` tasks etc.) will also have that extra stack space allocated. So this feature should be used with care. There is a tradeoff: C11 thread local variables are quite handy to use in programming and can be accessed using minimal CPU instructions, but this benefit goes with the cost of additional stack usage for all tasks in the system. Due to static nature of variables allocation all tasks in the system have the same sets of C11 thread local variables.

4.24 工具

4.24.1 IDF Frontend - idf.py

The `idf.py` command-line tool provides a front-end for easily managing your project builds, deployment and debugging, and more. It manages several tools, for example:

- CMake, which configures the project to be built
- Ninja which builds the project
- esptool.py for flashing the target.

The *getting started guide* contains a brief introduction to how to set up `idf.py` to configure, build, and flash projects.

重要： `idf.py` should be run in an ESP-IDF "project" directory, i.e. one containing a `CMakeLists.txt` file. Older style projects with a `Makefile` will not work with `idf.py`.

Commands

Start a new project: create-project

```
idf.py create-project <project name>
```

This will create a new ESP-IDF project, additionally folder where the project will be created can be specified by the `--path` option.

Create a new component: create-component This command creates a new component, which will have a minimum set of files necessary for building.

```
idf.py create-component <component name>
```

The `-C` option can be used to specify the directory the component will be created in. For more information about components see the *build system page*.

Select the Target Chip: set-target ESP-IDF supports multiple targets (chips). A full list of supported targets in your version of ESP-IDF can be seen by running `idf.py --list-targets`.

This sets the current project target:

```
idf.py set-target <target>
```

重要: `idf.py set-target` will clear the build directory and re-generate the `sdkconfig` file from scratch. The old `sdkconfig` file will be saved as `sdkconfig.old`.

备注: The behavior of `idf.py set-target` command is equivalent to:

1. clearing the build directory (`idf.py fullclean`)
2. removing the sdkconfig file (`mv sdkconfig sdkconfig.old`)
3. configuring the project with the new target (`idf.py -DIDF_TARGET=esp32 reconfigure`)

It is also possible to pass the desired `IDF_TARGET` as an environment variable (e.g. `export IDF_TARGET=esp32s2`) or as a CMake variable (e.g. `-DIDF_TARGET=esp32s2` argument to CMake or idf.py). Setting the environment variable is a convenient method if you mostly work with one type of the chip.

To specify the _default_ value of `IDF_TARGET` for a given project, add `CONFIG_IDF_TARGET` value to `sdkconfig.defaults`. For example, `CONFIG_IDF_TARGET="esp32s2"`. This value will be used if `IDF_TARGET` is not specified by other method: using an environment variable, CMake variable, or `idf.py set-target` command.

If the target has not been set by any of these methods, the build system will default to `esp32` target.

Start the graphical configuration tool: menuconfig

```
idf.py menuconfig
```

Build the project: build

```
idf.py build
```

Running this command will build the project found in the current directory. This can involve multiple steps:

- Create the build directory if needed. The sub-directory `build` is used to hold build output, although this can be changed with the `-B` option.
- Run CMake as necessary to configure the project and generate build files for the main build tool.
- Run the main build tool (Ninja or *GNU Make*). By default, the build tool is automatically detected but it can be explicitly set by passing the `-G` option to `idf.py`.

Building is incremental so if no source files or configuration has changed since the last build, nothing will be done.

Additionally, the command can be run with `app`, `bootloader` and `partition-table` arguments to build only the app, bootloader or partition table as applicable.

Remove the build output: clean It is possible to remove the project build output files from the build directory by using:

```
idf.py clean
```

The project will be fully rebuilt on next build. Using this does not remove the CMake configuration output inside the build folder.

Delete the entire build contents: fullclean

```
idf.py fullclean
```

Running this command will delete the entire "build" directory contents. This includes all CMake configuration output. The next time the project is built, CMake will configure it from scratch. Note that this option recursively deletes *all* files in the build directory, so use with care. Project configuration is not deleted.

Flash the project: flash Running the following command:

```
idf.py flash
```

will automatically build the project if necessary, and then flash it to the target. You can use `-p` and `-b` options to set serial port name and flasher baud rate, respectively.

备注: The environment variables `ESPPORT` and `ESPBAUD` can be used to set default values for the `-p` and `-b` options, respectively. Providing these options on the command line overrides the default.

Similarly to the `build` command, the command can be run with `app`, `bootloader` and `partition-table` arguments to flash only the app, bootloader or partition table as applicable.

Hints on how to resolve errors

`idf.py` will try to suggest hints on how to resolve errors. It works with a database of hints stored in tools/idf_py_actions/hints.yml and the hints will be printed if a match is found for the given error. The menuconfig target is not supported at the moment by automatic hints on resolving errors.

The `--no-hints` argument of `idf.py` can be used to turn the hints off in case they are not desired.

Important notes

Multiple `idf.py` commands can be combined into one. For example, `idf.py -p COM4 clean flash monitor` will clean the source tree, then build the project and flash it to the target before running the serial monitor.

The order of multiple `idf.py` commands on the same invocation is not important, they will automatically be executed in the correct order for everything to take effect (i.e. building before flashing, erasing before flashing, etc.).

For commands that are not known to `idf.py` an attempt to execute them as a build system target will be made.

The command `idf.py` supports shell autocompletion for bash, zsh and fish shells.

In order to make shell autocompletion supported, please make sure you have at least Python 3.5 and click 7.1 or newer (*see also*).

To enable autocompletion for `idf.py` use the `export` command (*see this*). Autocompletion is initiated by pressing the TAB key. Type `idf.py -` and press the TAB key to autocomplete options.

The autocomplete support for PowerShell is planned in the future.

Advanced Commands

Open the documentation: docs Using the following command the documentation for the projects target and version will be opened in the browser:

```
idf.py docs
```

Show size: size

```
idf.py size
```

Will print app size information including occupied RAM and FLASH and section sizes.

```
idf.py size-components
```

Similarly, this will print the same information for each component used in the project.

```
idf.py size-files
```

Will print size information per source file in the project.

Options

- `--format` specifies the output format with available options: `text`, `csv`, `json`, default being `text`.
- `--output-file` optionally specifies the name of the file to print the command output to instead of the standard output.

Reconfigure the project: reconfigure

```
idf.py reconfigure
```

This command re-runs CMake even if it doesn't seem to need re-running. This isn't necessary during normal usage, but can be useful after adding/removing files from the source tree, or when modifying CMake cache variables. For example, `idf.py -DNAME='VALUE' reconfigure` can be used to set variable `NAME` in CMake cache to value `VALUE`.

Clean the python byte code: python-clean Generated python byte code can be deleted from the IDF directory using:

```
idf.py python-clean
```

The byte code may cause issues when switching between IDF and Python versions. It is advised to run this target after switching versions of Python.

Global Options

To list all available root level options, run `idf.py --help`. To list options that are specific for a subcommand, run `idf.py <command> --help`, for example `idf.py monitor --help`. Here is a list of some useful options:

- `-C <dir>` allows overriding the project directory from the default current working directory.
- `-B <dir>` allows overriding the build directory from the default `build` subdirectory of the project directory.
- `--ccache` flag can be used to enable CCache when compiling source files, if the CCache tool is installed. This can dramatically reduce some build times.

Note that some older versions of CCache may exhibit bugs on some platforms, so if files are not rebuilt as expected then try disabling CCache and build again. CCache can be enabled by default by setting the `IDF_CCACHE_ENABLE` environment variable to a non-zero value.

- `-v` flag causes both `idf.py` and the build system to produce verbose build output. This can be useful for debugging build problems.
- `--cmake-warn-uninitialized` (or `-w`) will cause CMake to print uninitialized variable warnings found in the project directory only. This only controls CMake variable warnings inside CMake itself, not other types of build warnings. This option can also be set permanently by setting the `IDF_CMAKE_WARN_UNINITIALIZED` environment variable to a non-zero value.
- `--no-hints` flag to disable hints on resolving errors and disable capturing output.

4.24.2 IDF Docker Image

IDF Docker image (`espressif/idf`) is intended for building applications and libraries with specific versions of ESP-IDF, when doing automated builds.

The image contains:

- Common utilities such as git, wget, curl, zip.
- Python 3.7 or newer.
- A copy of a specific version of ESP-IDF (see below for information about versions). `IDF_PATH` environment variable is set, and points to ESP-IDF location in the container.
- All the build tools required for the specific version of ESP-IDF: CMake, ninja, cross-compiler toolchains, etc.
- All Python packages required by ESP-IDF are installed in a virtual environment.

The image entrypoint sets up `PATH` environment variable to point to the correct version of tools, and activates the Python virtual environment. As a result, the environment is ready to use the ESP-IDF build system.

The image can also be used as a base for custom images, if additional utilities are required.

Tags

Multiple tags of this image are maintained:

- `latest`: tracks `master` branch of ESP-IDF
- `vX.Y`: corresponds to ESP-IDF release `vX.Y`
- `release-vX.Y`: tracks `release/vX.Y` branch of ESP-IDF

备注: Versions of ESP-IDF released before this feature was introduced do not have corresponding Docker image versions. You can check the up-to-date list of available tags at https://hub.docker.com/r/espressif/idf/tags.

Usage

Setting up Docker Before using the `espressif/idf` Docker image locally, make sure you have Docker installed. Follow the instructions at https://docs.docker.com/install/, if it is not installed yet.

If using the image in CI environment, consult the documentation of your CI service on how to specify the image used for the build process.

Building a project with CMake In the project directory, run:

```
docker run --rm -v $PWD:/project -w /project espressif/idf idf.py build
```

The above command explained:

- `docker run`: runs a Docker image. It is a shorter form of the command `docker container run`.
- `--rm`: removes the container when the build is finished
- `-v $PWD:/project`: mounts the current directory on the host (`$PWD`) as `/project` directory in the container

- `espressif/idf`: uses Docker image `espressif/idf` with tag `latest` (implicitly added by Docker when no tag is specified)
- `idf.py build`: runs this command inside the container

To build with a specific Docker image tag, specify it as `espressif/idf:TAG`, for example:

```
docker run --rm -v $PWD:/project -w /project espressif/idf:release-v4.4 idf.py␣
↪build
```

You can check the up-to-date list of available tags at https://hub.docker.com/r/espressif/idf/tags.

Using the image interactively It is also possible to do builds interactively, to debug build issues or test the automated build scripts. Start the container with *-i -t* flags:

```
docker run --rm -v $PWD:/project -w /project -it espressif/idf
```

Then inside the container, use `idf.py` as usual:

```
idf.py menuconfig
idf.py build
```

备注: Commands which communicate with the development board, such as `idf.py flash` and `idf.py monitor` will not work in the container unless the serial port is passed through into the container. This can be done with Docker for Linux with the device option. However currently this is not possible with Docker for Windows (https://github.com/docker/for-win/issues/1018) and Docker for Mac (https://github.com/docker/for-mac/issues/900). This limitation may be overcome by using remote serial ports. An example how to do this can be found in the following *using remote serial port* section.

Using remote serial port The RFC2217 (Telnet) protocol can be used to remotely connect to a serial port. For more information please see the remote serial ports documentation in the esptool project. This method can also be used to access the serial port inside a Docker container if it cannot be accessed directly. Following is an example how to use the flash command from within a Docker container.

On host install and start `esp_rfc2217_server`:

- On Windows, package is available as a one-file bundled executable created by pyinstaller and it can be downloaded from the esptool releases page in a zip archive along with other esptool utilities:

  ```
  esp_rfc2217_server -v -p 4000 COM3
  ```

- On Linux/MacOS, package is available as part of *esptool* which can be found in ESP-IDF environment or by installing using pip:

  ```
  pip install esptool
  ```

  And then starting the server by executing:

  ```
  esp_rfc2217_server.py -v -p 4000 /dev/ttyUSB0
  ```

Now the device attached to the host can be flashed from inside a Docker container by using:

```
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
```

Please make sure that `<host_path>` is properly set to your project path on the host and `<container_path>` is set as a working directory inside the container with the `-w` option. The `host.docker.internal` is a special Docker DNS name to access the host. This can be replaced with host IP if necessary.

Building custom images

The Dockerfile in ESP-IDF repository provides several build arguments which can be used to customize the Docker image:

- `IDF_CLONE_URL`: URL of the repository to clone ESP-IDF from. Can be set to a custom URL when working with a fork of ESP-IDF. Default is `https://github.com/espressif/esp-idf.git`.
- `IDF_CLONE_BRANCH_OR_TAG`: Name of a git branch or tag use when cloning ESP-IDF. This value is passed to `git clone` command using the `--branch` argument. Default is `master`.
- `IDF_CHECKOUT_REF`: If this argument is set to a non-empty value, `git checkout $IDF_CHECKOUT_REF` command will be performed after cloning. This argument can be set to the SHA of the specific commit to check out, for example if some specific commit on a release branch is desired.
- `IDF_CLONE_SHALLOW`: If this argument is set to a non-empty value, `--depth=1 --shallow-submodules` arguments will be used when performing `git clone`. This significantly reduces the amount of data downloaded and the size of the resulting Docker image. However, if switching to a different branch in such a "shallow" repository is necessary, an additional `git fetch origin <branch>` command must be executed first.
- `IDF_INSTALL_TARGETS`: Comma-separated list of IDF targets to install toolchains for, or `all` to install toolchains for all targets. Selecting specific targets reduces the amount of data downloaded and the size of the resulting Docker image. Default is `all`.

To use these arguments, pass them via the `--build-arg` command line option. For example, the following command will build a Docker image with a shallow clone of ESP-IDF v4.4.1 and tools for ESP32-C3, only:

```
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
```

4.24.3 IDF Windows Installer

Command-line parameters

Windows Installer *esp-idf-tools-setup* provides the following command-line parameters:

- `/CONFIG=[PATH]` - Path to `ini` configuration file to override default configuration of the installer. Default: `config.ini`.
- `/GITCLEAN=[yes|no]` - Perform git clean and remove untracked directories in Offline mode installation. Default: yes.
- `/GITRECURSIVE=[yes|no]` - Clone recursively all git repository submodules. Default: yes
- `/GITREPO=[URL|PATH]` - URL of repository to clone ESP-IDF. Default: https://github.com/espressif/esp-idf.git
- `/GITRESET=[yes|no]` - Enable/Disable git reset of repository during installation. Default: yes.
- `/HELP` - Display command line options provided by Inno Setup installer.
- `/IDFDIR=[PATH]` - Path to directory where it will be installed. Default: `{userdesktop}\esp-idf}`
- `/IDFVERSION=[v4.3|v4.1|master]` - Use specific IDF version. E.g. v4.1, v4.2, master. Default: empty, pick the first version in the list.
- `/IDFVERSIONSURL=[URL]` - Use URL to download list of IDF versions. Default: https://dl.espressif.com/dl/esp-idf/idf_versions.txt
- `/LOG=[PATH]` - Store installation log file in specific directory. Default: empty.
- `/OFFLINE=[yes|no]` - Execute installation of Python packages by PIP in offline mode. The same result can be achieved by setting the environment variable PIP_NO_INDEX. Default: no.
- `/USEEMBEDDEDPYTHON=[yes|no]` - Use Embedded Python version for the installation. Set to `no` to allow Python selection screen in the installer. Default: yes.
- `/PYTHONNOUSERSITE=[yes|no]` - Set PYTHONNOUSERSITE variable before launching any Python command to avoid loading Python packages from AppDataRoaming. Default: yes.

- `/PYTHONWHEELSURL=[URL]` - Specify URLs to PyPi repositories for resolving binary Python Wheel dependencies. The same result can be achieved by setting the environment variable PIP_EXTRA_INDEX_URL. Default: https://dl.espressif.com/pypi
- `/SKIPSYSTEMCHECK=[yes|no]` - Skip System Check page. Default: no.
- `/VERYSILENT /SUPPRESSMSGBOXES /SP- /NOCANCEL` - Perform silent installation.

Unattended installation

The unattended installation of IDF can be achieved by following command-line parameters:

```
esp-idf-tools-setup-x.x.exe /VERYSILENT /SUPPRESSMSGBOXES /SP- /NOCANCEL
```

The installer detaches its process from the command-line. Waiting for installation to finish could be achieved by following PowerShell script:

```
esp-idf-tools-setup-x.x.exe /VERYSILENT /SUPPRESSMSGBOXES /SP- /NOCANCEL
$InstallerProcess = Get-Process esp-idf-tools-setup
Wait-Process -Id $InstallerProcess.id
```

Custom Python and custom location of Python wheels

The IDF installer is using by default embedded Python with reference to Python Wheel mirror.

Following parameters allows to select custom Python and custom location of Python wheels:

```
esp-idf-tools-setup-x.x.exe /USEEMBEDDEDPYTHON=no /PYTHONWHEELSURL=https://pypi.
↪org/simple/
```

4.24.4 IDF Component Manager

The IDF Component manager is a tool that downloads dependencies for any ESP-IDF CMake project. The download happens automatically during a run of CMake. It can source components either from the component registry or from a git repository.

A list of components can be found on https://components.espressif.com/

Using with a project

Dependencies for each component in the project are defined in a separate manifest file named `idf_component.yml` placed in the root of the component. The manifest file template can be created for a component by running `idf.py create-manifest --component=my_component`. When a new manifest is added to one of the components in the project it's necessary to reconfigure it manually by running `idf.py reconfigure`. Then build will track changes in `idf_component.yml` manifests and automatically triggers CMake when necessary.

There is an example application: example:*build_system/cmake/component_manager* that uses components installed by the component manager.

It's not necessary to have a manifest for components that don't need any managed dependencies.

When CMake configures the project (e.g. `idf.py reconfigure`) component manager does a few things:

- Processes `idf_component.yml` manifests for every component in the project and recursively solves dependencies
- Creates a `dependencies.lock` file in the root of the project with a full list of dependencies
- Downloads all dependencies to the `managed_components` directory

The lock-file `dependencies.lock` and content of `managed_components` directory is not supposed to be modified by a user. When the component manager runs it always make sure they are up to date. If these files were accidentally modified it's possible to re-run the component manager by triggering CMake with `idf.py reconfigure`

You may set build property `DEPENDENCIES_LOCK` to specify the lock-file path in the top-level CMakeLists.txt. For example, adding `idf_build_set_property(DEPENDENCIES_LOCK dependencies.lock.${IDF_TARGET})` before `project(PROJECT_NAME)` could help generate different lock files for different targets.

Defining dependencies in the manifest

```
dependencies:
  # Required IDF version
  idf: ">=4.1"
  # Defining a dependency from the registry:
  # https://components.espressif.com/component/example/cmp
  example/cmp: ">=1.0.0"

  # # Other ways to define dependencies
  #
  # # For components maintained by Espressif only name can be used.
  # # Same as `espressif/cmp`
  # component: "~1.0.0"
  #
  # # Or in a longer form with extra parameters
  # component2:
  #   version: ">=2.0.0"
  #
  #   # For transient dependencies `public` flag can be set.
  #   # `public` flag doesn't affect the `main` component.
  #   # All dependencies of `main` are public by default.
  #   public: true
  #
  #   # For components hosted on non-default registry:
  #   service_url: "https://componentregistry.company.com"
  #
  # # For components in git repository:
  # test_component:
  #   path: test_component
  #   git: ssh://git@gitlab.com/user/components.git
  #
  # # For test projects during component development
  # # components can be used from a local directory
  # # with relative or absolute path
  # some_local_component:
  #   path: ../../projects/component
```

Disabling the Component Manager

The component manager can be explicitly disabled by setting `IDF_COMPONENT_MANAGER` environment variable to `0`.

4.24.5 IDF Clang Tidy

The IDF Clang Tidy is a tool that uses clang-tidy to run static analysis on your current app.

警告： This functionality and the toolchain it relies on are still under development. There may be breaking changes before a final release.

警告： This tool does not support RISC-V based chips yet. For now, we don't provide clang based toolchain for RISC-V.

Prerequisites

If you have never run this tool before, take the following steps to get this tool prepared.

1. Run the export scripts (`export.sh` / `export.bat` / ···) to set up the environment variables.
2. Run `pip install --upgrade pyclang` to install this plugin. The extra commands would be activated in `idf.py` automatically.
3. Run `idf_tools.py install esp-clang` to install the clang-tidy required binaries

   备注： This toolchain is still under development. After the final release, you don't have to install them manually.

4. Get file from the llvm repository and add the folder of this script to the `$PATH`. Or you could pass an optional argument `--run-clang-tidy-py` later when you call `idf.py clang-check`. Please don't forget to make the script executable.

   备注： This file would be bundled in future toolchain releases. This is a temporary workaround.

5. Run the export scripts (`export.sh` / `export.bat` / ···) again to refresh the environment variables.

Extra Commands

`clang-check` Run `idf.py clang-check` to re-generate the compilation database and run `clang-tidy` under your current project folder. The output would be written to `<project_dir>/warnings.txt`.

Run `idf.py clang-check --help` to see the full documentation.

`clang-html-report`

1. Run `pip install codereport` to install the additional dependency.
2. Run `idf.py clang-html-report` to generate an HTML report in folder `<project_dir>/html_report` according to the `warnings.txt`. Please open the `<project_dir>/html_report/index.html` in your browser to check the report.

Bug Report

This tool is hosted in espressif/clang-tidy-runner. If you faced any bugs or have any feature request, please report them via github issues.

4.24.6 Downloadable Tools

ESP-IDF build process relies on a number of tools: cross-compiler toolchains, CMake build system, and others.

Installing the tools using an OS-specific package manager (like apt, yum, brew, etc.) is the preferred method when the required version of the tool is available. This recommendation is reflected in the Getting Started guide. For example, on Linux and macOS it is recommended to install CMake using an OS package manager.

However, some of the tools are IDF-specific and are not available in OS package repositories. Furthermore, different versions of ESP-IDF require different versions of the tools to operate correctly. To solve these two problems, ESP-IDF provides a set of scripts for downloading and installing the correct versions of tools, and exposing them in the environment.

The rest of the document refers to these downloadable tools simply as "tools". Other kinds of tools used in ESP-IDF are:

- Python scripts bundled with ESP-IDF (such as `idf.py`)
- Python packages installed from PyPI.

The following sections explain the installation method, and provide the list of tools installed on each platform.

备注: This document is provided for advanced users who need to customize their installation, users who wish to understand the installation process, and ESP-IDF developers.

If you are looking for instructions on how to install the tools, see the *Getting Started Guide*.

Tools metadata file

The list of tools and tool versions required for each platform is located in tools/tools.json. The schema of this file is defined by tools/tools_schema.json.

This file is used by tools/idf_tools.py script when installing the tools or setting up the environment variables.

Tools installation directory

`IDF_TOOLS_PATH` environment variable specifies the location where the tools are to be downloaded and installed. If not set, `IDF_TOOLS_PATH` defaults to `HOME/.espressif` on Linux and macOS, and `%USER_PROFILE%\.espressif` on Windows.

Inside `IDF_TOOLS_PATH`, the scripts performing tools installation create the following directories and files:

- `dist` —where the archives of the tools are downloaded.
- `tools` —where the tools are extracted. The tools are extracted into subdirectories: `tools/TOOL_NAME/VERSION/`. This arrangement allows different versions of tools to be installed side by side.
- `idf-env.json` —user install options (targets, features) are stored in this file. Targets are selected chip targets for which tools are installed and kept up-to-date. Features determine the Python package set which should be installed. These options will be discussed later.
- `python_env` —not tools related; virtual Python environments are installed in the sub-directories. Note that the Python environment directory can be placed elsewhere by setting the `IDF_PYTHON_ENV_PATH` environment variable.
- `espidf.constraints.*.txt` —one constraint file for each ESP-IDF release containing Python package version requirements.

GitHub Assets Mirror

Most of the tools downloaded by the tools installer are GitHub Release Assets, which are files attached to a software release on GitHub.

If GitHub downloads are inaccessible or slow to access, it's possible to configure a GitHub assets mirror.

To use Espressif's download server, set the environment variable `IDF_GITHUB_ASSETS` to `dl.espressif.com/github_assets`. When the install process is downloading a tool from `github.com`, the URL will be rewritten to use this server instead.

Any mirror server can be used provided the URL matches the `github.com` download URL format: the install process will replace `https://github.com` with `https://${IDF_GITHUB_ASSETS}` for any GitHub asset URL that it downloads.

备注： The Espressif download server doesn't currently mirror everything from GitHub, it only mirrors files attached as Assets to some releases as well as source archives for some releases.

idf_tools.py script

tools/idf_tools.py script bundled with ESP-IDF performs several functions:

- `install`: Download the tool into `${IDF_TOOLS_PATH}/dist` directory, extract it into `${IDF_TOOLS_PATH}/tools/TOOL_NAME/VERSION`.
  `install` command accepts the list of tools to install, in `TOOL_NAME` or `TOOL_NAME@VERSION` format. If `all` is given, all the tools (required and optional ones) are installed. If no argument or `required` is given, only the required tools are installed.
- `download`: Similar to `install` but doesn't extract the tools. An optional `--platform` argument may be used to download the tools for the specific platform.
- `export`: Lists the environment variables which need to be set to use the installed tools. For most of the tools, setting `PATH` environment variable is sufficient, but some tools require extra environment variables.
  The environment variables can be listed in either of `shell` or `key-value` formats, set by `--format` parameter:
  - `export` optional parameters:
    * `--unset` Creates statement that unset some global variables, so the environment gets to the state it was before calling `export.{sh/fish}`.
    * `--add_paths_extras` Adds extra ESP-IDF-related paths of `$PATH` to `${IDF_TOOLS_PATH}/esp-idf.json`, which is used to remove global variables when the active ESP-IDF environment is deactivated. Example: While processing `export.{sh/fish}` script, new paths are added to global variable `$PATH`. This option is used to save these new paths to the `${IDF_TOOLS_PATH}/esp-idf.json`.
  - `shell` produces output suitable for evaluation in the shell. For example,

    ```
    export PATH="/home/user/.espressif/tools/tool/v1.0.0/bin:$PATH"
    ```

    on Linux and macOS, and

    ```
    set "PATH=C:\Users\user\.espressif\tools\v1.0.0\bin;%PATH%"
    ```

    on Windows.

    备注： Exporting environment variables in Powershell format is not supported at the moment. `key-value` format may be used instead.

    The output of this command may be used to update the environment variables, if the shell supports this. For example:

    ```
    eval $($IDF_PATH/tools/idf_tools.py export)
    ```

  - `key-value` produces output in *VARIABLE=VALUE* format, suitable for parsing by other scripts:

    ```
    PATH=/home/user/.espressif/tools/tool/v1.0.0:$PATH
    ```

    Note that the script consuming this output has to perform expansion of `$VAR` or `%VAR%` patterns found in the output.
- `list`: Lists the known versions of the tools, and indicates which ones are installed.
  Following options are available to customize the output.
  - `--outdated`: List only outdated versions of tools installed in `IDF_TOOLS_PATH`.
- `check`: For each tool, checks whether the tool is available in the system path and in `IDF_TOOLS_PATH`.
- `install-python-env`: Create a Python virtual environment in the `${IDF_TOOLS_PATH}/python_env` directory (or directly in the directory set by the `IDF_PYTHON_ENV_PATH` environment variable) and install there the required Python packages. An optional `--features` argument allows one to specify a comma-separated list of features to be added or removed. Feature that begins with `-` will be removed and features with `+` or without any sign will be added. Example syntax for removing feature `XY`

is `--features=-XY` and for adding `--features=+XY` or `--features=XY`. If both removing and adding options are provided with the same feature, no operation is performed. For each feature a requirements file must exist. For example, feature `XY` is a valid feature if `${IDF_PATH}/tools/requirements/requirements.XY.txt` is an existing file with a list of Python packages to be installed. There is one mandatory `core` feature ensuring core functionality of ESP-IDF (build, flash, monitor, debug in console). There can be an arbitrary number of optional features. The selected list of features is stored in `idf-env.json`. The requirement files contain a list of the desired Python packages to be installed and `espidf.constraints.*.txt` downloaded from https://dl.espressif.com and stored in `${IDF_TOOLS_PATH}` the package version requirements for a given ESP-IDF version. Althought it is not recommended, the download and use of constraint files can be disabled with the `--no-constraints` argument or setting the `IDF_PYTHON_CHECK_CONSTRAINTS` environment variable to `no`.

- `check-python-dependencies`: Checks if all required Python packages are installed. Packages from `${IDF_PATH}/tools/requirements/requirements.*.txt` files selected by the feature list of `idf-env.json` are checked with the package versions specified in the `espidf.constraints.*.txt` file. The constraint file is downloaded with `install-python-env` command. The use of constraints files can be disabled similarly to the `install-python-env` command.
- `uninstall`: Print and remove tools, that are currently not used by active ESP-IDF version.
  - `--dry-run` Print installed unused tools.
  - `--remove-archives` Additionally remove all older versions of previously downloaded installation packages.

Install scripts

Shell-specific user-facing scripts are provided in the root of ESP-IDF repository to facilitate tools installation. These are:

- `install.bat` for Windows Command Prompt
- `install.ps1` for Powershell
- `install.sh` for Bash
- `install.fish` for Fish

Aside from downloading and installing the tools into `IDF_TOOLS_PATH`, these scripts prepare a Python virtual environment, and install the required packages into that environment.

These scripts accept optionally a comma separated list of chip targets and `--enable-*` arguments for enabling features. These arguments are passed to the `idf_tools.py` script which stores them in `idf-env.json`. Therefore, chip targets and features can be enabled incrementally.

Running the scripts without any optional arguments will install tools for all chip targets (by running `idf_tools.py install --targets=all`) and Python packages for core ESP-IDF functionality (by running `idf_tools.py install-python-env --features=core`).

Or for example, `install.sh esp32` will install tools only for ESP32. See the *Getting Started Guide* for more examples.

`install.sh --enable-XY` will enable feature `XY` (by running `idf_tools.py install-python-env --features=core,XY`).

Export scripts

Since the installed tools are not permanently added into the user or system `PATH` environment variable, an extra step is required to use them in the command line. The following scripts modify the environment variables in the current shell to make the correct versions of the tools available:

- `export.bat` for Windows Command Prompt
- `export.ps1` for Powershell
- `export.sh` for Bash
- `export.fish` for Fish

备注： To modify the shell environment in Bash, `export.sh` must be "sourced" : `. ./export.sh` (note the leading dot and space).

`export.sh` may be used with shells other than Bash (such as zsh). However in this case the `IDF_PATH` environment variable must be set before running the script. When used in Bash, the script will guess the `IDF_PATH` value from its own location.

In addition to calling `idf_tools.py`, these scripts list the directories which have been added to the `PATH`.

Other installation methods

Depending on the environment, more user-friendly wrappers for `idf_tools.py` are provided:

- *IDF Tools installer for Windows* can download and install the tools. Internally the installer uses `idf_tools.py`.
- Eclipse Plugin includes a menu item to set up the tools. Internally the plugin calls `idf_tools.py`.
- VSCode Extension for ESP-IDF includes an onboarding flow. This flow helps setting up the tools. Although the extension does not rely on `idf_tools.py`, the same installation method is used.

Custom installation

Although the methods above are recommended for ESP-IDF users, they are not a must for building ESP-IDF applications. ESP-IDF build system expects that all the necessary tools are installed somewhere, and made available in the `PATH`.

List of IDF Tools

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-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-x86_64-linux-gnu.tar.gz SHA256: d056f2435ef05cccadac5d8fcefa3efd8f8c456c3d853f5eba1edb501acfe4f7
linux-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-aarch64-linux-gnu.tar.gz SHA256: 7fc9674cc4f4c5e7bc94ca05bc5deaaa4c4bbcc972a9caee6fcd6a872c804c02
linux-armel	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-arm-linux-gnueabi.tar.gz SHA256: 68118ff36e9dd2284d92a7a529d0e2a8d20f6426036a0736fa1147935614ece2
linux-armhf	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-arm-linux-gnueabihf.tar.gz SHA256: 72d75d9bb9a09d0696aa86628b2dd1851755216b1b315743189ea37228e5c72f
linux-i686	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-i586-linux-gnu.tar.gz SHA256: cf6cac8ed70726d390d30713d537754544872715e1b70a8a4a28b5dc616193b9
macos	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-x86_64-apple-darwin14.tar.gz SHA256: 417fcf8d1b596b9481603d6987def1d6cfcebdb9739f53940887334a7de855fa
macos-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-aarch64-apple-darwin21.1.tar.gz SHA256: 95d6ed2311d6a72bf349e152d096aeeb151f9c5989bfa3120facb1c99e879196
win32	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-i686-w64-mingw32.zip SHA256: 642b6a135c38ff1d5e54ad2c29469b769f8e1b101dab363d06101b02284bb979
win64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xtensa-esp-elf-gdb-12.1_20221002-x86_64-w64-mingw32.zip SHA256: 2d958570ff6aa69ed32cbb076cbaf303349a26b3301a7c4628be8d7ad39cf9f1

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-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-linux-gnu.tar.gz SHA256: f0cf0821eaac7e8cf2c63b14f2b69d612f4f8c266b29d02d5547b7d7cbbd0e11
linux-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-aarch64-linux-gnu.tar.gz SHA256: 6812344dfb5c50a81d2fd8354463516f0aa5f582e8ab406cbaeca8722b45fa94
linux-armel	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-arm-linux-gnueabi.tar.gz SHA256: b73042b8e1df5a3fc8008ec3cd000ef579f155d72a66c6ade1d48906d843e738
linux-armhf	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-arm-linux-gnueabihf.tar.gz SHA256: 9dc5334042a169606d32ee454d35d93d216a24df027e4b0830ab268a8999db2d
linux-i686	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-i586-linux-gnu.tar.gz SHA256: 3f07a1b8dc87127a1a6bec6fbace4f8daca44755356f0692e9a5d4c8c4bfd81d
macos	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-apple-darwin14.tar.gz SHA256: bb139229f9a4998cab9cfb617d3ecb05b77cbfa9a3a59c54969035f1b4007487
macos-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-aarch64-apple-darwin21.1.tar.gz SHA256: f6513b57f28245497f9c39a201f3f6444d4180e16b39765c629e01036286c0e6
win32	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-i686-w64-mingw32.zip SHA256: 8287fa2891e8d032e8283210048d653705791cda31504369418288d3e4753dd6
win64	required	https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-w64-mingw32.zip SHA256: 9debae1135df8f5868a9d945468f0480cdaab25f77ead6a55cc85142c4487abd

xtensa-esp32-elf Toolchain for Xtensa (ESP32) 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-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-x86_64-linux-gnu.tar.xz SHA256: 4d2e02ef47f1a93a4dcfdbaecd486adfaab4c0e26deea2c18d6385527f39f864
linux-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-aarch64-linux-gnu.tar.xz SHA256: 9e211a182b6ea0396a41c78f52f51d964e7875fe274ea9c81111bf0dbc90c516
linux-armel	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-arm-linux-gnueabi.tar.xz SHA256: 2ddd91fb98b79b30042b7918eef60cf10c7bd5b1da853e83b65f293b96dec800
linux-armhf	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-arm-linux-gnueabihf.tar.xz SHA256: a683a468555dcbcb6ce32a190842110d6f853d4d6104d61cf0bc9dd50c6be1e6
linux-i686	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-i686-linux-gnu.tar.xz SHA256: 292b19ea6186508a923fb6fd0103977e001d4eb8e77836c7e3d6ce6e5fa7d305
macos	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-x86_64-apple-darwin.tar.xz SHA256: b09d87fdb1dc32cd1d718935065ef931b101a14df6b17be56748e52640955bff
macos-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-aarch64-apple-darwin.tar.xz SHA256: f50acab2b216e9475dc5313b3e4b424cbc70d0abd23ba1818aff4a019165da8e
win32	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-i686-w64-mingw32.zip SHA256: 62bb6428d107ed3f44c212c77ecf24804b74c97327b0f0ad2029c656c6dbd6ee
win64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32-elf-12.2.0_20230208-x86_64-w64-mingw32.zip SHA256: 8febfe4a6476efc69012390106c8c660a14418f025137b0513670c72124339cf

xtensa-esp32s2-elf Toolchain for Xtensa (ESP32-S2) 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-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-x86_64-linux-gnu.tar.xz SHA256: a1bd8f0252aae02cff2c289f742fbdbaa2c24644cc30e883d118253ea4df1799
linux-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-aarch64-linux-gnu.tar.xz SHA256: 48e88053e92bab1bf8d6dbad7ddb4d140c537159d607a36e73e74e1f5f23c892
linux-armel	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-arm-linux-gnueabi.tar.xz SHA256: 37cdd619fa56ce884570cedd00dd2f4a5eb9a1fce3755a2f4b9279d1136e47c1
linux-armhf	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-arm-linux-gnueabihf.tar.xz SHA256: 99a7b34e8826d0c0b5703e5a4e7db8716b9738fa4f03eed759f383a10617e788
linux-i686	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-i686-linux-gnu.tar.xz SHA256: d9b79e9e3204fa8e40f9942ea1197a83ae1527e3711a45bc17171ff5fec43e54
macos	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-x86_64-apple-darwin.tar.xz SHA256: e7b2fbacd8186b24d1b1264ad6cf639f476d51f5d908fb79504abfe6281d3c8c
macos-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-aarch64-apple-darwin.tar.xz SHA256: d2c997ce5f43a93c3787c224aa8742b0cd87443794514ab2153cd629665506f0
win32	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-i686-w64-mingw32.zip SHA256: 1e6dac5162ab75f94b88c47ebeabb6600c652fb4f615ed07c1724d037c02fd19
win64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s2-elf-12.2.0_20230208-x86_64-w64-mingw32.zip SHA256: 8a785cc4e0838cebe404f82c0ead7a0f9ac5fabc660a742e33a41ddac6326cc1

xtensa-esp32s3-elf Toolchain for Xtensa (ESP32-S3) 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-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-x86_64-linux-gnu.tar.xz SHA256: 29b5ea6b30d98231f0c17f2327404109e0abf59b48d0f2890d9d9899678a89a3
linux-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-aarch64-linux-gnu.tar.xz SHA256: 30a1fed3ab6341feb1ae986ee55f227df6a594293ced13c65a0136eb4681087d
linux-armel	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-arm-linux-gnueabi.tar.xz SHA256: c180836bf43b90b4b7c24166a3bd4156c74c8e58bb85761aa58da98d076e6f48
linux-armhf	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-arm-linux-gnueabihf.tar.xz SHA256: 4cc1adee141de67ffb7e94e53d30bf4e120ef07d4063fecc2153c69ad4b54f7f
linux-i686	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-i686-linux-gnu.tar.xz SHA256: 9a968f58085c66b41ca13af8d652e5250df0f8d8e17988e34846be9c76672cab
macos	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-x86_64-apple-darwin.tar.xz SHA256: 30375231847a9070e4e0acb3102b7d35a60448a55536bfa113c677c449da3eef
macos-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-aarch64-apple-darwin.tar.xz SHA256: ae9a1a3e12c0b6f6f28a3878f5964e91a410350248586c90db94f8bdaeef7695
win32	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-i686-w64-mingw32.zip SHA256: 3ddf51774817e815e5d41c312a90c1159226978fb45fd0d4f7085c567f8b73ab
win64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/xtensa-esp32s3-elf-12.2.0_20230208-x86_64-w64-mingw32.zip SHA256: 1d15ca65e3508388a86d8bed3048c46d07538f5bc88d3e4296f9c03152087cd1

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-15.0.0-20221201/llvm-esp-15.0.0-20221201-linux-amd64.tar.xz SHA256: 839e5adfa7f44982e8a2d828680f6e4aa435dcd3d1df765e02f015b04286056f
linux-arm64	optional	https://github.com/espressif/llvm-project/releases/download/esp-15.0.0-20221201/llvm-esp-15.0.0-20221201-linux-arm64.tar.xz SHA256: 614c44ab7305d65dde54a884c56145167770380 27dc61bcc125d02171c248c53
linux-armhf	optional	https://github.com/espressif/llvm-project/releases/download/esp-15.0.0-20221201/llvm-esp-15.0.0-20221201-linux-armhf.tar.xz SHA256: 158076696e4fc608e6e2b54bf739223b78949e0492ad4aa5119632ebfbea0499
macos	optional	https://github.com/espressif/llvm-project/releases/download/esp-15.0.0-20221201/llvm-esp-15.0.0-20221201-macos.tar.xz SHA256: 46f0f0368b5aa8d7e81558796c3acd67d943c9071b9619f2b487136c8e59c97c
macos-arm64	optional	https://github.com/espressif/llvm-project/releases/download/esp-15.0.0-20221201/llvm-esp-15.0.0-20221201-macos-arm64.tar.xz SHA256: dc5a99186f9f532a5076d6900828310e4673cf01e8071a3d041456e8aab2cc4a
win64	optional	https://github.com/espressif/llvm-project/releases/download/esp-15.0.0-20221201/llvm-esp-15.0.0-20221201-win64.tar.xz SHA256: 87c9b2c2b8837535f102ae3fd5789defecbffa80b317f86055f3e9d6292aaa05

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-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-x86_64-linux-gnu.tar.xz SHA256: 21694e5ee506f5e52908b12c6b5be7044d87cf34bb4dfcd151d0a10ea09dedc1
linux-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-aarch64-linux-gnu.tar.xz SHA256: aefbf1e6f2c91a10e8995399d2003502e167e8c95e77f40957309e843700906a
linux-armel	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-arm-linux-gnueabi.tar.xz SHA256: 9740cbddb4cb5e05382991c83d8c96a5fb7d87046449e77791b3b0de29a3ddd8
linux-armhf	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-arm-linux-gnueabihf.tar.xz SHA256: ee6210b1068802ed8486543c1f313cb8ac64571c20d51bf50fdb34ad4c457018
linux-i686	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-i686-linux-gnu.tar.xz SHA256: 9207fe3d1413cf29fad6dc4bdc9a35f538b0b2c48a70e9a89d2f0e930c346aed
macos	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-x86_64-apple-darwin.tar.xz SHA256: 78cd1afe458fceb7c2657fe346edb0ecfde3b8743ccf7a7a7509c456cad9de9a
macos-arm64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-aarch64-apple-darwin.tar.xz SHA256: 6c0a4151afb258766911fc7bcfe5f4fee6ee2cd9a5ff25542bc1228c1203a3f9
win32	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-i686-w64-mingw32.zip SHA256: a5dfbb6dbf6fc6c6ea9beb2723af059ba3c5b2c86c2f0dc3b21afdc7bb229bf5
win64	required	https://github.com/espressif/crosstool-NG/releases/download/esp-12.2.0_20230208/riscv32-esp-elf-12.2.0_20230208-x86_64-w64-mingw32.zip SHA256: 9deae9e0013b2f7bbf017f9c8135755bfa89522f337c7dca35872bf12ec08176

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-v2.35_20220830/esp32ulp-elf-2.35_20220830-linux-amd64.tar.gz SHA256: b1f7801c3a16162e72393ebb772c0cbfe4d22d907be7c2c2dac168736e9195fd
linux-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-2.35_20220830-linux-arm64.tar.gz SHA256: d6671b31bab31b9b13aea25bb7d60f15484cb8bf961ddbf67a62867e5563eae5
linux-armel	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-2.35_20220830-linux-armel.tar.gz SHA256: e107e7a9cd50d630b034f435a16a52db5a57388dc639a99c4c393c5e429711e9
linux-armhf	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-2.35_20220830-linux-armhf.tar.gz SHA256: 6c6dd25477b2e758d4669da3774bf664d1f012442c880f17dfdf0339e9c3dae9
macos	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-2.35_20220830-macos.tar.gz SHA256: 5a952087b621ced16af1e375feac1371a61cb51ab7e7b44cbefb5afda2d573de
macos-arm64	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-2.35_20220830-macos-arm64.tar.gz SHA256: 73bda8476ef92d4f4abee96519abbba40e5ee32f368427469447b83cc7bb9b42
win32	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-2.35_20220830-win32.zip SHA256: 77344715ea7d7a7a9fd0b27653f880efaf3bcc1ac843f61492d8a0365d91f731
win64	required	https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-2.35_20220830-win64.zip SHA256: 525e5b4c8299869a3fdddb51baad76612c5c104bd96952ae6460ad7e5b5a4e21

cmake CMake build system

On Linux and macOS, it is recommended to install CMake using the OS package manager. 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/v3.24.0/cmake-3.24.0-linux-x86_64.tar.gz SHA256: 726f88e6598523911e4bce9b059dc20b851aa77f97e4cc5573f4e42775a5c16f
linux-arm64	optional	https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-linux-aarch64.tar.gz SHA256: 50c3b8e9d3a3cde850dd1ea143df9d1ae546cbc5e74dc6d223eefc1979189651
linux-armel	optional	https://dl.espressif.com/dl/cmake/cmake-3.24.0-Linux-armv7l.tar.gz SHA256: 7dc787ef968dfef92491a4f191b8739ff70f8a649608b811c7a737b52481beb0
linux-armhf	optional	https://dl.espressif.com/dl/cmake/cmake-3.24.0-Linux-armv7l.tar.gz SHA256: 7dc787ef968dfef92491a4f191b8739ff70f8a649608b811c7a737b52481beb0
macos	optional	https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-macos-universal.tar.gz SHA256: 3e0cca74a56d9027dabb845a5a26e42ef8e8b33beb1655d6a724187a345145e4
macos-arm64	optional	https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-macos-universal.tar.gz SHA256: 3e0cca74a56d9027dabb845a5a26e42ef8e8b33beb1655d6a724187a345145e4
win32	required	https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-windows-x86_64.zip SHA256: b1ad8c2dbf0778e3efcc9fd61cd4a962e5c1af40aabdebee3d5074bcff2e103c
win64	required	https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-windows-x86_64.zip SHA256: b1ad8c2dbf0778e3efcc9fd61cd4a962e5c1af40aabdebee3d5074bcff2e103c

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-esp32-20230419/openocd-esp32-linux-amd64-0.12.0-esp32-20230419.tar.gz SHA256: 5144e7516cd75a2152b35ecae0a400f7d3d4424c2488fbacc49433564f54c70d
linux-arm64	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20230419/openocd-esp32-linux-arm64-0.12.0-esp32-20230419.tar.gz SHA256: 1c4d900c738fe00730c6033abb6cf1cc6587717dbeee291d5908272d153d329a
linux-armel	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20230419/openocd-esp32-linux-armel-0.12.0-esp32-20230419.tar.gz SHA256: 293258fd67618dd352e1096137ad9f2b801926eaf74ffcd570540ae94ad8ee5c
linux-armhf	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20230419/openocd-esp32-linux-armhf-0.12.0-esp32-20230419.tar.gz SHA256: b87cfb291476fc2e34468ea9175a9e195c6f1fce88e643c955c87ccc58bfb1f8
macos	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20230419/openocd-esp32-macos-0.12.0-esp32-20230419.tar.gz SHA256: 621aad7d011c6817cde9570dfea42c7bcc699458bf43c37706cb4c2f6475a247
macos-arm64	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20230419/openocd-esp32-macos-arm64-0.12.0-esp32-20230419.tar.gz SHA256: 3af7eac3a7de3939731ec4c13fb5d72a8e6ce5e5d274bb9697f5d93039561e42
win32	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20230419/openocd-esp32-win32-0.12.0-esp32-20230419.zip SHA256: f2cb3d9cacfe789c20d3272af846d726a062ce8f2e4ee142bddb27501d7dd7a7
win64	required	https://github.com/espressif/openocd-esp32/releases/download/v0.12.0-esp32-20230419/openocd-esp32-win32-0.12.0-esp32-20230419.zip SHA256: f2cb3d9cacfe789c20d3272af846d726a062ce8f2e4ee142bddb27501d7dd7a7

ninja Ninja build system

On Linux and macOS, it is recommended to install ninja using the OS package manager. 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://dl.espressif.com/dl/ninja-1.10.2-linux64.tar.gz SHA256: 32bb769de4d57aa7ee0e292cfcb7553e7cc8ea0961f7aa2b3aee60aa407c4033
macos	optional	https://dl.espressif.com/dl/ninja-1.10.2-osx.tar.gz SHA256: 847bb1ca4bc16d8dba6aeed3ecb5055498b86bc68c364c37583eb5738bb440f1
macos-arm64	optional	https://dl.espressif.com/dl/ninja-1.10.2-osx.tar.gz SHA256: 847bb1ca4bc16d8dba6aeed3ecb5055498b86bc68c364c37583eb5738bb440f1
win64	required	https://dl.espressif.com/dl/ninja-1.10.2-win64.zip SHA256: bbde850d247d2737c5764c927d1071cbb1f1957dcabda4a130fa8547c12c695f

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/v4.8/ccache-4.8-windows-x86_64.zip SHA256: a2b3bab4bb8318ffc5b3e4074dc25636258bc7e4b51261f7d9bef8127fda8309

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/20230320/esp-rom-elfs-20230320.tar.gz SHA256: 24bcc8cb3287175d4a0bfd65e04bf7ef592a10f022acffca0d5e87eee05996d4

4.25 ESP32-C2 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 esp32c2 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 IDF 组件支持 Mock，具体请参考*基于 Linux 主机的单元测试*。

4.25.1 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。

4.25.2 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，您需要在 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。

4.25.3 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", "[esp32c2]", 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。

4.25.4 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，让您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()` 包装您的测试代码，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)
```

如果您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。如果您已经进行上述操作，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. 对于某些您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。如果您将一些代码放在 `test/esp32` 目录下来避免 esp32s2 芯片执行它，一旦您需要在新的芯片（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，您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。

4.25.5 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。您只需要运行 `idf.py flash` 即可烧写所有编译输出的文件。

您还可以运行 `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。

4.25.6 运行单元测试

烧写完成后重启 ESP32-C2，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's 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]
(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]
```

(下页继续)

(续上页)

```
(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" [esp32c2][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"
```

您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。

4.25.7 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。

4.25.8 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-C2 上运行，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-C2 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）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”地 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：

- Installed IDF including all IDF requirements
- System package requirements (`libbsd`, `libbsd-dev`)
- A recent enough Linux or MacOS version and gcc compiler
- All components the application depends on must be either supported on the Linux target (Linux/POSIX simulator) or mock-able

An application that runs on the Linux target has to set the `COMPONENTS` variable to `main` in the CMakeLists.txt of the application’s root directory:

```
set(COMPONENTS main)
```

This prevents the automatic inclusion of all components from IDF to the build process which is otherwise done for convenience.

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*Component Linux/Mock Support Overview*。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 IDF 构建系统函数 `idf_component_mock` 指定。您可以使用 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，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 IDF 目录的 tools/mocks 下找到组件模拟的示例。有关如何 *覆盖 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。如果您使用的是 IDF 提供的组件模拟，则第二个方法更加方便。

您可参考 `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。

4.26 Running Applications on Host

备注： Running IDF applications on host is currently still an experimental feature, thus there is no guarantee for API stability. However, user feedback via the ESP-IDF GitHub repository or the ESP32 forum is highly welcome, and may help influence the future of design of the IDF host-based applications.

This document provides an overview of the methods to run IDF applications on Linux, and what type of IDF applications can typically be run on Linux.

4.26.1 Introduction

Typically, an IDF application is built (cross-compiled) on a host machine, uploaded (i.e., flashed) to an ESP chip for execution, and monitored by the host machine via a UART/USB port. However, execution of an IDF application on an ESP chip (hence forth referred to as "running on target") can be limiting in various development/usage/testing scenarios.

Therefore, it is possible for an IDF application to be built and executed entirely within the same Linux host machine (hence forth referred to as "running on host"). Running ESP-IDF applications on host has several advantages:

- No need to upload to a target.
- Faster execution on a host machine, compared to running on an ESP chip.
- No requirements for any specific hardware, except the host machine itself.
- Easier automation and setup for software testing.
- Large number of tools for code and runtime analysis (e.g. Valgrind).

A large number of IDF components depend on chip-specific hardware. These hardware dependencies must be mocked or simulated when running on host. ESP-IDF currently supports the following mocking and simulation approaches:

1. Using the FreeRTOS POSIX/Linux simulator that simulates FreeRTOS scheduling. On top of this simulation, other APIs are also simulated or implemented when running on host.
2. Using CMock to mock all dependencies and run the code in complete isolation.

In principle, it is possible to mix both approaches (POSIX/Linux simulator and mocking using CMock), but this has not been done yet in ESP-IDF. Note that despite the name, the FreeRTOS POSIX/Linux simulator currently also works on MacOS. Running IDF applications on host machines is often used for testing. However, simulating the environment and mocking dependencies does not fully represent the target device. Thus, testing on the target device is still necessary, though with a different focus that usually puts more weight on integration and system testing.

备注： Another possibility to run applications on the host is to use the QEMU simulator. However, QEMU development for IDF applications is currently work in progress and has not been documented yet.

CMock-Based Approach

This approach uses the CMock framework to solve the problem of missing hardware and software dependencies. CMock-based applications running on the host machine have the added advantage that they usually only compile the

necessary code, i.e., the (mostly mocked) dependencies instead of the entire system. For a general introduction to Mocks and how to configure and use them in ESP-IDF, please refer to *Mocks*.

POSIX/Linux Simulator Approach

The FreeRTOS POSIX/Linux simulator is available on ESP-IDF as a preview target already. It is the base for the Linux target which is already available as a preview. Using this simulator, IDF components can be implemented on the host to make them available to IDF applications when running on host. Currently, only a limited number of components are ready to be built on Linux. Furthermore the functionality of each component ported to Linux may also be limited or different compared to the functionality when building that component for a chip target. For more information if the desired components are supported on Linux, please refer to *Component Linux/Mock Support Overview*.

4.26.2 Requirements

- Installed IDF including all IDF requirements
- System package requirements (`libbsd`, `libbsd-dev`)
- A recent enough Linux or MacOS version and gcc compiler
- All components the application depends on must be either supported on the Linux target (Linux/POSIX simulator) or mock-able

An application that runs on the Linux target has to set the `COMPONENTS` variable to `main` in the CMakeLists.txt of the application's root directory:

```
set(COMPONENTS main)
```

This prevents the automatic inclusion of all components from IDF to the build process which is otherwise done for convenience.

If any mocks are used, then `Ruby` is required, too.

4.26.3 Build and Run

To build the application on Linux, the target has to be set to `linux` and then it can be built and run:

```
idf.py --preview set-target linux
idf.py build
idf.py monitor
```

4.26.4 Component Linux/Mock Support Overview

Note that any "Yes" here does not necessarily mean a full implementation or mocking. It can also mean a partial implementation or mocking of functionality. Usually, the implementation or mocking is done to a point where enough functionality is provided to build and run a test application.

Component	Mock	Simulation
driver	Yes	No
esp_common	No	Yes
esp_event	Yes	Yes
esp_hw_support	Yes	Yes
esp_partition	Yes	No
esp_rom	No	Yes
esp_system	No	Yes
esp_timer	Yes	No
esp_tls	Yes	No
freertos	Yes	Yes
hal	No	Yes
heap	No	Yes
http_parser	Yes	No
log	No	Yes
lwip	Yes	No
soc	No	Yes
spi_flash	Yes	No
tcp_transport	Yes	No

4.27 Wi-Fi 驱动程序

4.27.1 ESP32-C2 Wi-Fi 功能列表

ESP32-C2 支持以下 Wi-Fi 功能：

- 支持 3 个虚拟接口，即 STA、AP 和 Sniffer。
- 支持仅 station 模式、仅 AP 模式、station/AP 共存模式
- 支持使用 IEEE 802.11b、IEEE 802.11g、IEEE 802.11n 和 API 配置协议模式
- 支持 WPA/WPA2/WPA3/WPA2-企业版/WPA3-企业版/WPS 和 DPP
- 支持 AMPDU、QoS 以及其它主要功能
- 支持 Modem-sleep
- 空中数据传输最高可达 20 MBit/s TCP 吞吐量和 30 MBit/s UDP 吞吐量
- 支持 Sniffer
- 支持快速扫描和全信道扫描
- 支持多个天线

4.27.2 如何编写 Wi-Fi 应用程序

准备工作

一般来说，要编写自己的 Wi-Fi 应用程序，最高效的方式是先选择一个相似的应用程序示例，然后将其中可用的部分移植到自己的项目中。如果您希望编写一个强健的 Wi-Fi 应用程序，强烈建议您在开始之前先阅读本文。**非强制要求，请依个人情况而定**。

本文将补充说明 Wi-Fi API 和 Wi-Fi 示例的相关信息，重点描述使用 Wi-Fi API 的原则、当前 Wi-Fi API 实现的限制以及使用 Wi-Fi 时的常见错误。同时，本文还介绍了 Wi-Fi 驱动程序的一些设计细节。建议您选择一个示例 example 进行参考。

设置 Wi-Fi 编译时选项

请参阅*Wi-Fi menuconfig*。

Wi-Fi 初始化

请参阅*ESP32-C2 Wi-Fi station 一般情况*、*ESP32-C2 Wi-Fi AP 一般情况*。

启动/连接 Wi-Fi

请参阅*ESP32-C2 Wi-Fi station 一般情况*、*ESP32-C2 Wi-Fi AP 一般情况*。

事件处理

通常，在理想环境下编写代码难度并不大，如*WIFI_EVENT_STA_START*、*WIFI_EVENT_STA_CONNECTED*中所述。难度在于如何在现实的困难环境下编写代码，如*WIFI_EVENT_STA_DISCONNECTED*中所述。能否在后者情况下完美地解决各类事件冲突，是编写一个强健的 Wi-Fi 应用程序的根本。请参阅*ESP32-C2 Wi-Fi 事件描述*, *ESP32-C2 Wi-Fi station 一般情况*, *ESP32-C2 Wi-Fi AP 一般情况*。另可参阅 ESP-IDF 中的事件处理概述。

编写错误恢复程序

除了在能在比较差的环境下工作，错误恢复能力也对一个强健的 Wi-Fi 应用程序至关重要。请参阅*ESP32-C2 Wi-Fi API 错误代码*。

4.27.3 ESP32-C2 Wi-Fi API 错误代码

所有 ESP32-C2 Wi-Fi API 都有定义好的返回值，即错误代码。这些错误代码可分类为：

- 无错误，例如：返回值 ESP_OK 代表 API 成功返回
- 可恢复错误，例如：ESP_ERR_NO_MEM
- 不可恢复的非关键性错误
- 不可恢复的关键性错误

一个错误是否为关键性取决于其 API 和应用场景，并且由 API 用户定义。

要使用 Wi-Fi API 编写一个强健的应用程序，根本原则便是要时刻检查错误代码并编写相应的错误处理代码。一般来说，错误处理代码可用于解决：

- 可恢复错误，您可以编写一个可恢复错误处理代码解决该类错误。例如，当`esp_wifi_start()`返回 ESP_ERR_NO_MEM 时，调用可恢复错误处理代码 vTaskDelay 可以获取几微秒的重试时间。
- 不可恢复非关键性错误，打印错误代码可以帮助您更好地处理该类错误。
- 不可恢复关键性错误，可使用“assert”语句处理该类错误。例如，如果`esp_wifi_set_mode()`返回 ESP_ERR_WIFI_NOT_INIT，该值意为`esp_wifi_init()`未成功初始化 Wi-Fi 驱动程序。您可以在应用程序开发阶段非常快速地检测到此类错误。

在 esp_err.h 中，ESP_ERROR_CHECK 负责检查返回值。这是一个较为常见的错误处理代码，可在应用程序开发阶段作为默认的错误处理代码。但是，我们强烈建议 API 的使用者编写自己的错误处理代码。

4.27.4 初始化 ESP32-C2 Wi-Fi API 参数

初始化 API 的结构参数时，应遵循以下两种方式之一：

- 设置该参数的所有字段
- 先使用 get API 获取当前配置，然后只设置特定于应用程序的字段

初始化或获取整个结构这一步至关重要，因为大多数情况下，返回值 0 意味着程序使用了默认值。未来，我们将会在该结构中加入更多字段，并将这些字段初始化为 0，确保即使 IDF 版本升级后您的应用程序依然能够正常运行。

4.27.5 ESP32-C2 Wi-Fi 编程模型

ESP32-C2 Wi-Fi 编程模型如下图所示：

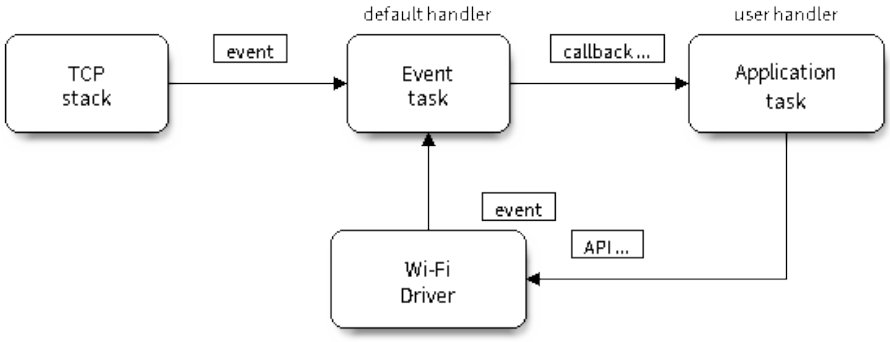


图 26: Wi-Fi 编程模型

Wi-Fi 驱动程序可以看作是一个无法感知上层代码（如 TCP/IP 堆栈、应用程序任务、事件任务等）的黑匣子。通常，应用程序任务（代码）负责调用 *Wi-Fi 驱动程序 APIs* 来初始化 Wi-Fi，并在必要时处理 Wi-Fi 事件。然后，Wi-Fi 驱动程序接收并处理 API 数据，并在应用程序中插入事件。

Wi-Fi 事件处理是在 *esp_event 库* 的基础上进行的。Wi-Fi 驱动程序将事件发送至默认事件循环，应用程序便可以使用 `esp_event_handler_register()` 中的回调函数处理这些事件。除此之外，*esp_netif 组件* 也负责处理 Wi-Fi 事件，并产生一系列默认行为。例如，当 Wi-Fi station 连接至一个 AP 时，esp_netif 将自动开启 DHCP 客户端服务（系统默认）。

4.27.6 ESP32-C2 Wi-Fi 事件描述

WIFI_EVENT_WIFI_READY

Wi-Fi 驱动程序永远不会生成此事件，因此，应用程序的事件回调函数可忽略此事件。在未来的版本中，此事件可能会被移除。

WIFI_EVENT_SCAN_DONE

扫描完成事件，由 `esp_wifi_scan_start()` 函数触发，将在以下情况下产生：

- 扫描已完成，例如：Wi-Fi 已成功找到目标 AP 或已扫描所有信道。
- 当前扫描因函数 `esp_wifi_scan_stop()` 而终止。
- 在当前扫描完成之前调用了函数 `esp_wifi_scan_start()`。此时，新的扫描将覆盖当前扫描过程，并生成一个扫描完成事件。

以下情况下将不会产生扫描完成事件：

- 当前扫描被阻止。
- 当前扫描是由函数 `esp_wifi_connect()` 触发的。

接收到此事件后，事件任务暂不做任何响应。首先，应用程序的事件回调函数需调用*esp_wifi_scan_get_ap_num()* 和*esp_wifi_scan_get_ap_records()* 获取已扫描的 AP 列表，然后触发 Wi-Fi 驱动程序释放在扫描过程中占用的内存空间（**切记该步骤**）。更多详细信息，请参阅*ESP32-C2 Wi-Fi 扫描*。

WIFI_EVENT_STA_START

如果调用函数*esp_wifi_start()* 后接收到返回值 ESP_OK，且当前 Wi-Fi 处于 station 或 station/AP 共存模式，则将产生此事件。接收到此事件后，事件任务将初始化 LwIP 网络接口 (netif)。通常，应用程序的事件回调函数需调用*esp_wifi_connect()* 来连接已配置的 AP。

WIFI_EVENT_STA_STOP

如果调用函数*esp_wifi_stop()* 后接收到返回值 ESP_OK，且当前 Wi-Fi 处于 station 或 station/AP 共存模式，则将产生此事件。接收到此事件后，事件任务将进行释放 station IP 地址、终止 DHCP 客户端服务、移除 TCP/UDP 相关连接并清除 LwIP station netif 等动作。此时，应用程序的事件回调函数通常不需做任何响应。

WIFI_EVENT_STA_CONNECTED

如果调用函数*esp_wifi_connect()* 后接收到返回值 ESP_OK，且 station 已成功连接目标 AP，则将产生此连接事件。接收到此事件后，事件任务将启动 DHCP 客户端服务并开始获取 IP 地址。此时，Wi-Fi 驱动程序已准备就绪，可发送和接收数据。如果您的应用程序不依赖于 LwIP（即 IP 地址），则此刻便可以开始应用程序开发工作。但是，如果您的应用程序需基于 LwIP 进行，则还需等待 *got ip* 事件发生后才可开始。

WIFI_EVENT_STA_DISCONNECTED

此事件将在以下情况下产生：

- 调用了函数*esp_wifi_disconnect()* 或*esp_wifi_stop()*，且 Wi-Fi station 已成功连接至 AP。
- 调用了函数*esp_wifi_connect()*，但 Wi-Fi 驱动程序因为某些原因未能成功连接至 AP，例如：未扫描到目标 AP、验证超时等。或存在多个 SSID 相同的 AP，station 无法连接所有已找到的 AP，也将产生该事件。
- Wi-Fi 连接因为某些原因而中断，例如：station 连续多次丢失 N beacon、AP 踢掉 station、AP 认证模式改变等。

接收到此事件后，事件任务的默认动作为：

- 关闭 station 的 LwIP netif。
- 通知 LwIP 任务清除导致所有套接字状态错误的 UDP/TCP 连接。针对基于套接字编写的应用程序，其回调函数可以在接收到此事件时（如有必要）关闭并重新创建所有套接字。

应用程序处理此事件最常用的方法为：调用函数*esp_wifi_connect()* 重新连接 Wi-Fi。但是，如果此事件是由函数*esp_wifi_disconnect()* 引发的，则应用程序不应调用*esp_wifi_connect()* 来重新连接。应用程序须明确区分此事件的引发原因，因为某些情况下应使用其它更好的方式进行重新连接。请参阅*Wi-Fi 重新连接* 和*连接 Wi-Fi 时扫描*。

需要注意的另一点是：接收到此事件后，LwIP 的默认动作是终止所有 TCP 套接字连接。大多数情况下，该动作不会造成影响。但对某些特殊应用程序可能除外。例如：

- 应用程序创建一个了 TCP 连接，以维护每 60 秒发送一次的应用程序级、保持活动状态的数据。
- 由于某些原因，Wi-Fi 连接被切断并引发了 *WIFI_EVENT_STA_DISCONNECTED* 事件。根据当前实现，此时所有 TCP 连接都将被移除，且保持活动的套接字将处于错误的状态中。但是，由于应用程序设计者认为网络层 **不应**考虑这个 Wi-Fi 层的错误，因此应用程序不会关闭套接字。
- 5 秒后，因为在应用程序的事件回调函数中调用了*esp_wifi_connect()*，Wi-Fi 连接恢复。**同时，station 连接至同一个 AP 并获得与之前相同的 IPV4 地址**。

- 60 秒后，当应用程序发送具有保持活动状态的套接字的数据时，套接字将返回错误，应用程序将关闭套接字并在必要时重新创建。

在上述场景中，理想状态下应用程序套接字和网络层将不会受到影响，因为在此过程中 Wi-Fi 连接只是短暂地断开然后快速恢复。应用程序可通过 LwIP menuconfig 启动“IP 改变时保持 TCP 连接”的功能。

IP_EVENT_STA_GOT_IP

当 DHCP 客户端成功从 DHCP 服务器获取 IPV4 地址或 IPV4 地址发生改变时，将引发此事件。此事件意味着应用程序一切就绪，可以开始任务（如：创建套接字）。

IPV4 地址可能由于以下原因而发生改变：

- DHCP 客户端无法重新获取/绑定 IPV4 地址，且 station 的 IPV4 重置为 0。
- DHCP 客户端重新绑定了其它地址。
- 静态配置的 IPV4 地址已发生改变。

函数 `ip_event_got_ip_t` 中的字段 `ip_change` 说明了 IPV4 地址是否发生改变。

套接字的状态是基于 IPV4 地址的，这意味着，如果 IPV4 地址发生改变，则所有与此 IPV4 相关的套接字都将变为异常。接收到此事件后，应用程序需关闭所有套接字，并在 IPV4 变为有效地址时重新创建应用程序。

IP_EVENT_GOT_IP6

当 IPV6 SLAAC 支持自动为 ESP32-C2 配置一个地址，或 ESP32-C2 地址发生改变时，将引发此事件。此事件意味着应用程序一切就绪，可以开始任务（如：创建套接字）。

IP_EVENT_STA_LOST_IP

当 IPV4 地址失效时，将引发此事件。

此事件不会在 Wi-Fi 断连后立刻出现。Wi-Fi 连接断开后，首先将启动一个 IPV4 地址丢失计时器，如果 station 在该计时器超时之前成功获取了 IPV4 地址，则不会发生此事件。否则，此事件将在计时器超时时发生。

一般来说，应用程序可忽略此事件。这只是一个调试事件，主要使应用程序获知 IPV4 地址已丢失。

WIFI_EVENT_AP_START

与 *WIFI_EVENT_STA_START* 事件相似。

WIFI_EVENT_AP_STOP

与 *WIFI_EVENT_STA_STOP* 事件相似。

WIFI_EVENT_AP_STACONNECTED

每当有一个 station 成功连接 ESP32-C2 AP 时，将引发此事件。接收到此事件后，事件任务将不做任何响应，应用程序的回调函数也可忽略这一事件。但是，您可以在此时进行一些操作，例如：获取已连接 station 的信息等。

WIFI_EVENT_AP_STADISCONNECTED

此事件将在以下情况下发生：

- 应用程序通过调用函数*esp_wifi_disconnect()* 或*esp_wifi_deauth_sta()* 手动断开 station 连接。
- Wi-Fi 驱动程序出于某些原因断开 station 连接，例如：AP 在过去 5 分钟（可通过函数*esp_wifi_set_inactive_time()* 修改该时间）内未接收到任何数据包等。
- station 断开与 AP 之间的连接。

发生此事件时，事件任务将不做任何响应，但应用程序的事件回调函数需执行一些操作，例如：关闭与此 station 相关的套接字等。

WIFI_EVENT_AP_PROBEREQRECVED

默认情况下，此事件处于禁用状态，应用程序可以通过调用 API *esp_wifi_set_event_mask()* 启用。启用后，每当 AP 接收到 probe request 时都将引发此事件。

WIFI_EVENT_STA_BEACON_TIMEOUT

如果 station 在 inactive 时间内未收到所连接 AP 的 beacon，将发生 beacon 超时，将引发此事件。inactive 时间通过调用函数*esp_wifi_set_inactive_time()* 设置。

WIFI_EVENT_CONNECTIONLESS_MODULE_WAKE_INTERVAL_START

非连接模块在 *Interval* 开始时触发此事件。请参考非连接模块功耗管理 。

4.27.7 ESP32-C2 Wi-Fi station 一般情况

下图为 station 模式下的宏观场景，其中包含不同阶段的具体描述：

1. Wi-Fi/LwIP 初始化阶段

- s1.1：主任务通过调用函数*esp_netif_init()* 创建一个 LwIP 核心任务，并初始化 LwIP 相关工作。
- s1.2：主任务通过调用函数*esp_event_loop_create()* 创建一个系统事件任务，并初始化应用程序事件的回调函数。在此情况下，该回调函数唯一的动作就是将事件中继到应用程序任务中。
- s1.3：主任务通过调用函数*esp_netif_create_default_wifi_ap()* 或*esp_netif_create_default_wifi_sta()* 创建有 TCP/IP 堆栈的默认网络接口实例绑定 station 或 AP。
- s1.4：主任务通过调用函数*esp_wifi_init()* 创建 Wi-Fi 驱动程序任务，并初始化 Wi-Fi 驱动程序。
- s1.5：主任务通过调用 OS API 创建应用程序任务。

推荐按照 s1.1 ~ s1.5 的步骤顺序针对基于 Wi-Fi/LwIP 的应用程序进行初始化。但这一顺序 **并非**强制，您可以在第 s1.1 步创建应用程序任务，然后在该应用程序任务中进行所有其它初始化操作。不过，如果您的应用程序任务依赖套接字，那么在初始化阶段创建应用程序任务可能并不适用。此时，您可以在接收到 IP 后再进行任务创建。

2. Wi-Fi 配置阶段

Wi-Fi 驱动程序初始化成功后，可以进入到配置阶段。该场景下，Wi-Fi 驱动程序处于 station 模式。因此，首先您需调用函数*esp_wifi_set_mode()* (WIFI_MODE_STA) 将 Wi-Fi 模式配置为 station 模式。可通过调用其它 esp_wifi_set_xxx API 进行更多设置，例如：协议模式、国家代码、带宽等。请参阅*ESP32-C2 Wi-Fi 配置*。

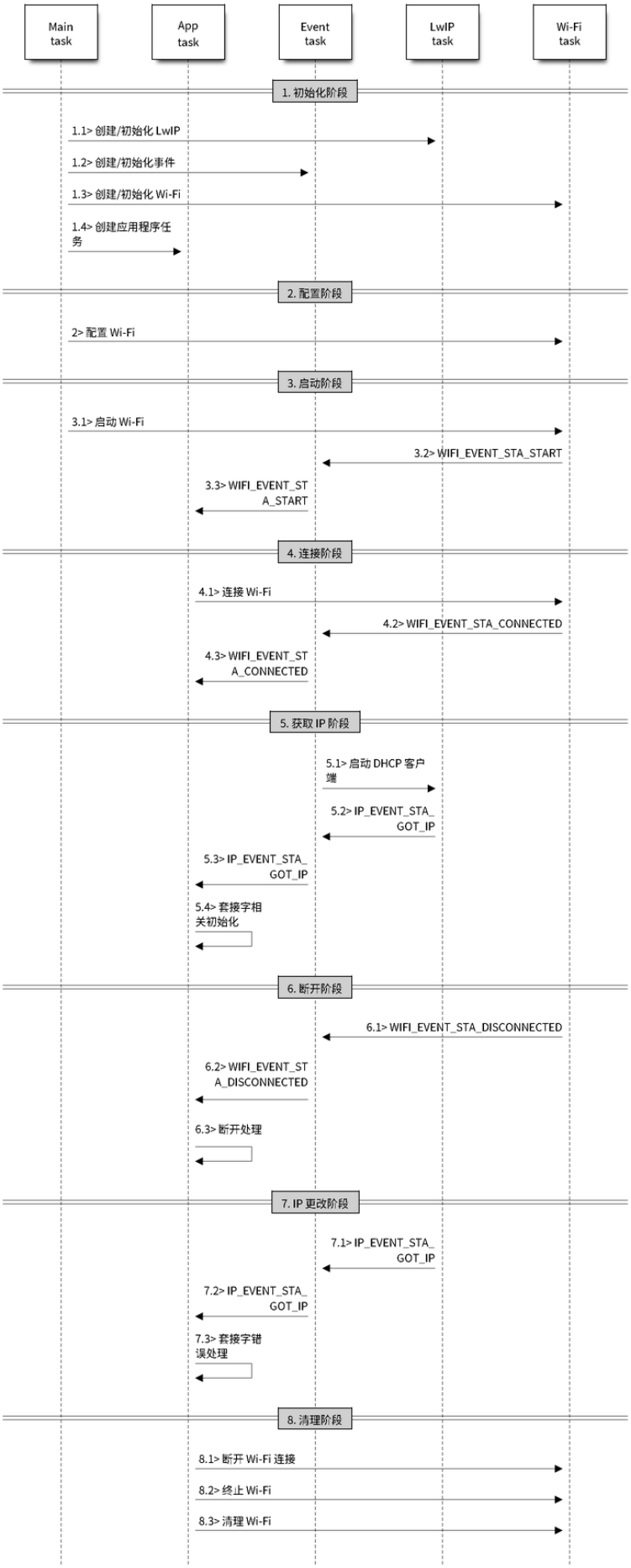


图 27: station 模式下 Wi-Fi 事件场景示例

一般情况下，我们会在建立 Wi-Fi 连接之前配置 Wi-Fi 驱动程序，但这 **并非**强制要求。也就是说，只要 Wi-Fi 驱动程序已成功初始化，您可以在任意阶段进行配置。但是，如果您的 Wi-Fi 在建立连接后不需要更改配置，则应先在此阶段完成配置。因为调用配置 API（例如`esp_wifi_set_protocol()`）将会导致 Wi-Fi 连接断开，为您的操作带来不便。

如果 menuconfig 已使能 Wi-Fi NVS flash，则不论当前阶段还是后续的 Wi-Fi 配置信息都将被存储至该 flash 中。那么，当主板上电/重新启动时，就不需从头开始配置 Wi-Fi 驱动程序。您只需调用函数 esp_wifi_get_xxx API 获取之前存储的配置信息。当然，如果不想使用之前的配置，您依然可以重新配置 Wi-Fi 驱动程序。

3. Wi-Fi 启动阶段

- s3.1：调用函数`esp_wifi_start()`启动 Wi-Fi 驱动程序。
- s3.2：Wi-Fi 驱动程序将事件*WIFI_EVENT_STA_START* 发布到事件任务中，然后，事件任务将执行一些正常操作并调用应用程序的事件回调函数。
- s3.3：应用程序的事件回调函数将事件*WIFI_EVENT_STA_START* 中继到应用程序任务中。推荐您此时调用函数`esp_wifi_connect()`进行 Wi-Fi 连接。当然，您也可以等待在*WIFI_EVENT_STA_START* 事件发生后的其它阶段再调用此函数。

4. Wi-Fi 连接阶段

- s4.1：调用函数`esp_wifi_connect()`后，Wi-Fi 驱动程序将启动内部扫描/连接过程。
- s4.2：如果内部扫描/连接过程成功，将产生*WIFI_EVENT_STA_CONNECTED* 事件。然后，事件任务将启动 DHCP 客户端服务，最终触发 DHCP 程序。
- s4.3：在此情况下，应用程序的事件回调函数会将*WIFI_EVENT_STA_CONNECTED* 事件中继到应用程序任务中。通常，应用程序不需进行操作，而您可以执行任何动作，例如：打印日志等。

步骤 s4.2 中 Wi-Fi 连接可能会由于某些原因而失败，例如：密码错误、未找到 AP 等。这种情况下，将引发*WIFI_EVENT_STA_DISCONNECTED* 事件并提示连接错误原因。有关如何处理中断 Wi-Fi 连接的事件，请参阅下文阶段 6 的描述。

5. Wi-Fi 获取 IP 阶段

- s5.1：一旦步骤 4.2 中的 DHCP 客户端初始化完成，Wi-Fi 驱动程序将进入 *获取 IP* 阶段。
- s5.2：如果 Wi-Fi 成功从 DHCP 服务器接收到 IP 地址，则将引发*IP_EVENT_STA_GOT_IP* 事件，事件任务将执行正常处理。
- s5.3：应用程序的事件回调函数将事件*IP_EVENT_STA_GOT_IP* 中继到应用程序任务中。对于那些基于 LwIP 构建的应用程序，此事件较为特殊，因为它意味着应用程序已准备就绪，可以开始任务，例如：创建 TCP/UDP 套接字等。此时较为容易犯的一个错误就是在接收到*IP_EVENT_STA_GOT_IP* 事件之前就初始化套接字。**切忌在接收到 IP 之前启动任何套接字相关操作**。

6. Wi-Fi 断开阶段

- s6.1：当 Wi-Fi 因为某些原因（例如：AP 掉电、RSSI 较弱等）连接中断时，将产生*WIFI_EVENT_STA_DISCONNECTED* 事件。此事件也可能在上文阶段 3 中发生。在这里，事件任务将通知 LwIP 任务清除/移除所有 UDP/TCP 连接。然后，所有应用程序套接字都将处于错误状态。也就是说，*WIFI_EVENT_STA_DISCONNECTED* 事件发生时，任何套接字都无法正常工作。
- s6.2：上述情况下，应用程序的事件回调函数会将*WIFI_EVENT_STA_DISCONNECTED* 事件中继到应用程序任务中。推荐您调用函数`esp_wifi_connect()`重新连接 Wi-Fi，关闭所有套接字，并在必要时重新创建套接字。请参阅*WIFI_EVENT_STA_DISCONNECTED*。

7. Wi-Fi IP 更改阶段

- s7.1：如果 IP 地址发生更改，将引发*IP_EVENT_STA_GOT_IP* 事件，其中 “ip_change” 被置为 “true”。

- s7.2：**此事件对应用程序至关重要。这一事件发生时，适合关闭所有已创建的套接字并进行重新创建。**

8. Wi-Fi 清理阶段

- s8.1：调用函数*esp_wifi_disconnect()* 断开 Wi-Fi 连接。
- s8.2：调用函数*esp_wifi_stop()* 终止 Wi-Fi 驱动程序。
- s8.3：调用函数*esp_wifi_deinit()* 清理 Wi-Fi 驱动程序。

4.27.8 ESP32-C2 Wi-Fi AP 一般情况

下图为 AP 模式下的宏观场景，其中包含不同阶段的具体描述：

4.27.9 ESP32-C2 Wi-Fi 扫描

目前，仅 station 或 station/AP 共存模式支持*esp_wifi_scan_start()* API。

扫描类型

模式	描述
主动扫描	通过发送 probe request 进行扫描。该模式为默认的扫描模式。
被动扫描	不发送 probe request。跳至某一特定信道并等待 beacon。应用程序可通过 wifi_scan_config_t 中的 scan_type 字段使能被动扫描。
前端扫描	在 station 模式下 Wi-Fi 未连接时，可进行前端扫描。Wi-Fi 驱动程序决定进行前端扫描还是后端扫描，应用程序无法配置这两种模式。
后端扫描	在 station 模式或 station/AP 共存模式下 Wi-Fi 已连接时，可进行后端扫描。Wi-Fi 驱动程序决定进行前端扫描还是后端扫描，应用程序无法配置这两种模式。
全信道扫描	扫描所有信道。wifi_scan_config_t 中的 channel 字段为 0 时，当前模式为全信道扫描。
特定信道扫描	仅扫描特定的信道。wifi_scan_config_t 中的 channel 字段为 1-14 时，当前模式为特定信道扫描。

上表中的扫描模式可以任意组合，因此共有 8 种不同扫描方式：

- 全信道后端主动扫描
- 全信道后端被动扫描
- 全信道前端主动扫描
- 全信道后端被动扫描
- 特定信道后端主动扫描
- 特定信道后端被动扫描
- 特定信道前端主动扫描
- 特定信道前端被动扫描

扫描配置

扫描类型与其他扫描属性通过函数*esp_wifi_scan_start()* 进行配置。下表详细描述了函数 wifi_scan_config_t 各字段信息。

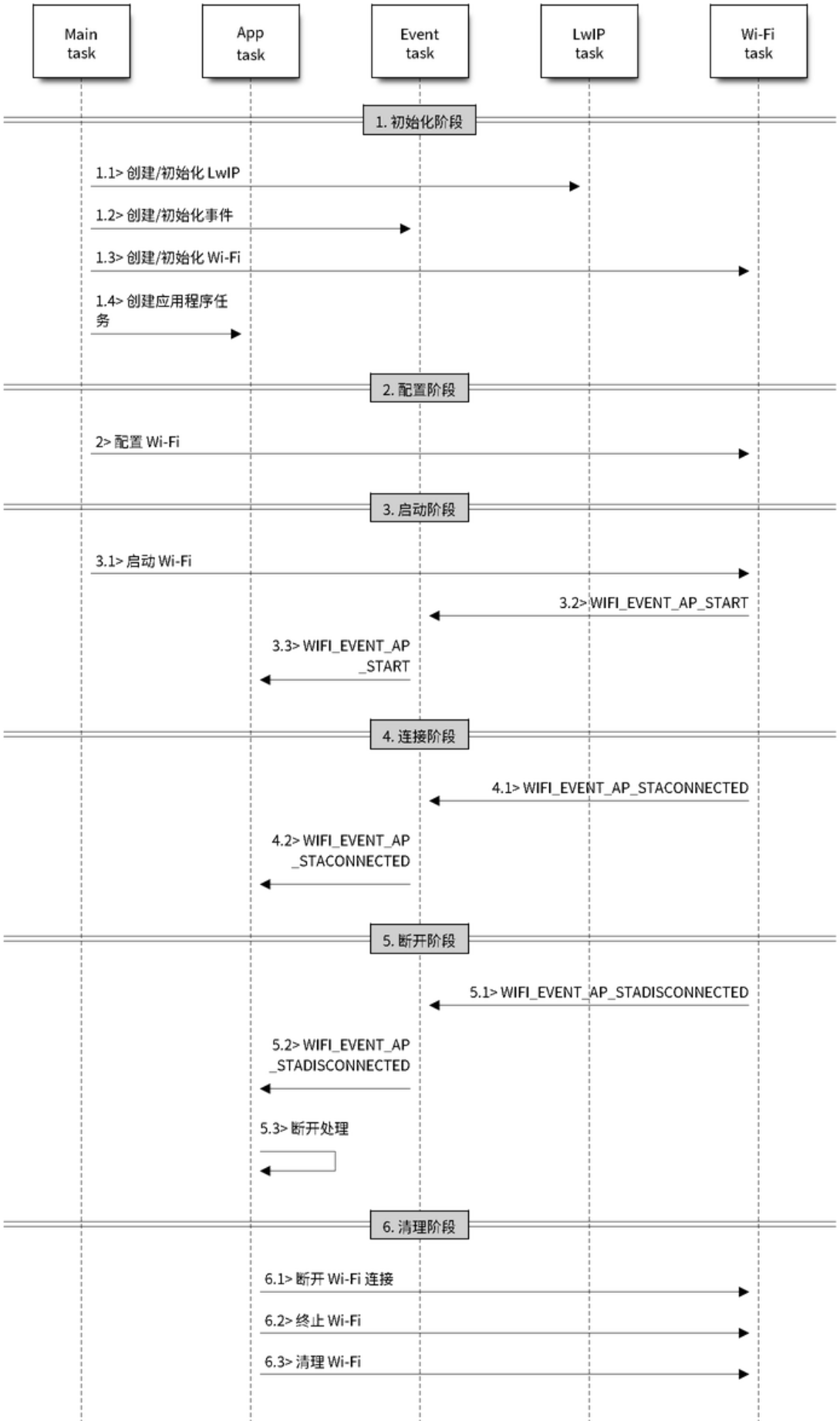


图 28: AP 模式下 Wi-Fi 事件场景示例

字段	描述
ssid	如果该字段的值不为 NULL，则仅可扫描到具有相同 SSID 值的 AP。
bssid	如果该字段的值不为 NULL，则仅可扫描到具有相同 BSSID 值的 AP。
channel	如果该字段值为 0，将进行全信道扫描；反之，将针对特定信道进行扫描。
show_hidden	如果该字段值为 0，本次扫描将忽略具有隐藏 SSID 的 AP；反之，这些 AP 也会在扫描时被视为正常 AP。
scan_type	如果该字段值为为 WIFI_SCAN_TYPE_ACTIVE，则本次扫描为主动扫描；反之，将被视为被动扫描。
scan_time	该字段用于控制每个信道的扫描时间。 被动扫描时，scan_time.passive 字段负责为每个信道指定扫描时间。 主动扫描时，每个信道的扫描时间如下列表所示。其中，min 代表 scan_time_active_min，max 代表 scan_time_active_max。 • min=0, max=0：每个信道的扫描时间为 120 ms。 • min>0, max=0：每个信道的扫描时间为 120 ms。 • min=0, max>0：每个信道的扫描时间为 `max` ms。 • min>0, max>0：每个信道扫描的最短时间为 `min` ms。如果在这段时间内未找到 AP，将跳转至下一个信道。如这段时间内找到 AP，则该信道的扫描时间为 `max` ms。 如希望提升 Wi-Fi 扫描性能，则可修改上述两个参数。

调用 API *esp_wifi_set_config()* 可全局配置一些扫描属性，请参阅*station 基本配置*。

在所有信道中扫描全部 AP（前端）

场景：

上述场景中描述了全信道前端扫描过程。仅 station 模式支持前端扫描，该模式下 station 未连接任何 AP。前端扫描还是后端扫描完全由 Wi-Fi 驱动程序决定，应用程序无法配置这一模式。

详细描述：

扫描配置阶段

- s1.1：如果默认的国家信息有误，调用函数*esp_wifi_set_country()* 进行配置。请参阅*Wi-Fi 国家/地区代码*。
- s1.2：调用函数*esp_wifi_scan_start()* 配置扫描信息，可参阅*扫描配置*。该场景为全信道扫描，将 SSID/BSSID/channel 设置为 0 即可。

Wi-Fi 驱动程序内部扫描阶段

- s2.1：Wi-Fi 驱动程序切换至信道 1，此时的扫描类型为 WIFI_SCAN_TYPE_ACTIVE，同时发送一个 probe request。反之，Wi-Fi 将等待接收 AP beacon。Wi-Fi 驱动程序将在信道 1 停留一段时间。min/max 扫描时间中定义了 Wi-Fi 在信道 1 中停留的时间长短，默认为 120 ms。
- s2.2：Wi-Fi 驱动程序跳转至信道 2，并重复进行 s2.1 中的步骤。
- s2.3:Wi-Fi 驱动程序扫描最后的信道 N，N 的具体数值由步骤 s1.1 中配置的国家代码决定。

扫描完成后事件处理阶段

- s3.1：当所有信道扫描全部完成后，将产生*WIFI_EVENT_SCAN_DONE* 事件。
- s3.2：应用程序的事件回调函数告知应用程序任务已接收到*WIFI_EVENT_SCAN_DONE* 事件。调用函数*esp_wifi_scan_get_ap_num()* 获取在本次扫描中找到的 AP 数量。然后，分配出足够的事物槽，并调用函数*esp_wifi_scan_get_ap_records()* 获取 AP 记录。请注意，一旦调用*esp_wifi_scan_get_ap_records()*，Wi-Fi 驱动程序中的 AP 记录将被释放。但是，请不要在单个扫描完成事件中重复调用两次*esp_wifi_scan_get_ap_records()*。反之，如果扫描完成事件发生后未调用*esp_wifi_scan_get_ap_records()*，则 Wi-Fi 驱动程序中的 AP 记录不会被释放。因此，请务必确保调用函数*esp_wifi_scan_get_ap_records()*，且仅调用一次。

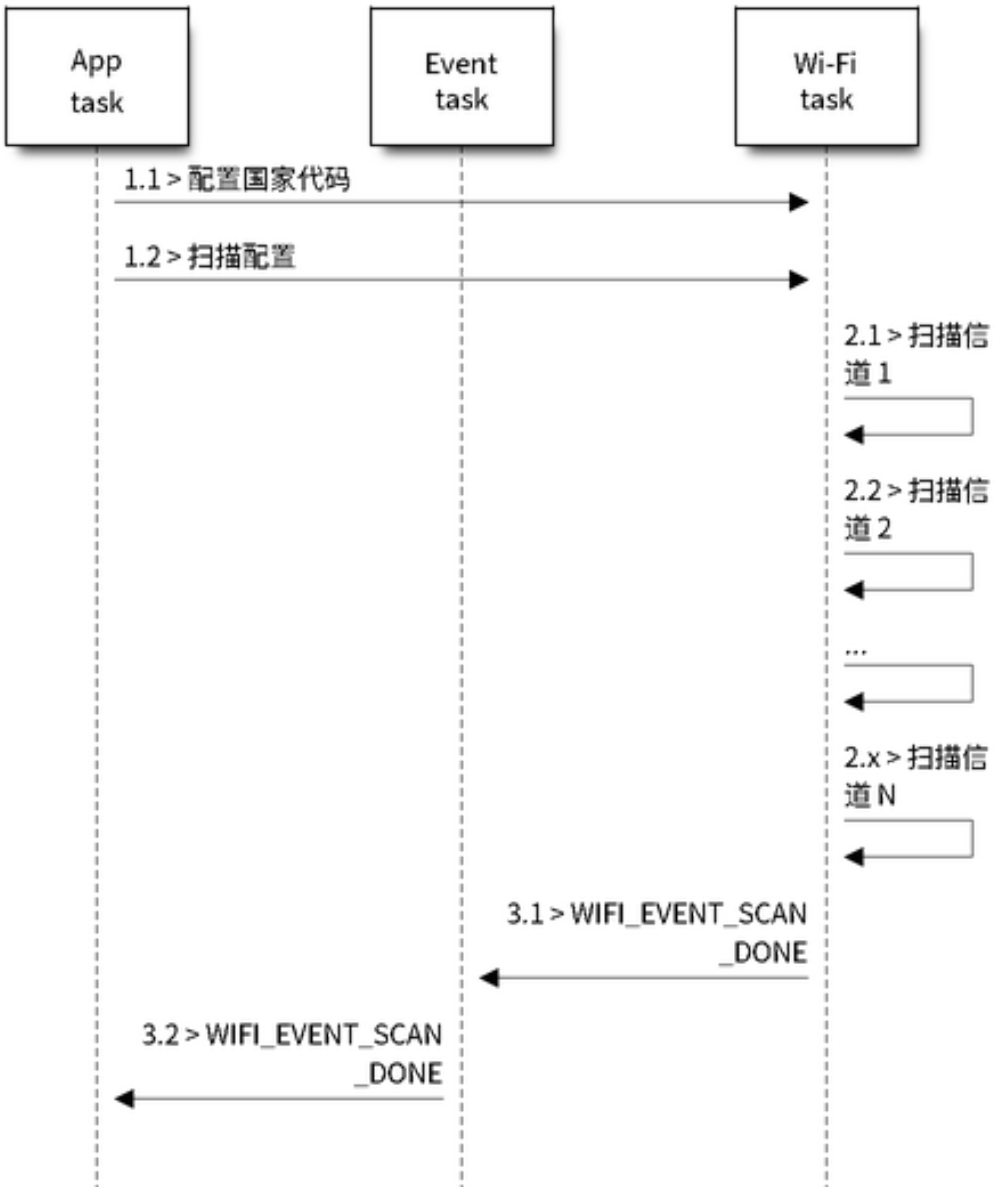


图 29: 所有 Wi-Fi 信道的前端扫描

在所有信道上扫描全部 AP（后端）

场景：

上述场景为一次全信道后端扫描。与*在所有信道中扫描全部 AP（前端）* 相比，全信道后端扫描的不同之处在于：在跳至下一个信道之前，Wi-Fi 驱动程序会先返回主信道停留 30 ms，以便 Wi-Fi 连接有一定的时间发送/接收数据。

在所有信道中扫描特定 AP

场景：

该扫描过程与*在所有信道中扫描全部 AP（前端）* 相似。区别在于：

- s1.1：在步骤 1.2 中，目标 AP 将配置为 SSID/BSSID。
- s2.1 ~ s2.N：每当 Wi-Fi 驱动程序扫描某个 AP 时，它将检查该 AP 是否为目标 AP。如果本次扫描类型为 WIFI_FAST_SCAN，且确认已找到目标 AP，则将产生扫描完成事件，同时结束本次扫描；反之，扫描将继续。请注意，第一个扫描的信道可能不是信道 1，因为 Wi-Fi 驱动程序会优化扫描顺序。

如果有多个匹配目标 AP 信息的 AP，例如：碰巧扫描到两个 SSID 为 “ap” 的 AP。如果本次扫描类型为 WIFI_FAST_SCAN，则仅可找到第一个扫描到的 “ap” ；如果本次扫描类型为 WIFI_ALL_CHANNEL_SCAN，则两个 “ap “都将被找到，且 station 将根据配置规则连接至其需要连接的 “ap”，请参阅*station 基本配置*。

您可以在任意信道中扫描某个特定的 AP，或扫描该信道中的所有 AP。这两种扫描过程也较为相似。

在 Wi-Fi 连接模式下扫描

调用函数`esp_wifi_connect()`后，Wi-Fi 驱动程序将首先尝试扫描已配置的 AP。Wi-Fi 连接模式下的扫描过程与*在所有信道中扫描特定 AP* 过程相同，但连接模式下扫描结束后将不会产生扫描完成事件。如果已找到目标 AP，则 Wi-Fi 驱动程序将开始 Wi-Fi 连接；反之，将产生*WIFI_EVENT_STA_DISCONNECTED* 事件。请参阅*在所有信道中扫描特定 AP*。

在禁用模式下扫描

如果函数`esp_wifi_scan_start()`中的禁用参数为 “true”，则本次扫描为禁用模式下的扫描。在该次扫描完成之前，应用程序任务都将被禁用。禁用模式下的扫描和正常扫描相似，不同之处在于，禁用模式下扫描完成之后将不会出现扫描完成事件。

并行扫描

有时，可能会有两个应用程序任务同时调用函数`esp_wifi_scan_start()`，或者某个应用程序任务在获取扫描完成事件之前再次调用了函数`esp_wifi_scan_start()`。这两种情况都有可能会发生。**但是，Wi-Fi 驱动程序并不足以支持多个并行的扫描。因此，应避免上述并行扫描。**随着 ESP32-C2 的 Wi-Fi 功能不断提升，未来的版本中可能会增加并行扫描支持。

连接 Wi-Fi 时扫描

如果 Wi-Fi 正在连接，则调用函数`esp_wifi_scan_start()`后扫描将立即失败，因为 Wi-Fi 连接优先级高于扫描。如果扫描是因为 Wi-Fi 连接而失败的，此时推荐采取的策略为：等待一段时间后重试。因为一旦 Wi-Fi 连接完成后，扫描将立即成功。

但是，延时重试策略并非万无一失。试想以下场景：

- 如果 station 正在连接一个不存在的 AP，或正在使用错误的密码连接一个 AP，此时将产生事件*WIFI_EVENT_STA_DISCONNECTED*。
- 接收到断开连接事件后，应用程序调用函数`esp_wifi_connect()`进行重新连接。

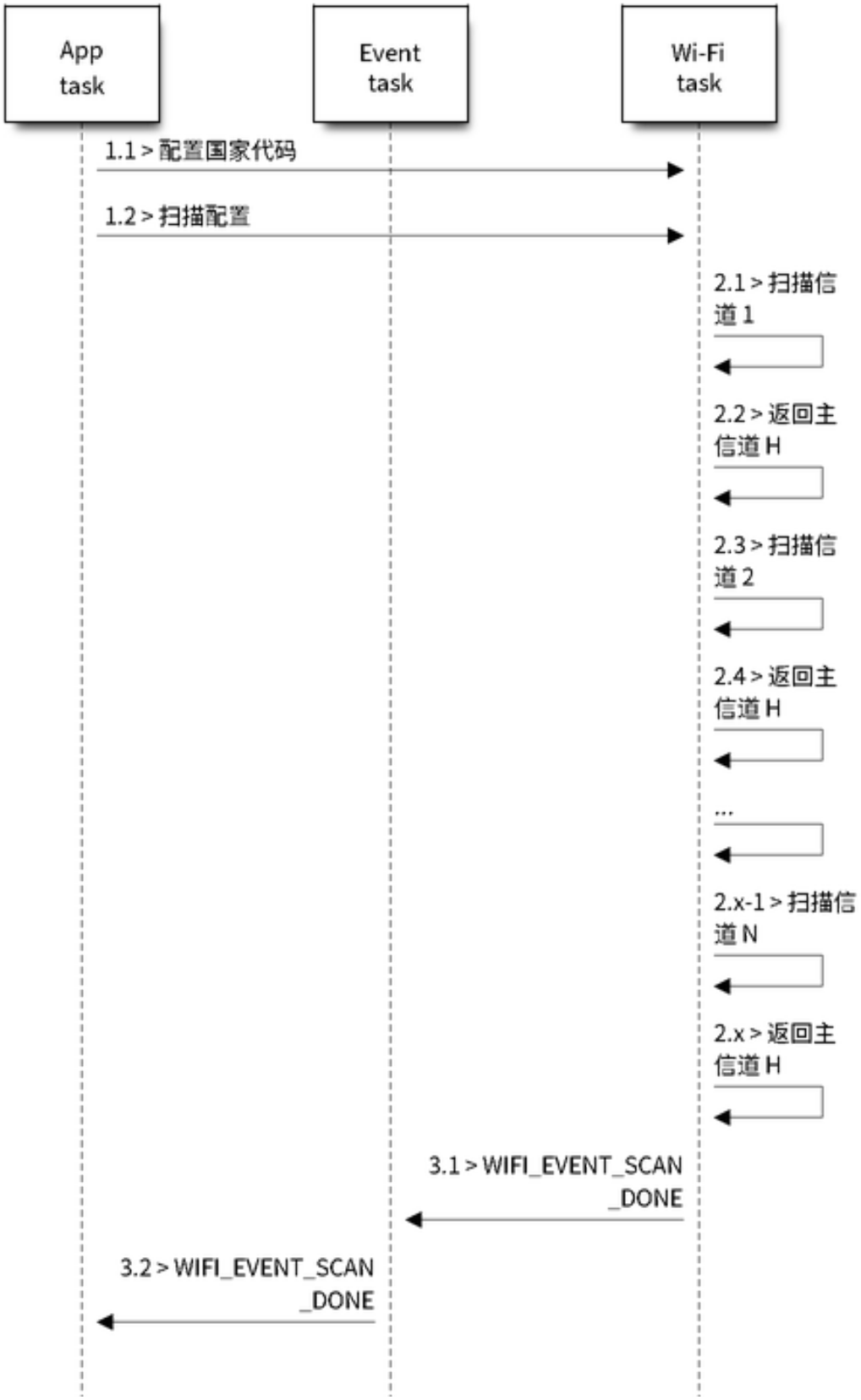


图 30: 所有 Wi-Fi 信道的后端扫描

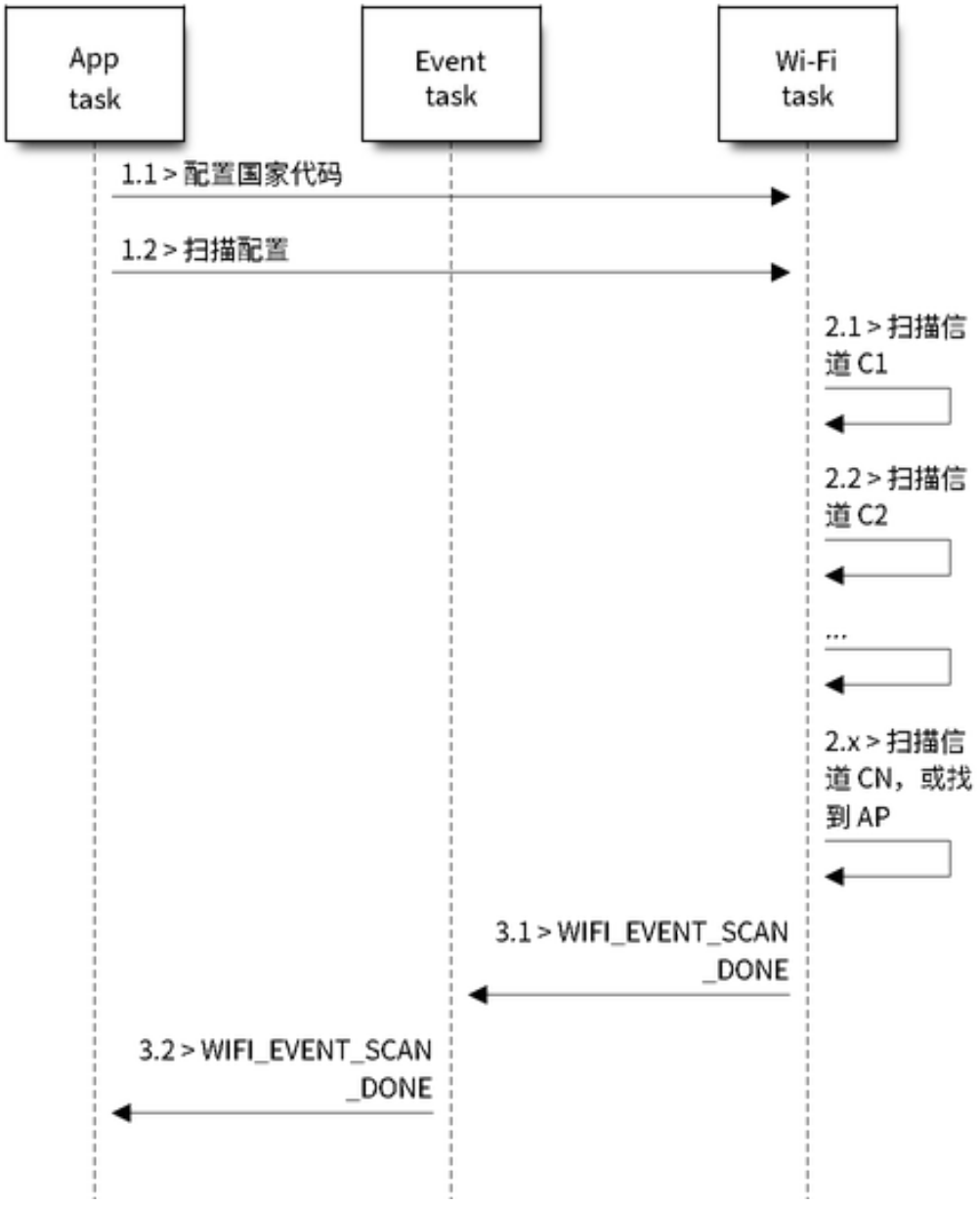


图 31: 扫描特定的 Wi-Fi 信道

- 而另一个应用程序任务（如，控制任务）调用了函数`esp_wifi_scan_start()`进行扫描。这种情况下，每一次扫描都会立即失败，因为 station 一直处于正在连接状态。
- 扫描失败后，应用程序将等待一段时间后进行重新扫描。

上述场景中的扫描永远不会成功，因为 Wi-Fi 一直处于正在连接过程中。因此，如果您的应用程序也可能发生相似的场景，那么就需要为其配置一个更佳的重新连接策略。例如：

- 应用程序可以定义一个连续重新连接次数的最大值，当重新连接的次数达到这个最大值时，立刻停止重新连接。
- 应用程序可以在首轮连续重新连接 N 次后立即进行重新连接，然后延时一段时间后再进行下一次重新连接。

可以给应用程序定义其特殊的重新连接策略，以防止扫描无法成功。请参阅*Wi-Fi 重新连接*。

4.27.10 ESP32-C2 Wi-Fi station 连接场景

该场景仅针对在扫描阶段只找到一个目标 AP 的情况，对于多个相同 SSID AP 的情况，请参阅*找到多个 AP 时的 ESP32-C2 Wi-Fi station 连接*。

通常，应用程序无需关心这一连接过程。如感兴趣，可参看下述简介。

场景：

扫描阶段

- s1.1：Wi-Fi 驱动程序开始在 “Wi-Fi 连接” 模式下扫描。详细信息请参阅*在 Wi-Fi 连接模式下扫描*。
- s1.2：如果未找到目标 AP，将产生*WIFI_EVENT_STA_DISCONNECTED* 事件，且原因代码为 WIFI_REASON_NO_AP_FOUND。请参阅*Wi-Fi 原因代码*。

认证阶段

- s2.1：发送认证请求数据包并使能认证计时器。
- s1.2：如果在认证计时器超时之前未接收到认证响应数据包，将产生*WIFI_EVENT_STA_DISCONNECTED* 事件，且原因代码为 WIFI_REASON_AUTH_EXPIRE。请参阅*Wi-Fi 原因代码*。
- s2.3：接收到认证响应数据包，且认证计时器终止。
- s2.4：AP 在响应中拒绝认证且产生*WIFI_EVENT_STA_DISCONNECTED* 事件，原因代码为 WIFI_REASON_AUTH_FAIL 或为 AP 指定的其它原因。请参阅*Wi-Fi 原因代码*。

关联阶段

- s3.1：发送关联请求并使能关联计时器。
- s3.2：如果在关联计时器超时之前未接收到关联响应，将产生*WIFI_EVENT_STA_DISCONNECTED* 事件，且原因代码为 WIFI_REASON_ASSOC_EXPIRE。请参阅*Wi-Fi 原因代码*。
- s3.3：接收到关联响应，且关联计时器终止。
- s3.4：AP 在响应中拒绝关联且产生*WIFI_EVENT_STA_DISCONNECTED* 事件，原因代码将在关联响应中指定。请参阅*Wi-Fi 原因代码*。

四次握手阶段

- s4.1：使能握手定时器，定时器终止之前未接收到 1/4 EAPOL，此时将产生*WIFI_EVENT_STA_DISCONNECTED* 事件，且原因代码为 WIFI_REASON_HANDSHAKE_TIMEOUT。请参阅*Wi-Fi 原因代码*。
- s4.2：接收到 1/4 EAPOL。
- s4.3：station 回复 2/4 EAPOL。
- s4.4：如果在握手定时器终止之前未接收到 3/4 EAPOL，将产生*WIFI_EVENT_STA_DISCONNECTED* 事件，且原因代码为 WIFI_REASON_HANDSHAKE_TIMEOUT。请参阅*Wi-Fi 原因代码*。

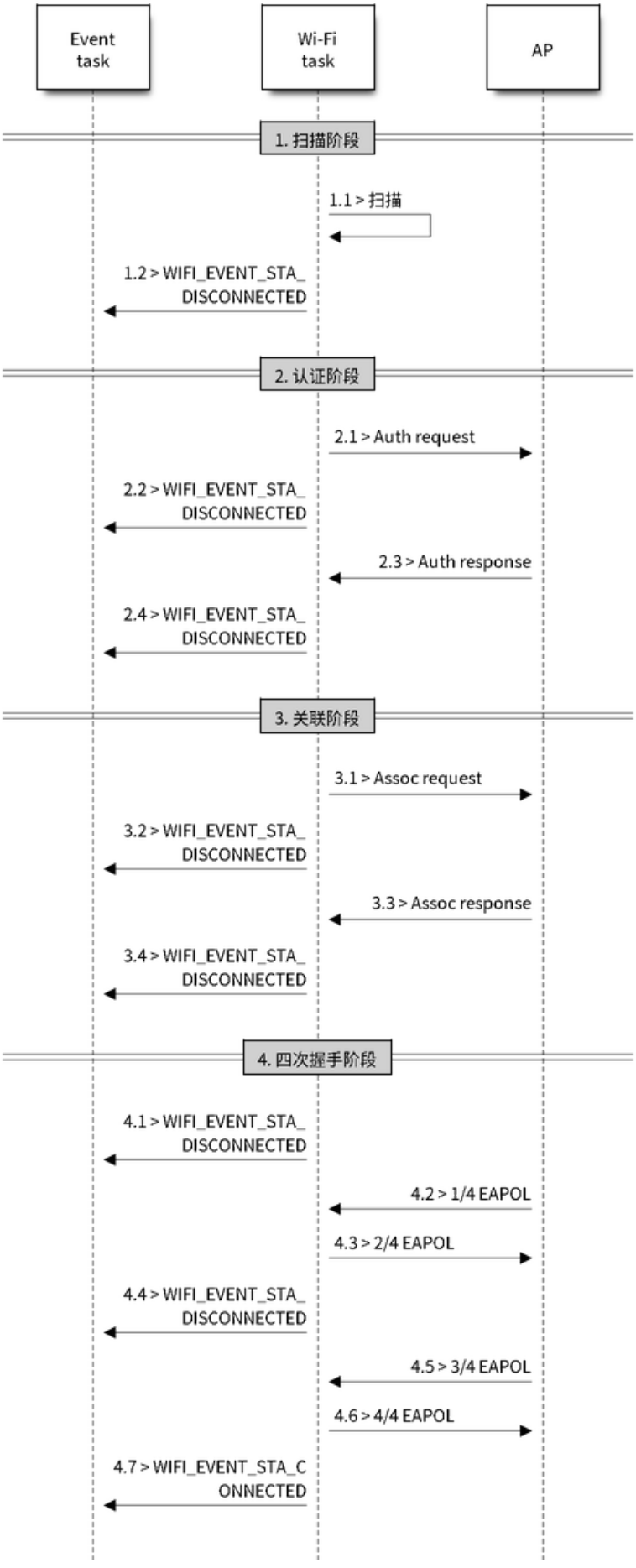


图 32: Wi-Fi station 连接过程

- s4.5：接收到 3/4 EAPOL。
- s4.6：station 回复 4/4 EAPOL。
- s4.7：station 产生 *WIFI_EVENT_STA_CONNECTED* 事件。

Wi-Fi 原因代码

下表罗列了 ESP32-C2 中定义的原因代码。其中，第一列为 esp_wifi_types.h 中定义的宏名称。名称中省去了前缀 *WIFI_REASON*，也就是说，名称 *UNSPECIFIED* 实际应为 *WIFI_REASON_UNSPECIFIED*，以此类推。第二列为原因代码的相应数值。第三列为该原因映射到 IEEE 802.11-2020 中 9.4.1.7 段的标准值。（更多详细信息，请参阅前文描述。）最后一列为这一原因的描述。

原因代码	数值	映射值	描述
UNSPECIFIED	1	1	出现内部错误，例如：内存已满，内部发送失败，或该原因已被远端接收等。
AUTH_EXPIRE	2	2	先前的 authentication 已失效。 对于 ESP station，出现以下情况时将报告该代码： • authentication 超时； • 从 AP 接收到该代码。 对于 ESP AP，出现以下情况时将报告该代码： • 在过去五分钟之内，AP 未从 station 接收到任何数据包； • 由于调用了函数 `esp_wifi_stop()` 导致 AP 终止； • 由于调用了函数 `esp_wifi_deauth_sta()` 导致 station 的 authentication 取消。
AUTH_LEAVE	3	3	authentication 取消，因为发送 station 正在离开（或已经离开）。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
ASSOC_EXPIRE	4	4	因为 AP 不活跃，association 取消。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 对于 ESP AP，出现以下情况时将报告该代码： • 在过去五分钟之内，AP 未从 station 接收到任何数据包； • 由于调用了函数 `esp_wifi_stop()` 导致 AP 终止； • 由于调用了函数 `esp_wifi_deauth_sta()` 导致 station 的 authentication 取消。
ASSOC_TOOMANY	5	5	association 取消，因为 AP 无法同时处理所有当前已关联的 STA。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 对于 ESP AP，出现以下情况时将报告该代码： • 与 AP 相关联的 station 数量已到达 AP 可支持的最大值。

下页继续

表 9 – 续上页

原因代码	数值	映射值	描述
NOT_AUTHED	6	6	从一个未认证 station 接收到 class-2 frame。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 对于 ESP AP，出现以下情况时将报告该代码： • AP 从一个未认证 station 接收到数据包。
NOT_ASSOCED	7	7	从一个未关联 station 接收到的 class-3 frame。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 对于 ESP AP，出现以下情况时将报告该代码： • AP 从未关联 station 接收到数据包。
ASSOC_LEAVE	8	8	association 取消，因为发送 station 正在离开（或已经离开）BSS。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 • 由于调用 `esp_wifi_disconnect()` 和其它 API，station 断开连接。
ASSOC_NOT_AUTHED	9	9	station 的 re(association) 请求未被响应 station 认证。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 对于 ESP AP，出现以下情况时将报告该代码： • AP 从一个已关联，但未认证的 station 接收到数据包。
ASSOC_PWRCAP_BAD	10	10	association 取消，因为无法接收功率能力 (Power Capability) 元素中的信息。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
ASSOC_SUPCHAN_BAD	11	11	association 取消，因为无法接收支持的信道 (Supported Channels) 元素中的信息。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
IE_INVALID	13	13	无效元素，即内容不符合 Wi-Fi 协议中帧格式 (Frame formats) 章节所描述标准的元素。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 对于 ESP AP，出现以下情况时将报告该代码： • AP 解析了一个错误的 WPA 或 RSN IE。
MIC_FAILURE	14	14	消息完整性代码 (MIC) 出错。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
4WAY_HANDSHAKE_TIMEOUT	15	15	四次握手超时。由于某些历史原因，在 ESP 中该原因代码实为 WIFI_REASON_HANDSHAKE_TIMEOUT。 对于 ESP station，出现以下情况时报告该代码： • 握手超时。 • 从 AP 接收到该代码。

下页继续

表 9 – 续上页

原因代码	数值	映射值	描述
GROUP1_KEY_UPDATE_TIMEOUT	16	16	组密钥 (Group-Key) 握手超时。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
IE_IN_4WAY_DIFFERS	17	17	四次握手中产生的元素与 (re-)association 后的 request/probe 以及 response/beacon frame 中的信息不同。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 • station 发现四次握手的 IE 与 (re-)association 后的 request/probe 以及 response/beacon frame 中的 IE 不同。
GROUP1_CIPHER_INVALID	18	18	无效组密文。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
PAIRWISE_CIPHER_INVALID	19	19	无效成对密文。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
AKMP_INVALID	20	20	无效 AKMP。 对于 ESP station，出现以下情况时报告该代码：- 从 AP 接收到该代码。
UNSUPP_RSN_IE_VERSION	21	21	不支持的 RSNE 版本。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
INVALID_RSN_IE_CAP	22	22	无效的 RSNE 性能。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
802_1X_AUTH_FAILED	23	23	IEEE 802.1X. authentication 失败。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。 对于 ESP AP，出现以下情况时将报告该代码： • IEEE 802.1X. authentication 失败。
CIPHER_SUITE_REJECTED	24	24	因安全策略，安全密钥算法套件 (cipher suite) 被拒。 对于 ESP station，出现以下情况时报告该代码： • 从 AP 接收到该代码。
TDLS_PEER_UNREACHABLE	25	25	通过 TDLS 直连无法到达 TDLS 对端 STA，导致 TDLS 直连中断。
TDLS_UNSPECIFIED	26	26	不明原因的 TDLS 直连中断。
SSP_REQUESTED_DISASSOC	27	27	association 取消，由于会话被 SSP request 终止。
NO_SSP_ROAMING_AGREEMENT	28	28	association 取消，由于缺乏 SSP 漫游认证。
BAD_CIPHER_OR_AKM	29	29	请求的服务被拒绝，由于 SSP 密码套件或者 AKM 的需求。
NOT_AUTHORIZED_THIS_LOCATION	30	30	请求的服务在此位置未得到授权。
SERVICE_CHANGE_PRECLUDES_TS	31	31	TS 被删除，原因是：BSS 服务特性或者运行模式改变导致 Qos AP 缺少足够的带宽给 Qos STA 使用（例如：一个 HT BSS 从 40 MHz 的信道切换到 20 MHz 的信道）。

下页继续

表 9 – 续上页

原因代码	数值	映射值	描述
UNSPECIFIED_QOS	32	32	association 取消，由于不明确的 QoS 相关原因。
NOT_ENOUGH_BANDWIDTH	33	33	association 取消，由于 QoS AP 缺少足够的带宽给该 QoS STA 使用。
MISSING_ACKS	34	34	association 取消，原因是：大量的帧需要被确认，但由于 AP 传输或者糟糕的信道条件而没有被确认。
EXCEEDED_TXOP	35	35	association 取消，由于 STA 的传输超过了 TXOPs 的限制。
STA_LEAVING	36	36	请求 STA 离开了 BSS 或者重置了。
END_BA	37	37	请求 STA 不再使用该流或者会话。
UNKNOWN_BA	38	38	请求 STA 使用一种尚未完成的机制接收帧。
TIMEOUT	39	39	对端 STA 的请求超时。
Reserved	40 ~ 45	40 ~ 45	保留
PEER_INITIATED	46	46	在 Disassociation 帧中：已达到授权访问限制。
AP_INITIATED	47	47	在 Disassociation 帧中：外部服务需求。
INVALID_FT_ACTION_FRAME_COUNT	48	48	无效的 FT Action 帧计数。
INVALID_PMKID	49	49	无效的成对主密钥标识符（PMKID）。
INVALID_MDE	50	50	无效的 MDE。
INVALID_FTE	51	51	无效的 FTE。
TRANSMISSION_LINK_ESTABLISHMENT_FAILED	67	67	在备用信道中建立传输链路失败。
ALTERATIVE_CHANNEL_OCCUPIED	68	68	备用信道被占用。
BEACON_TIMEOUT	200	保留	乐鑫特有的 Wi-Fi 原因代码：当 station 连续失去 N 个 beacon，将中断连接并报告该代码。
NO_AP_FOUND	201	保留	乐鑫特有的 Wi-Fi 原因代码：当 station 未扫描到目标 AP 时，将报告该代码。
AUTH_FAIL	202	保留	乐鑫特有的 Wi-Fi 原因代码：authentication 失败，但并非由超时而引发。
ASSOC_FAIL	203	保留	乐鑫特有的 Wi-Fi 原因代码：association 失败，但并非由 ASSOC_EXPIRE 或 ASSOC_TOOMANY 引发。
HANDSHAKE_TIMEOUT	204	保留	乐鑫特有的 Wi-Fi 原因代码：握手失败，与 WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT 中失败原因相同。
CONNECTION_FAIL	205	保留	乐鑫特有的 Wi-Fi 原因代码：AP 连接失败。

与密码错误有关的 Wi-Fi 原因代码

下表罗列了与密码错误相关的 Wi-Fi 原因代码。

原因代码	数值	描述
4WAY_HANDSHAKE_TIMEOUT	15	四次握手超时。STA 在连接加密的 AP 的时候输入了错误的密码
NO_AP_FOUND	201	密码错误会出现这个原因代码的场景有如下两个： • STA 在连接加密的 AP 的时候没有输入密码 • STA 在连接非加密的 AP 的时候输入了密码
HANDSHAKE_TIMEOUT	204	握手超时。

与低 RSSI 有关的 Wi-Fi 原因代码

下表罗列了与低 RSSI 相关的 Wi-Fi 原因代码。

原因代码	数值	描述
NO_AP_FOUND	201	低 RSSI 导致 station 无法扫描到目标 AP
HANDSHAKE_TIMEOUT	204	握手超时。

4.27.11 找到多个 AP 时的 ESP32-C2 Wi-Fi station 连接

该场景与*ESP32-C2 Wi-Fi station 连接场景*相似，不同之处在于该场景中不会产生*WIFI_EVENT_STA_DISCONNECTED*事件，除非 station 无法连接所有找到的 AP。

4.27.12 Wi-Fi 重新连接

出于多种原因，station 可能会断开连接，例如：连接的 AP 重新启动等。应用程序应负责重新连接。推荐使用的方法为：在接收到*WIFI_EVENT_STA_DISCONNECTED*事件后调用函数`esp_wifi_connect()`。

但有时，应用程序需要更复杂的方式进行重新连接：

- 如果断开连接事件是由调用函数`esp_wifi_disconnect()`引发的，那么应用程序可能不希望进行重新连接。
- 如果 station 随时可能调用函数`esp_wifi_scan_start()`开始扫描，此时就需要一个更佳的重新连接方法，请参阅*连接 Wi-Fi 时扫描*。

另一点需要注意的是，如果存在多个具有相同 SSID 的 AP，那么重新连接后可能不会连接到之前的同一个 AP。重新连接时，station 将永远选择最佳的 AP 进行连接。

4.27.13 Wi-Fi beacon 超时

ESP32-C2 使用 beacon 超时机制检测 AP 是否活跃。如果 station 在 inactive 时间内未收到所连接 AP 的 beacon，将发生 beacon 超时。inactive 时间通过调用函数`esp_wifi_set_inactive_time()`设置。

beacon 超时发生后，station 将向 AP 发送 5 个 probe request，如果仍未从 AP 接收到 probe response 或 beacon，station 将与 AP 断开连接并产生*WIFI_EVENT_STA_DISCONNECTED*事件。

需要注意的是，扫描过程中会重置 beacon 超时所使用的定时器，即扫描过程会影响*WIFI_EVENT_STA_BEACON_TIMEOUT*事件的触发。

4.27.14 ESP32-C2 Wi-Fi 配置

使能 Wi-Fi NVS 时，所有配置都将存储到 flash 中；反之，请参阅*Wi-Fi NVS Flash*。

Wi-Fi 模式

调用函数*esp_wifi_set_mode()* 设置 Wi-Fi 模式。

模式	描述
WIFI_MODE_NULL	NULL 模式：此模式下，内部数据结构不分配给 station 和 AP，同时，station 和 AP 接口不会为发送/接收 Wi-Fi 数据进行初始化。通常，此模式用于 Sniffer，或者您不想通过调用函数 *esp_wifi_deinit()* 卸载整个 Wi-Fi 驱动程序来同时停止 station 和 AP。
WIFI_MODE_STA	station 模式：此模式下，*esp_wifi_start()* 将初始化内部 station 数据，同时 station 接口准备发送/接收 Wi-Fi 数据。调用函数*esp_wifi_connect()* 后，station 将连接到目标 AP。
WIFI_MODE_AP	AP 模式：在此模式下，*esp_wifi_start()* 将初始化内部 AP 数据，同时 AP 接口准备发送/接收 Wi-Fi 数据。随后，Wi-Fi 驱动程序开始广播 beacon，AP 即可与其它 station 连接。
WIFI_MODE_APSTA	station/AP 共存模式：在此模式下，函数 *esp_wifi_start()* 将同时初始化 station 和 AP。该步骤在 station 模式和 AP 模式下完成。请注意 ESP station 所连外部 AP 的信道优先于 ESP AP 信道。

station 基本配置

API *esp_wifi_set_config()* 可用于配置 station。配置的参数信息会保存到 NVS 中。下表详细介绍了各个字段。

字段	描述
ssid	station 想要连接的目标 AP 的 SSID。
password	目标 AP 的密码。
scan_method	WIFI_FAST_SCAN 模式下，扫描到一个匹配的 AP 时即结束。WIFI_ALL_CHANNEL_SCAN 模式下，在所有信道扫描所有匹配的 AP。默认扫描模式是 WIFI_FAST_SCAN。
bssid_set	如果 bssid_set 为 0，station 连接 SSID 与“ssid”字段相同的 AP，同时忽略字段“bssid”。其他情况下，station 连接 SSID 与“ssid”字段相同、BSSID 与“bssid”字段也相同的 AP。
bssid	只有当 bssid_set 为 1 时有效。见字段“bssid_set”。
channel	该字段为 0 时，station 扫描信道 1 ~ N 寻找目标 AP；否则，station 首先扫描值与“channel”字段相同的信道，再扫描其他信道。比如，当该字段设置为 3 时，扫描顺序为 3，1，2，…，N 。如果您不知道目标 AP 在哪个信道，请将该字段设置为 0。
sort_method	该字段仅用于 WIFI_ALL_CHANNEL_SCAN 模式。 如果设置为 WIFI_CONNECT_AP_BY_SIGNAL，所有匹配的 AP 将会按照信号强度排序，信号最好的 AP 会被首先连接。比如，如果 station 想要连接 ssid 为“apxx”的 AP，且扫描到两个这样的 AP。第一个 AP 的信号为 -90 dBm，第二个 AP 的信号为 -30 dBm，station 首先连接第二个 AP。除非失败，才会连接第一个。 如果设置为 WIFI_CONNECT_AP_BY_SECURITY，所有匹配的 AP 将会按照安全性排序。比如，如果 station 想要连接 ssid 为“apxx”的 AP，并且扫描到两个这样的 AP。第一个 AP 为开放式，第二个 AP 为 WPA2 加密，station 首先连接第二个 AP。除非失败，才会连接第一个。
threshold	该字段用来筛选找到的 AP，如果 AP 的 RSSI 或安全模式小于配置的阈值，则不会被连接。 如果 RSSI 设置为 0，则表示默认阈值、默认 RSSI 阈值为 -127 dBm。如果 authmode 阈值设置为 0，则表示默认阈值，默认 authmode 阈值无授权。

注意： WEP/WPA 安全模式在 IEEE802.11-2016 协议中已弃用，建议不要使用。可使用 authmode 阈值代替，通过将 threshold.authmode 设置为 WIFI_AUTH_WPA2_PSK 使用 WPA2 模式

AP 基本配置

API *esp_wifi_set_config()* 可用于配置 AP。配置的参数信息会保存到 NVS 中。下表详细介绍了各个字段。

字段	描述
ssid	指 AP 的 SSID。如果 ssid[0] 和 ssid[1] 均为 0xFF，AP 默认 SSID 为 ESP_aabbcc，" aabbcc" 是 AP MAC 的最后三个字节。
password	AP 的密码。如果身份验证模式为 WIFI_AUTH_OPEN，此字段将被忽略。
ssid_len	SSID 的长度。如果 ssid_len 为 0，则检查 SSID 直至出现终止字符。如果 ssid_len 大于 32，请更改为 32，或者根据 ssid_len 设置 SSID 长度。
channel	AP 的信道。如果信道超出范围，Wi-Fi 驱动程序将默认为信道 1。所以，请确保信道在要求的范围内。有关详细信息，请参阅 *Wi-Fi 国家/地区代码*。
authmode	ESP AP 的身份验证模式。目前，ESP AP 不支持 AUTH_WEP。如果 authmode 是一个无效值，AP 默认该值为 WIFI_AUTH_OPEN。
ssid_hidden	如果 ssid_hidden 为 1，AP 不广播 SSID。若为其他值，则广播。
max_connection	允许连接 station 的最大数目，默认值是 2。ESP Wi-Fi 支持 4 (ESP_WIFI_MAX_CONN_NUM) 个 Wi-Fi 连接。请注意，ESP AP 和 ESP-NOW 共享同一块加密硬件 keys，因此 max_connection 参数将受到*CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM* 的影响。加密硬件 keys 的总数是 4，max_connection 最大可以设置为 (17 - *CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM*)。
beacon_interval	beacon 间隔。值为 100 ~ 60000 ms，默认值为 100 ms。如果该值不在上述范围，AP 默认取 100 ms。

Wi-Fi 协议模式

目前，IDF 支持以下协议模式：

协议模式	描述
802.11b	调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B)，将 station/AP 设置为仅 802.11b 模式。
802.11bg	调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B|WIFI_PROTOCOL_11G)，将 station/AP 设置为 802.11bg 模式。
802.11g	调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B|WIFI_PROTOCOL_11G) 和 esp_wifi_config_11b_rate(ifx, true)，将 station/AP 设置为 802.11g 模式。
802.11bgn	调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B|WIFI_PROTOCOL_11G|WIFI_PROTOC 将 station/AP 设置为 802.11bgn 模式。
802.11gn	调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B|WIFI_PROTOCOL_11G|WIFI_PROTOC 和 esp_wifi_config_11b_rate(ifx, true)，将 station/AP 设置为 802.11gn 模式。

Wi-Fi 国家/地区代码

调用*esp_wifi_set_country()*，设置国家/地区信息。下表详细介绍了各个字段，请在配置这些字段之前参考当地的 2.4 GHz RF 操作规定。

字段	描述
cc[3]	国家/地区代码字符串，此属性标识 station/AP 位于的国家/地区或非国家/地区实体。如果是一个国家/地区，该字符串的前两个八位字节是 ISO/IEC3166-1 中规定的国家/地区两位字母代码。第三个八位字节应是下述之一： • ASCII 码空格字符，代表 station/AP 所处国家/地区的规定允许当前频段所需的所有环境。 • ASCII 码‘O’字符，代表 station/AP 所处国家/地区的规定仅允许室外环境。 • ASCII 码‘I’字符，代表 station/AP 所处国家/地区的规定仅允许室内环境。 • ASCII 码‘X’字符，代表 station/AP 位于非国家/地区实体。非国家实体的前两个八位字节是两个 ASCII 码‘XX’字符。 • 当前使用的操作类表编号的二进制形式。见 IEEE Std 802.11-2020 附件 E。
schan	起始信道，station/AP 所处国家/地区规定的最小信道值。
nchan	规定的总信道数，比如，如果 schan=1，nchan=13，那么 station/AP 可以从信道 1 至 13 发送数据。
policy	国家/地区策略，当配置的国家/地区信息与所连 AP 的国家/地区信息冲突时，该字段决定使用哪一信息。更多策略相关信息，可参见下文。

默认国家/地区信息为:

```
wifi_country_t config = {
    .cc = "01",
    .schan = 1,
    .nchan = 11,
    .policy = WIFI_COUNTRY_POLICY_AUTO,
};
```

如果 Wi-Fi 模式为 station/AP 共存模式，则它们配置的国家/地区信息相同。有时，station 所连 AP 的国家/地区信息与配置的不同。例如，配置的 station 国家/地区信息为:

```
wifi_country_t config = {
    .cc = "JP",
    .schan = 1,
    .nchan = 14,
    .policy = WIFI_COUNTRY_POLICY_AUTO,
};
```

但所连 AP 的国家/地区信息为:

```
wifi_country_t config = {
    .cc = "CN",
    .schan = 1,
    .nchan = 13,
};
```

此时，使用所连 AP 的国家/地区信息。

下表描述了在不同 Wi-Fi 模式和不同国家/地区策略下使用的国家/地区信息，并描述了对主动扫描的影响。

Wi-Fi 模式	策略	描述
station 模式	WIFI_COUNTRY_POLICY_AUTO	如果所连 AP 的 beacon 中有国家/地区的 IE，使用的国家/地区信息为 beacon 中的信息，否则，使用默认信息。 扫描时： 主动扫描信道 1 至信道 11，被动扫描信道 12 至信道 14。 请记住，如果带有隐藏 SSID 的 AP 和 station 被设置在被动扫描信道上，被动扫描将无法找到该 AP。也就是说，如果应用程序希望在每个信道中找到带有隐藏 SSID 的 AP，国家/地区信息应该配置为 WIFI_COUNTRY_POLICY_MANUAL。
station 模式	WIFI_COUNTRY_POLICY_MANUAL	总是使用配置的国家/地区信息。 扫描时： 主动扫描信道 schan 至信道 schan+nchan-1。
AP 模式	WIFI_COUNTRY_POLICY_AUTO	总是使用配置的国家/地区信息。
AP 模式	WIFI_COUNTRY_POLICY_MANUAL	总是使用配置的国家/地区信息。
station/AP 共存模式	WIFI_COUNTRY_POLICY_AUTO	station 与 station 模式、WIFI_COUNTRY_POLICY_AUTO 策略下使用的国家/地区信息相同。如果 station 不连接任何外部 AP，AP 使用配置的国家/地区信息。如果 station 连接一个外部 AP，该 AP 的国家/地区信息与该 station 相同。
station/AP 共存模式	WIFI_COUNTRY_POLICY_MANUAL	station 与 station 模式、WIFI_COUNTRY_POLICY_MANUAL 策略下使用的国家/地区信息相同。该 AP 与 AP 模式、WIFI_COUNTRY_POLICY_MANUAL 策略下使用的国家/地区信息相同。

主信道 AP 模式下，AP 的信道定义为主信道。station 模式下，station 所连 AP 的信道定义为主信道。station/AP 共存模式下，AP 和 station 的主信道必须相同。如果不同，station 的主信道始终优先。比如，初始时，AP 位于信道 6，但 station 连接信道 9 的 AP。因为 station 的主信道具有优先性，该 AP 需要将信道从 6 切换至 9，确保与 station 主信道相同。切换信道时，AP 模式下的 ESP32-C2 将使用信道切换公告 (CSA) 通知连接的 station。支持信道切换的 station 将直接通过，无需与 AP 断连再重新连接。

Wi-Fi 供应商 IE 配置

默认情况下，所有 Wi-Fi 管理帧都由 Wi-Fi 驱动程序处理，应用程序不需要任何操作。但是，某些应用程序可能需要处理 beacon、probe request、probe response 和其他管理帧。例如，如果在管理帧中插入一些只针对供应商的 IE，则只有包含此 IE 的管理帧才能得到处理。ESP32-C2 中，`esp_wifi_set_vendor_ie()` 和`esp_wifi_set_vendor_ie_cb()` 负责此类任务。

4.27.15 Wi-Fi Easy Connect™ (DPP)

Wi-Fi Easy Connect™ （也称为设备配置协议）是一个安全且标准化的配置协议，用于配置 Wi-Fi 设备。更多信息请参考*esp_dpp*。

WPA2-Enterprise

WPA2-Enterprise 是企业无线网络的安全认证机制。在连接到接入点之前，它使用 RADIUS 服务器对网络用户进行身份验证。身份验证过程基于 802.1X 标准，并有不同的扩展身份验证协议 (EAP) 方法，如 TLS、TTLS、PEAP 等。RADIUS 服务器根据用户的凭据（用户名和密码）、数字证书或两者对用户进行身份验证。当处于 station 模式的 ESP32-C2 尝试连接到企业模式的 AP 时，它会向 AP 发送身份验证请求，AP

会将该请求发送到 RADIUS 服务器以对 station 进行身份验证。根据不同的 EAP 方式，可以通过 `idf.py menuconfig` 打开配置，并在配置中设置参数。ESP32-C2 仅在 station 模式下支持 WPA2_Enterprise。

为了建立安全连接，AP 和 station 协商并就要使用的最佳密码套件达成一致。ESP32-C2 支持 AKM 的 802.1X/EAP (WPA) 方法和 AES-CCM（高级加密标准-带密码块链消息验证码协议的计数器模式）支持的密码套件。如果设置了 *USE_MBEDTLS_CRYPTO* 标志，ESP32-C2 也支持 mbedtls 支持的密码套件。

目前，ESP32-C2 支持以下 EAP 方法：

- EAP-TLS: 这是基于证书的方法，只需要 SSID 和 EAP-IDF。
- PEAP: - PEAP：这是受保护的 EAP 方法。用户名和密码是必填项。
- **EAP-TTLS: 这是基于凭据的方法。只有服务器身份验证是强制性的，而用户身份验证是可选的。用户名和密**
  - PAP: 密码认证协议
  - CHAP: 询问握手身份验证协议
  - MSCHAP 和 MSCHAP-V2
- EAP-FAST: 这是一种基于受保护的访问凭据 (PAC) 的认证方法，使用身份验证和密码。目前使用此功能时需要禁用 USE_MBEDTLS_CRYPTO 标志。

请查看 wifi/wifi_enterprise 获取关于证书创建以及如何在 ESP32-C2 上运行 wpa2_enterprise 示例的详细信息。

4.27.16 无线网络管理

无线网络管理让客户端设备能够交换有关网络拓扑结构的信息，包括与射频环境相关的信息。这使每个客户端都能感知网络状况，从而促进无线网络性能的整体改进。这是 802.11v 规范的一部分。它还使客户端能够支持网络辅助漫游。网络辅助漫游让 WLAN 能够向关联的客户端发送消息，从而使客户端与具有更好链路指标的 AP 关联。这对于促进负载平衡以及引导连接不良的客户端都很有用。

目前 802.11v 的实现支持 BSS 过渡管理帧。

4.27.17 无线资源管理

无线电资源测量（802.11k）旨在改善网络内流量的分配方式。在无线局域网中，一般情况下，无线设备会连接发射信号最强的接入点 (AP)。根据用户的数量和地理位置，这种分配方式有时会导致某个接入点超负荷而其它接入点利用不足，从而导致整体网络性能下降。在符合 802.11k 规范的网络中，如果信号最强的 AP 已满负荷加载，无线设备则转移到其它未充分利用的 AP。尽管信号可能较弱，但由于更有效地利用了网络资源，总体吞吐量会更大。

目前 802.11k 的实现支持信标测量报告、链路测量报告和邻居请求。

请参考 IDF 示例程序 examples/wifi/roaming/README.md 来设置和使用这些 API。示例代码只演示了如何使用这些 API，应用程序应根据需要定义自己的算法和案例。

4.27.18 ESP32-C2 Wi-Fi 节能模式

station 睡眠

目前，ESP32-C2 Wi-Fi 支持 Modem-sleep 模式，该模式是 IEEE 802.11 协议中的传统节能模式。仅 station 模式支持该模式，station 必须先连接到 AP。如果使能了 Modem-sleep 模式，station 将定期在活动状态和睡眠状态之间切换。在睡眠状态下，RF、PHY 和 BB 处于关闭状态，以减少功耗。Modem-sleep 模式下，station 可以与 AP 保持连接。

Modem-sleep 模式包括最小和最大节能模式。在最小节能模式下，每个 DTIM 间隔，station 都将唤醒以接收 beacon。广播数据在 DTIM 之后传输，因此不会丢失。但是，由于 DTIM 间隔长短由 AP 决定，如果该间隔时间设置较短，则省电效果不大。

在最大节能模式下，每个监听间隔，station 都将唤醒以接收 beacon。可以设置该监听间隔长于 AP 的 DTIM 周期。在 DTIM 期间内，station 可能处于睡眠状态，广播数据会丢失。如果监听间隔较长，则可以

节省更多电量，但广播数据更容易丢失。连接 AP 前，可以通过调用 API *esp_wifi_set_config()* 配置监听间隔。

调用*esp_wifi_init()* 后，调用 esp_wifi_set_ps(WIFI_PS_MIN_MODEM) 可使能 Modem-sleep 最小节能模式。调用 esp_wifi_set_ps(WIFI_PS_MAX_MODEM) 可使能 Modem-sleep 最大节能模式。station 连接到 AP 时，Modem-sleep 模式将启动。station 与 AP 断开连接时，Modem-sleep 模式将停止。

调用 esp_wifi_set_ps(WIFI_PS_NONE) 可以完全禁用 Modem-sleep 模式。禁用会增大功耗，但可以最大限度减少实时接收 Wi-Fi 数据的延迟。使能 Modem-sleep 时，接收 Wi-Fi 数据的延迟时间可能与 DTIM 周期（最小节能模式）或监听间隔（最大节能模式）相同。在 Wi-Fi 与 Bluetooth LE 共存模式下，无法完全禁用 modem-sleep 模式。

默认的 Modem-sleep 模式是 WIFI_PS_MIN_MODEM。

AP 睡眠

目前，ESP32-C2 AP 不支持 Wi-Fi 协议中定义的所有节能功能。具体来说，AP 只缓存所连 station 单播数据，不缓存组播数据。如果 ESP32-C2 AP 所连的 station 已使能节能功能，可能发生组播数据包丢失。

未来，ESP32-C2 AP 将支持所有节能功能。

非连接状态下的休眠

非连接状态指的是*esp_wifi_start()* 至*esp_wifi_stop()* 期间内，没有建立 Wi-Fi 连接的阶段。

目前，ESP32-C2 Wi-Fi 支持以 station 模式运行时，在非连接状态下休眠。可以通过选项*CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE* 配置该功能。

如果打开配置选项*CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE*，则在该阶段内，RF, PHY and BB 将在空闲时被关闭，电流将会等同于 Modem-sleep 模式下的休眠电流。

配置选项*CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE* 默认情况下将会被打开，共存模式下被 Menuconfig 强制打开。

非连接模块功耗管理

非连接模块指的是一些不依赖于 Wi-Fi 连接的 Wi-Fi 模块，例如 ESP-NOW，DPP，FTM。这些模块从*esp_wifi_start()* 开始工作至*esp_wifi_stop()* 结束。

目前，ESP-NOW 以 station 模式工作时，既支持在连接状态下休眠，也支持在非连接状态下休眠。

非连接模块发包 对于任何非连接模块，在开启了休眠的任何时间点都可以发包，不需要进行任何额外的配置。

此外，*esp_wifi_80211_tx()* 也在休眠时被支持。

非连接模块收包 对于非连接模块，在开启休眠时如果需要进行收包，需要配置两个参数，分别为 *Window* 和 *Interval*。

在每个 *Interval* 开始时，RF, PHY and BB 将会被打开并保持 *Window* 的时间。非连接模块可以在此时间内收包。

Interval

- 全局只有一个 *Interval* 参数，所有非连接模块共享它。其数值由 API esp_wifi_set_connectionless_interval() 配置，单位为毫秒。
- *Interval* 的默认值为 *ESP_WIFI_CONNECTIONLESS_INTERVAL_DEFAULT_MODE* 。
- 在 *Interval* 开始时，将会给出 *WIFI_EVENT_CONNECTIONLESS_MODULE_WAKE_INTERVAL_START* 事件，由于 *Window* 将在此时开始，可以在此事件内布置发包动作。
- 在连接状态下，*Interval* 开始的时间点将会与 TBTT 时间点对齐。

Window

- 每个非连接模块在启动后都有其自身的 *Window* 参数，休眠模块将取所有模块 *Window* 的最大值运作。
- 其数值由 API `module_name_set_wake_window()` 配置，单位为毫秒。
- 模块 *Window* 的默认值为最大值。

表 10: 不同 Window 与 Interval 组合下的 RF, PHY and BB 使用情况

		Interval	
		ESP_WIFI_CONNECTIONLESS_INTERVAL_DEFAULT_MODE	1 - maximum
Window	0	not used	
	1 - maximum	default mode	used periodically (Window < Interval) / used all time (Window ≥ Interval)

默认模式 当 *Interval* 参数被配置为 *ESP_WIFI_CONNECTIONLESS_INTERVAL_DEFAULT_MODE*，且有非零的 *Window* 参数时，非连接模块功耗管理将会按默认模式运行。

在没有与非 Wi-Fi 协议共存时，RF, PHY and BB 将会在默认模式下被一直打开。

在与非 Wi-Fi 协议共存时，RF, PHY and BB 资源被共存模块分时划给 Wi-Fi 非连接模块和非 Wi-Fi 协议使用。在默认模式下，Wi-Fi 非连接模块被允许周期性使用 RF, PHY and BB，并且具有稳定性能。

推荐在与非 Wi-Fi 协议共存时将非连接模块功耗管理配置为默认模式。

4.27.19 ESP32-C2 Wi-Fi 吞吐量

下表是我们在 Espressif 实验室和屏蔽箱中获得的最佳吞吐量结果。

4.27.20 Wi-Fi 80211 数据包发送

esp_wifi_80211_tx() API 可用于：

- 发送 beacon、probe request、probe response 和 action 帧。
- 发送非 QoS 数据帧。

不能用于发送加密或 QoS 帧。

使用 esp_wifi_80211_tx() 的前提条件

- Wi-Fi 模式为 station 模式，AP 模式，或 station/AP 共存模式。
- API esp_wifi_set_promiscuous(true) 或*esp_wifi_start()*，或者二者都返回 ESP_OK。这是为确保在调用函数*esp_wifi_80211_tx()* 前，Wi-Fi 硬件已经初始化。对于 ESP32-C2，esp_wifi_set_promiscuous(true) 和*esp_wifi_start()* 都可以触发 Wi-Fi 硬件初始化。
- 提供正确的*esp_wifi_80211_tx()* 参数。

传输速率

- 默认传输速率为 1 Mbps。
- 可以通过函数*esp_wifi_config_80211_tx_rate()* 设置任意速率。
- 可以通过函数*esp_wifi_set_bandwidth()* 设置任意带宽。

在不同情况下需要避免的副作用

理论上，如果不考虑 API 对 Wi-Fi 驱动程序或其他 station 或 AP 的副作用，可以通过空中发送一个原始的 802.11 数据包，包括任何目的地址的 MAC、任何源地址的 MAC、任何 BSSID、或任何其他类型的数据包。但是，一个具有强健、有用的应用程序应该避免这种副作用。下表针对如何避免*esp_wifi_80211_tx()*的副作用提供了一些提示或建议。

场景	描述
无 Wi-Fi 连接	在这种情况下，因为没有 Wi-Fi 连接，Wi-Fi 驱动程序不会受到副作用影响。如果 en_sys_seq==true，则 Wi-Fi 驱动程序负责序列控制。如果 en_sys_seq==false，应用程序需要确保缓冲区的序列正确。 理论上，MAC 地址可以是任何地址。但是，这样可能会影响其他使用相同 MAC/BSSID 的 station/AP。 例如，AP 模式下，应用程序调用函数 esp_wifi_80211_tx() 发送带有 BSSID == mac_x 的 beacon，但是 mac_x 并非 AP 接口的 MAC。而且，还有另一个 AP（我们称之为“other-AP”）的 bssid 是 mac_x。因此，连接到“other-AP”的 station 无法分辨 beacon 来自“other-AP”还是 esp_wifi_80211_tx()，就会出现“意外行为”。 为了避免上述副作用，我们建议： • 如果在 station 模式下调用函数 esp_wifi_80211_tx()，第一个 MAC 应该是组播 MAC 或是目标设备的 MAC，第二个 MAC 应该是 station 接口的 MAC。 • 如果在 AP 模式下调用函数 esp_wifi_80211_tx，第一个 MAC 应该是组播 MAC 或是目标设备的 MAC，第二个 MAC 应该是 AP 接口的 MAC。 上述建议仅供避免副作用，在有充分理由的情况下可以忽略。
有 Wi-Fi 连接	当 Wi-Fi 已连接，且序列由应用程序控制，应用程序可能会影响整个 Wi-Fi 连接的序列控制。因此，en_sys_seq 要为 true，否则将返回 ESP_ERR_WIFI_ARG。 “无 Wi-Fi 连接”情况下的 MAC 地址建议也适用于此情况。 如果 Wi-Fi 模式是 station 模式，MAC 的地址 1 是 station 所连 AP 的 MAC，地址 2 是 station 接口的 MAC，那么就称数据包是从 station 发送到 AP。另一方面，如果 Wi-Fi 模式是 AP 模式，且 MAC 地址 1 是该 AP 所连 station 的 MAC，地址 2 是 AP 接口的 MAC，那么就称数据包是从 AP 发送到 station。为避免与 Wi-Fi 连接冲突，可采用以下检查方法： • 如果数据包类型是数据，且是从 station 发送到 AP，IEEE 802.11 Frame control 字段中的 ToDS 位应该为 1，FromDS 位为 0，否则，Wi-Fi 驱动程序不接受该数据包。 • 如果数据包类型是数据，且是从 AP 发送到 station，IEEE 802.11 Frame control 字段中的 ToDS 位应该为 0，FromDS 位为 1，否则，Wi-Fi 驱动程序不接受该数据包。 • 如果数据包是从 station 发送到 AP，或从 AP 到 station，Power Management、More Data 和 Re-Transmission 位应该为 0，否则，Wi-Fi 驱动程序不接受该数据包。 如果任何检查失败，将返回 ESP_ERR_WIFI_ARG。

4.27.21 Wi-Fi Sniffer 模式

Wi-Fi Sniffer 模式可以通过*esp_wifi_set_promiscuous()*使能。如果使能 Sniffer 模式，**可以**向应用程序转储以下数据包。

- 802.11 管理帧
- 802.11 数据帧，包括 MPDU、AMPDU、AMSDU 等
- 802.11 MIMO 帧，Sniffer 模式仅转储 MIMO 帧的长度。
- 802.11 控制帧
- 802.11 CRC 错误帧

不可以向应用程序转储以下数据包。

- 802.11 其它错误帧

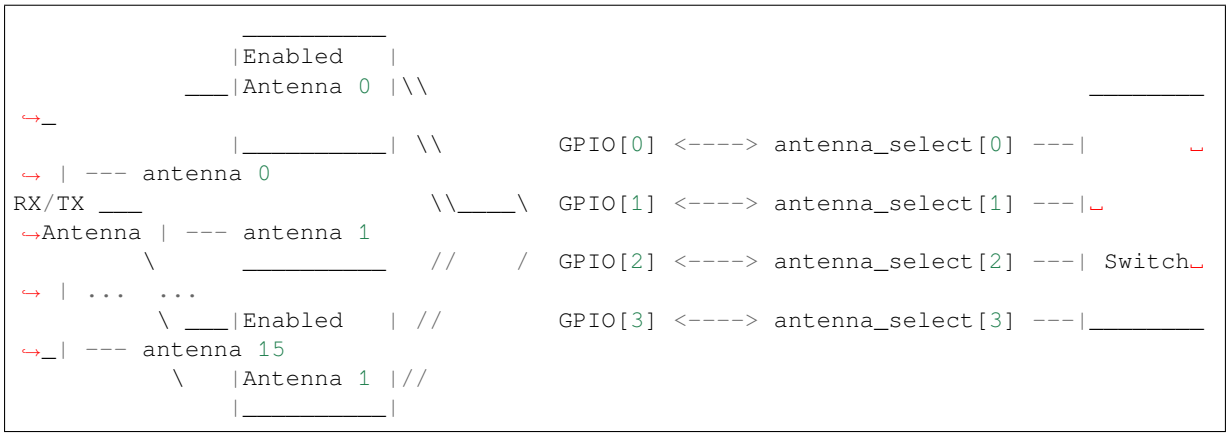
对于 Sniffer 模式 **可以**转储的帧，应用程序可以另外使用*esp_wifi_set_promiscuous_filter()*和*esp_wifi_set_promiscuous_ctrl_filter()*决定筛选哪些特定类型的数据包。应用程序默认筛选所有 802.11 数据和管理帧。

可以在 WIFI_MODE_NULL、WIFI_MODE_STA、WIFI_MODE_AP、WIFI_MODE_APSTA 等 Wi-Fi 模式下使能 Wi-Fi Sniffer 模式。也就是说，当 station 连接到 AP，或者 AP 有 Wi-Fi 连接时，就可以使能。请注意，Sniffer 模式对 station/AP Wi-Fi 连接的吞吐量有 **很大影响**。通常，除非有特别原因，当 station/AP Wi-Fi 连接出现大量流量，不应使能。

该模式下还应注意回调函数 wifi_promiscuous_cb_t 的使用。该回调将直接在 Wi-Fi 驱动程序任务中进行，所以如果应用程序需处理大量过滤的数据包，建议在回调中向应用程序任务发布一个事件，把真正的工作推迟到应用程序任务中完成。

4.27.22 Wi-Fi 多根天线

下图描述 Wi-Fi 多根天线的选择过程:



ESP32-C2 通过外部天线开关，最多支持 16 根天线。天线开关最多可由四个地址管脚控制 - antenna_select[0:3]。向 antenna_select[0:3] 输入不同的值，以选择不同的天线。例如，输入值 '0b1011' 表示选中天线 11 。antenna_select[3:0] 的默认值为 "0b0000"，表示默认选择了天线 0。

四个高电平有效 antenna_select 管脚有多达四个 GPIO 连接。ESP32-C2 可以通过控制 GPIO[0:3] 选择天线。API *esp_wifi_set_ant_gpio()* 用于配置 antenna_selects 连接哪些 GPIO。如果 GPIO[x] 连接到 antenna_select[x], gpio_config->gpio_cfg[x].gpio_select 应设置为 1，且要提供 gpio_config->gpio_cfg[x].gpio_num 的值。

天线开关的具体实现不同，*antenna_select[0:3]* 的输入值中可能存在非法值，即 ESP32-C2 通过外部天线开关支持的天线数可能小于 16 根。例如，ESP32-WROOM-DA 使用 RTC6603SP 作为天线开关，仅支持 2 根天线。两个天线选择输入管脚为高电平有效，连接到两个 GPIO。'0b01' 表示选中天线 0，'0b10' 表示选中天线 1。输入值 '0b00' 和 '0b11' 为非法值。

尽管最多支持 16 根天线，发送和接收数据时，最多仅能同时使能两根天线。API *esp_wifi_set_ant()* 用于配置使能哪些天线。

使能天线后，选择算法的过程同样可由*esp_wifi_set_ant()* 配置。接收/发送数据源的天线模式可以是 WIFI_ANT_MODE_ANT0、WIFI_ANT_MODE_ANT1 或 WIFI_ANT_MODE_AUTO。如果天线模式为 WIFI_ANT_MODE_ANT0，使能的天线 0 用于接收/发送数据。如果天线模式为 WIFI_ANT_MODE_ANT1，使能天线 1 用于接收/发送数据。否则，Wi-Fi 会自动选择使能天线中信号较好的天线。

如果接收数据的天线模式为 WIFI_ANT_MODE_AUTO，还需要设置默认天线模式。只有在满足某些条件时，接收数据天线才会切换，例如，如果 RSSI 低于 -65 dBm，或另一根天线信号更好。如果条件不满足，接收数据使用默认天线。如果默认天线模式为 WIFI_ANT_MODE_ANT1，则使能的天线 1 是默认接收数据天线，否则是使能的天线 0。

有一些限制情况需要考虑:

- 因为发送数据天线基于 WIFI_ANT_MODE_AUTO 类型的接收数据天线选择算法，只有接收数据的天线模式为 WIFI_ANT_MODE_AUTO 时，发送数据天线才能设置为 WIFI_ANT_MODE_AUTO。
- 目前，Bluetooth® 不支持多根天线功能，请不要使用与多根天线有关的 API。

推荐在以下场景中使用多根天线:

- Wi-Fi 模式 WIFI_MODE_STA 下，接收/发送数据的天线模式均配置为 WIFI_ANT_MODE_AUTO。Wi-Fi 驱动程序自动选择更好的接收/发送数据天线。
- 接收数据天线模式配置为 WIFI_ANT_MODE_AUTO。发送数据的天线模式配置为 WIFI_ANT_MODE_ANT0 或 WIFI_ANT_MODE_ANT1。应用程序可以始终选择指定的天线用于发送数据，也可以执行自身发送数据天线选择算法，如根据信道切换信息选择发送数据的天线模式等。
- 接收/发送数据的天线模式均配置为 WIFI_ANT_MODE_ANT0 或 WIFI_ANT_MODE_ANT1。

Wi-Fi 多根天线配置

通常，可以执行以下步骤来配置多根天线：

- 配置 antenna_selects 连接哪些 GPIOs，例如，如果支持四根天线，且 GPIO20/GPIO21 连接到 antenna_select[0]/antenna_select[1]，配置如下所示：

```
wifi_ant_gpio_config_t config = {
    { .gpio_select = 1, .gpio_num = 20 },
    { .gpio_select = 1, .gpio_num = 21 }
};
```

- 配置使能哪些天线、以及接收/发送数据如何使用使能的天线，例如，如果使能了天线 1 和天线 3，接收数据需要自动选择较好的天线，并将天线 1 作为默认天线，发送数据始终选择天线 3。配置如下所示：

```
wifi_ant_config_t config = {
    .rx_ant_mode = WIFI_ANT_MODE_AUTO,
    .rx_ant_default = WIFI_ANT_ANT0,
    .tx_ant_mode = WIFI_ANT_MODE_ANT1,
    .enabled_ant0 = 1,
    .enabled_ant1 = 3
};
```

4.27.23 Wi-Fi HT20/40

ESP32-C2 仅支持 Wi-Fi 带宽 HT20，不支持 Wi-Fi 带宽 HT40 或 HT20/40 共存。

4.27.24 Wi-Fi QoS

ESP32-C2 支持 WFA Wi-Fi QoS 认证所要求的所有必备功能。

Wi-Fi 协议中定义了四个 AC （访问类别），每个 AC 有各自的优先级访问 Wi-Fi 信道。此外，还定义了映射规则以映射其他协议的 QoS 优先级，例如 802.11D 或 TCP/IP 到 Wi-Fi AC。

下表描述 ESP32-C2 中 IP 优先级如何映射到 Wi-Fi AC，还指明此 AC 是否支持 AMPDU。该表按优先级降序排列，即 AC_VO 拥有最高优先级。

IP 优先级	Wi-Fi AC	是否支持 AMPDU
6, 7	AC_VO (Voice)	否
4, 5	AC_VI (Video)	是
3, 0	AC_BE (Best Effort)	是
1, 2	AC_BK (Background)	是

应用程序可以通过套接字选项 IP_TOS 配置 IP 优先级使用 QoS 功能。下面是使套接字使用 VI 队列的示例：

```
const int ip_precedence_vi = 4;
const int ip_precedence_offset = 5;
int priority = (ip_precedence_vi << ip_precedence_offset);
setsockopt(socket_id, IPPROTO_IP, IP_TOS, &priority, sizeof(priority));
```

理论上，高优先级的 AC 比低优先级 AC 具有更好的性能，但并非总是如此，下面是一些关于如何使用 Wi-Fi QoS 的建议：

- 可以把一些真正重要的应用程序流量放到 AC_VO 队列中。避免通过 AC_VO 队列发送大流量。一方面，AC_VO 队列不支持 AMPDU，如果流量很大，性能不会优于其他队列。另一方面，可能会影响同样使用 AC_VO 队列的管理帧。
- 避免使用 AMPDU 支持的、两个以上的不同优先级，比如 socket A 使用优先级 0，socket B 使用优先级 1，socket C 使用优先级 2。因为可能需要更多的内存，不是好的设计。具体来说，Wi-Fi 驱动程序可能会为每个优先级生成一个 Block Ack 会话，如果设置了 Block Ack 会话，则需要更多内存。

4.27.25 Wi-Fi AMSDU

ESP32-C2 支持接收 AMSDU。

4.27.26 Wi-Fi 分片

4.27.27 WPS 注册

在 Wi-Fi 模式 WIFI_MODE_STA 或 WIFI_MODE_APSTA 下，ESP32-C2 支持 WPS 注册功能。目前，ESP32-C2 支持的 WPS enrollee 类型有 PBC 和 PIN。

4.27.28 Wi-Fi 缓冲区使用情况

本节只介绍动态缓冲区配置。

缓冲区配置的重要性

为了获得一个具有强健、高性能的系统，我们需要非常谨慎地考虑内存的使用或配置情况，因为：

- ESP32-C2 的可用内存有限。
- 目前，LwIP 和 Wi-Fi 驱动程序中默认的缓冲区类型是“动态”，**意味着 LwIP 和 Wi-Fi 都与应用程序共享内存**。程序员应该时刻牢记这一点，否则将面临如“堆内存耗尽”等的内存问题。
- “堆耗尽”情况非常危险，会导致 ESP32-C2 出现“未定义行为”。因此，应该为应用程序预留足够的堆内存，防止耗尽。
- Wi-Fi 的吞吐量很大程度上取决于与内存相关的配置，如 TCP 窗口大小、Wi-Fi 接收/发送数据动态缓冲区数量等。
- ESP32-C2 LwIP/Wi-Fi 可能使用的堆内存峰值取决于许多因素，例如应用程序可能拥有的最大 TCP/UDP 连接等。
- 在考虑内存配置时，应用程序所需的总内存也是一个重要因素。

由于这些原因，不存在一个适合所有应用程序的配置。相反，我们必须为每个不同的应用程序考虑不同的内存配置。

动态与静态缓冲区

Wi-Fi 驱动程序中默认的缓存类型是“动态”。大多数情况下，动态缓冲区可以极大地节省内存。但是因为应用程序需要考虑 Wi-Fi 的内存使用情况，会给应用程序编程造成一定的难度。

lwIP 还在 TCP/IP 层分配缓冲区，这种缓冲区分配也是动态的。具体内容，见 lwIP 文档内存使用和性能部分。

Wi-Fi 动态缓冲区峰值

Wi-Fi 驱动程序支持多种类型的缓冲区（参考*Wi-Fi 缓冲区配置*）。但本节只介绍 Wi-Fi 动态缓冲的使用方法 Wi-Fi 使用的堆内存峰值是 Wi-Fi 驱动程序 **理论上消耗的最大内存**。通常，该内存峰值取决于：

- 配置的动态接收数据缓冲区数：wifi_rx_dynamic_buf_num
- 配置的动态发送数据缓冲区数：wifi_tx_dynamic_buf_num
- Wi-Fi 驱动程序可以接收的最大数据包：wifi_rx_pkt_size_max
- Wi-Fi 驱动程序可以发送的最大数据包：wifi_tx_pkt_size_max

因此，Wi-Fi 驱动程序消耗的内存峰值可以用下面的公式计算：

wifi_dynamic_peek_memory = (wifi_rx_dynamic_buf_num * wifi_rx_pkt_size_max) + (wifi_tx_dynamic_buf_num * wifi_tx_pkt_size_max)

一般情况下，不需要关心动态发送数据长缓冲区和超长缓冲区，因为它们是管理帧，对系统的影响很小。

4.27.29 如何提高 Wi-Fi 性能

ESP32-C2 Wi-Fi 的性能受许多参数的影响，各参数之间存在相互制约。如果配置地合理，不仅可以提高性能，还可以增加应用程序的可用内存，提高稳定性。

在本节中，我们将简单介绍 Wi-Fi/LWIP 协议栈的工作模式，并说明各个参数的作用。我们将推荐几种配置等级，您可以根据使用场景选择合适的等级。

协议栈工作模式

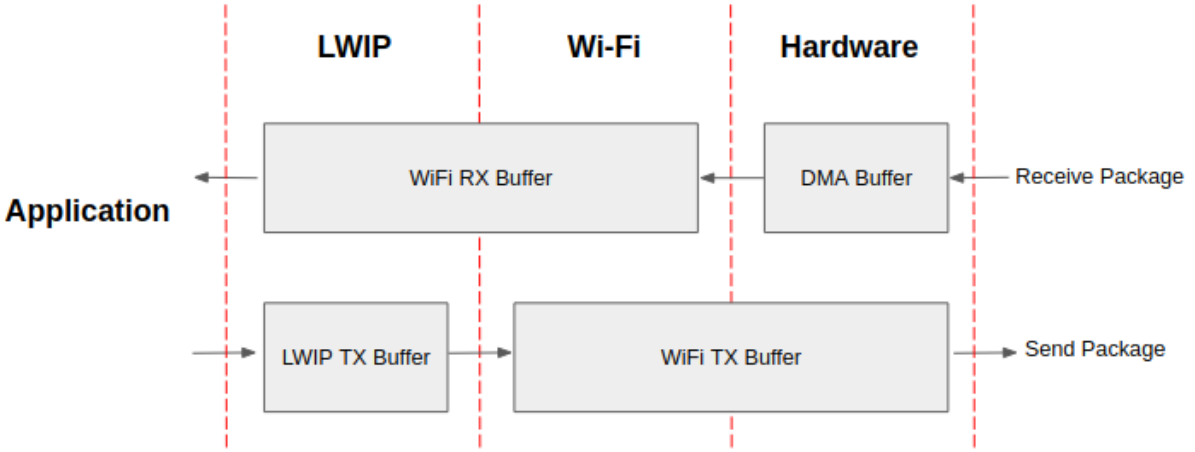


图 33: ESP32-C2 数据路径

ESP32-C2 协议栈分为四层，分别为应用层、LWIP 层、Wi-Fi 层和硬件层。

- 在接收过程中，硬件将接收到的数据包放入 DMA 缓冲区，然后依次传送到 Wi-Fi 的接收数据缓冲区、LWIP 的接收数据缓冲区进行相关协议处理，最后传送到应用层。Wi-Fi 的接收数据缓冲区和 LWIP 的接收数据缓冲区默认共享同一个缓冲区。也就是说，Wi-Fi 默认将数据包转发到 LWIP 作为参考。
- 在发送过程中，应用程序首先将要发送的消息复制到 LWIP 层的发送数据缓冲区，进行 TCP/IP 封装。然后将消息发送到 Wi-Fi 层的发送数据缓冲区进行 MAC 封装，最后等待发送。

参数

适当增加上述缓冲区的大小或数量，可以提高 Wi-Fi 性能，但同时，会减少应用程序的可用内存。下面我们将介绍您需要配置的参数：

接收数据方向：

- *CONFIG_ESP_WIFI_STATIC_RX_BUFFER_NUM* 该参数表示硬件层的 DMA 缓冲区数量。提高该参数将增加发送方的一次性接收吞吐量，从而提高 Wi-Fi 协议栈处理突发流量的能力。
- *CONFIG_ESP_WIFI_DYNAMIC_RX_BUFFER_NUM* 该参数表示 Wi-Fi 层中接收数据缓冲区的数量。提高该参数可以增强数据包的接收性能。该参数需要与 LWIP 层的接收数据缓冲区大小相匹配。
- *CONFIG_ESP_WIFI_RX_BA_WIN* 该参数表示接收端 AMPDU BA 窗口的大小，应配置为*CONFIG_ESP_WIFI_STATIC_RX_BUFFER_NUM* 和*CONFIG_ESP_WIFI_DYNAMIC_RX_BUFFER_NUM* 的二倍数值中较小的数值。
- *CONFIG_LWIP_TCP_WND_DEFAULT* 该参数表示 LWIP 层用于每个 TCP 流的的接收数据缓冲区大小，应配置为 WIFI_DYNAMIC_RX_BUFFER_NUM (KB) 的值，从而实现高稳定性能。同时，在有多个流的情况下，应相应降低该参数值。

发送数据方向：

- *CONFIG_ESP_WIFI_TX_BUFFER* 该参数表示发送数据缓冲区的类型，建议配置为动态缓冲区，该配置可以充分利用内存。
- *CONFIG_ESP_WIFI_DYNAMIC_TX_BUFFER_NUM* 该参数表示 Wi-Fi 层发送数据缓冲区数量。提高该参数可以增强数据包发送的性能。该参数值需要与 LWIP 层的发送数据缓冲区大小相匹配。
- *CONFIG_LWIP_TCP_SND_BUF_DEFAULT* 该参数表示 LWIP 层用于每个 TCP 流的的发送数据缓冲区大小，应配置为 WIFI_DYNAMIC_TX_BUFFER_NUM (KB) 的值，从而实现高稳定性能。在有多个流的情况下，应相应降低该参数值。

通过在 IRAM 中放置代码优化吞吐量：

备注： 上述的缓冲区大小固定为 1.6 KB。

如何配置参数

ESP32-C2 的内存由协议栈和应用程序共享。

在这里，我们给出了几种配置等级。在大多数情况下，您应根据应用程序所占用内存的大小，选择合适的等级进行参数配置。

下表中未提及的参数应设置为默认值。

等级	Iperf	默认	最小
可用内存 (KB)	37	56	84
WIFI_STATIC_RX_BUFFER_NUM	14	7	3
WIFI_DYNAMIC_RX_BUFFER_NUM	18	14	6
WIFI_DYNAMIC_TX_BUFFER_NUM	18	14	6
WIFI_RX_BA_WIN	16	12	6
TCP_SND_BUF_DEFAULT (KB)	18	14	6
TCP_WND_DEFAULT (KB)	18	14	6
LWIP_IRAM_OPTIMIZATION	13	13	0
TCP 发送数据吞吐量 (Mbit/s)	21.6	21.4	14.3
TCP 接收数据吞吐量 (Mbit/s)	19.1	17.9	12.4
UDP 发送数据吞吐量 (Mbit/s)	26.4	26.3	25.0
UDP 接收数据吞吐量 (Mbit/s)	32.3	31.5	27.7

备注： 以上结果使用红米 RM2100 路由器，在屏蔽箱中进行单流测试得出。ESP32-C2 的 CPU 为单核，频率为 120 MHz，flash 为 QIO 模式，频率为 60 MHz。

4.27.30 Wi-Fi Menuconfig

Wi-Fi 缓冲区配置

如果您要修改默认的缓冲区数量或类型，最好也了解缓冲区在数据路径中是如何分配或释放的。下图显示了发送数据方向的这一过程。

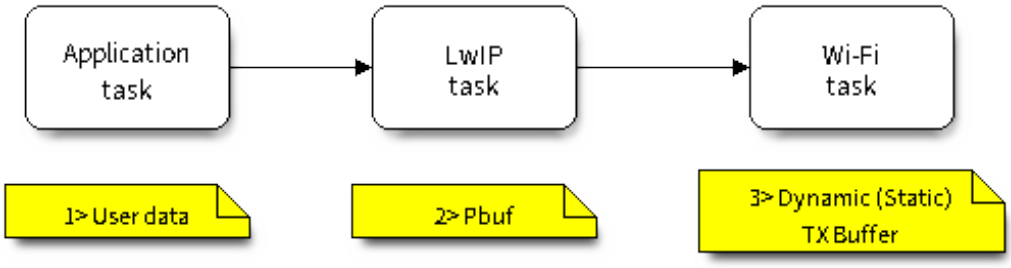


图 34: TX Buffer Allocation

描述：

- 应用程序分配需要发送的数据。
- 应用程序调用 TCPIP 或套接字相关的 API 发送用户数据。这些 API 会分配一个在 LwIP 中使用的 PBUF，并复制用户数据。
- 当 LwIP 调用 Wi-Fi API 发送 PBUF 时，Wi-Fi API 会分配一个“动态发送数据缓冲区”或“静态发送数据缓冲区”，并复制 LwIP PBUF，最后发送数据。

下图展示了如何在接收数据方向分配或释放缓冲区：

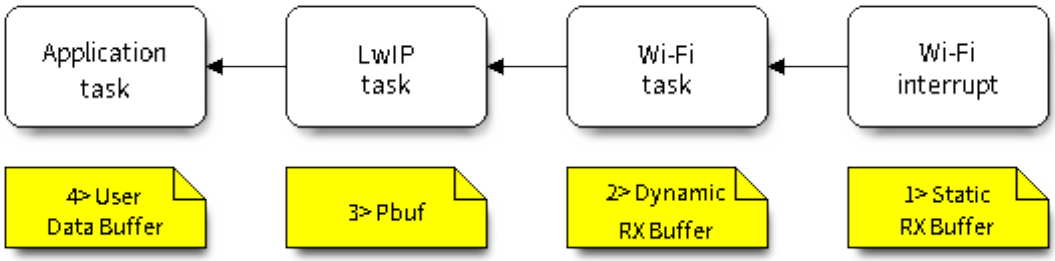


图 35: 接收数据缓冲区分配

描述：

- Wi-Fi 硬件在空中接收到数据包后，将数据包内容放到“静态接收数据缓冲区”，也就是“接收数据 DMA 缓冲区”。
- Wi-Fi 驱动程序分配一个“动态接收数据缓冲区”、复制“静态接收数据缓冲区”，并将“静态接收数据缓冲区”返回给硬件。
- Wi-Fi 驱动程序将数据包传送到上层 (LwIP)，并分配一个 PBUF 用于存放“动态接收数据缓冲区”。
- 应用程序从 LwIP 接收数据。

下表是 Wi-Fi 内部缓冲区的配置情况。

缓冲区类型	分配类型	默认	是否可配置	描述
静态接收数据缓冲区（硬件接收数据缓冲区）	静态	10 * 1600 Bytes	是	这是一种 DMA 内存，在函数 `esp_wifi_init()` 中初始化，在函数 `esp_wifi_deinit()` 中释放。该缓冲区形成硬件接收列表。当通过空中接收到一个帧时，硬件将该帧写入缓冲区，并向 CPU 发起一个中断。然后，Wi-Fi 驱动程序从缓冲区中读取内容，并将缓冲区返回到列表中。 如果应用程序希望减少 Wi-Fi 静态分配的内存，可以将该值从 10 减少到 6，从而节省 6400 Bytes 的内存。除非禁用 AMPDU 功能，否则不建议将该值降低到 6 以下。
动态接收数据缓冲区	动态	32	是	缓冲区的长度可变，取决于所接收帧的长度。当 Wi-Fi 驱动程序从“硬件接收数据缓冲区”接收到一帧时，需要从堆中分配“动态接收数据缓冲区”。在 Menuconfig 中配置的“动态接收数据缓冲区”数量用来限制未释放的“动态接收数据缓冲区”总数量。
动态发送数据缓冲区	动态	32	是	这是一种 DMA 内存，位于堆内存中。当上层 (LwIP) 向 Wi-Fi 驱动程序发送数据包时，该缓冲区首先分配一个“动态发送数据缓冲区”，并复制上层缓冲区。 动态发送数据缓冲区和静态发送数据缓冲区相互排斥。
静态发送数据缓冲区	静态	16 * 1600 Bytes	是	这是一种 DMA 内存，在函数 `esp_wifi_init()` 中初始化，在函数 `esp_wifi_deinit()` 中释放。当上层 (LwIP) 向 Wi-Fi 驱动程序发送数据包时，该缓冲区首先分配一个“静态发送数据缓冲区”，并复制上层缓冲区。 动态发送数据缓冲区和静态发送数据缓冲区相互排斥。 由于发送数据缓冲区必须是 DMA 缓冲区，所以当使能 PSRAM 时，发送数据缓冲区必须是静态的。
管理短缓冲区	动态	8	否	Wi-Fi 驱动程序的内部缓冲区。
管理长缓冲区	动态	32	否	Wi-Fi 驱动程序的内部缓冲区。
管理超长缓冲区	动态	32	否	Wi-Fi 驱动程序的内部缓冲区。

Wi-Fi NVS Flash

如果使能 Wi-Fi NVS flash，所有通过 Wi-Fi API 设置的 Wi-Fi 配置都会被存储到 flash 中，Wi-Fi 驱动程序在下次开机或重启时将自动加载这些配置。但是，应用程序可视情况禁用 Wi-Fi NVS flash，例如：其配置信息不需要存储在非易失性内存中、其配置信息已安全备份，或仅出于某些调试原因等。

Wi-Fi AMPDU

ESP32-C2 同时支持接收和发送 AMPDU，AMPDU 可以大大提高 Wi-Fi 的吞吐量。

通常，应使能 AMPDU。禁用 AMPDU 通常用于调试目的。

4.27.31 故障排除

请见*乐鑫 Wireshark 使用指南*。

乐鑫 Wireshark 使用指南

1. 概述

1.1 什么是 Wireshark？ Wireshark（原称 Ethereal）是一个网络封包分析软件。网络封包分析软件的功能是撷取网络封包，并尽可能显示出最为详细的网络封包资料。Wireshark 使用 WinPCAP 作为接口，直接与网卡进行数据报文交换。

网络封包分析软件的功能可想像成“电工技师使用电表来量测电流、电压、电阻”的工作，只是将场景移植到网络上，并将电线替换成网线。

在过去，网络封包分析软件是非常昂贵，或是专门属于营利用的软件。Wireshark 的出现改变了这一切。

在 GNU GPL 通用许可证的保障范围下，使用者可以以免费的代价取得软件与其源代码，并拥有针对其源代码修改及客制化的权利。

Wireshark 是目前全世界最广泛的网络封包分析软件之一。

1.2 Wireshark 的主要应用 下面是 Wireshark 一些应用的举例：

- 网络管理员用来解决网络问题
- 网络安全工程师用来检测安全隐患
- 开发人员用来测试协议执行情况
- 用来学习网络协议

除了上面提到的，Wireshark 还可以用在其它许多场合。

1.3 Wireshark 的特性

- 支持 UNIX 和 Windows 平台
- 在接口实时捕捉包
- 能详细显示包的详细协议信息
- 可以打开/保存捕捉的包
- 可以导入导出其他捕捉程序支持的包数据格式
- 可以通过多种方式过滤包
- 多种方式查找包
- 通过过滤以多种色彩显示包
- 创建多种统计分析
- 等等

1.4 Wireshark 的“能”与“不能”？

- **捕捉多种网络接口**
  Wireshark 可以捕捉多种网络接口类型的包，哪怕是无线局域网接口。
- **支持多种其它程序捕捉的文件**
  Wireshark 可以打开多种网络分析软件捕捉的包。
- **支持多格式输出**
  Wireshark 可以将捕捉文件输出为多种其他捕捉软件支持的格式。
- **对多种协议解码提供支持**
  Wireshark 可以支持许多协议的解码。
- **Wireshark 不是入侵检测系统**
  如果您的网络中存在任何可疑活动，Wireshark 并不会主动发出警告。不过，当您希望对这些可疑活动一探究竟时，Wireshark 可以发挥作用。
- **Wireshark 不会处理网络事务，它仅仅是“测量”（监视）网络**
  Wireshark 不会发送网络包或做其它交互性的事情（名称解析除外，但您也可以禁止解析）。

2. 如何获取 Wireshark 官网链接：https://www.wireshark.org/download.html

Wireshark 支持多种操作系统，请在下载安装文件时，注意选择与您所用操作系统匹配的安装文件。

3. 使用步骤 **本文档仅以 Linux 系统下的 Wireshark（版本号：2.2.6）为例。**

1) 启动 Wireshark

Linux 下，可编写一个 Shell 脚本，运行该文件即可启动 Wireshark 配置抓包网卡和信道。Shell 脚本如下：

```
ifconfig $1 down
iwconfig $1 mode monitor
iwconfig $1 channel $2
ifconfig $1 up
Wireshark&
```

脚本中有两个参数：`$1` 和 `$2`，分别表示网卡和信道，例如，`./xxx.sh wlan0 6`（此处，`wlan0` 即为抓包使用的网卡，后面的数字 `6` 即为 AP 或 soft-AP 所在的 channel）。

2) 运行 Shell 脚本打开 Wireshark，会出现 Wireshark 抓包开始界面

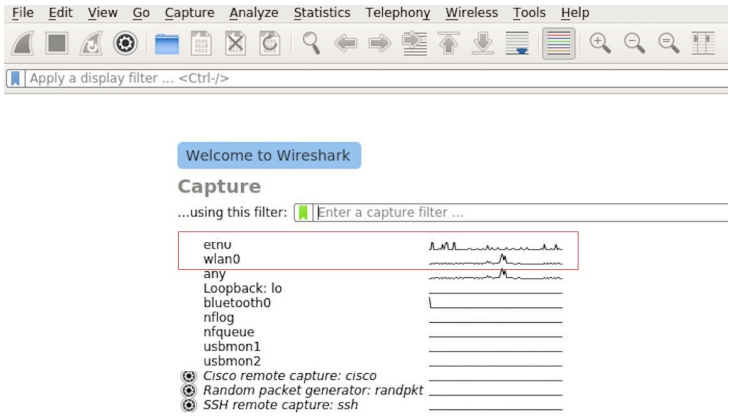


图 36: Wireshark 抓包界面

3) 选择接口，开始抓包

从上图红色框中可以看到有多个接口，第一个为本地网卡，第二个为无线网络。

可根据自己的需求选取相应的网卡，本文是以利用无线网卡抓取空中包为例进行简单说明。

双击 *wlan0* 即可开始抓包。

4) 设置过滤条件

抓包过程中会抓取到同信道所有的空中包，但其实很多都是我们不需要的，因此很多时候我们会设置抓包的过滤条件从而得到我们想要的包。

下图中红色框内即为设置 filter 的位置。

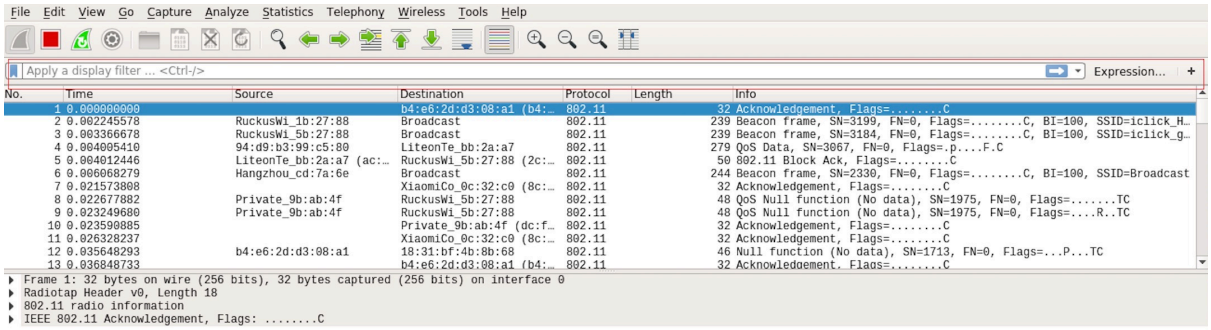


图 37: 设置 Wireshark 过滤条件

点击 *Filter* 按钮（下图的左上角蓝色按钮）会弹出 *display filter* 对话框。

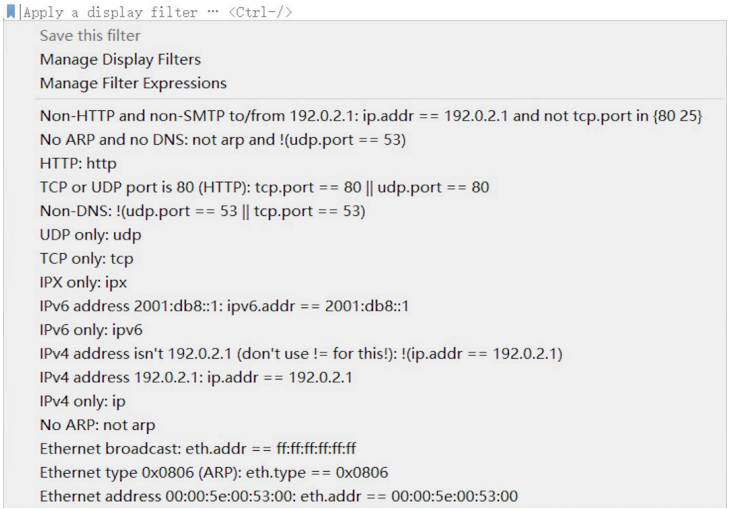


图 38: *Display Filter* 对话框

点击 *Expression* 按钮，会出现 *Filter Expression* 对话框，在此你可以根据需求进行 filter 的设置。

最直接的方法：直接在工具栏上输入过滤条件。

点击在此区域输入或修改显示的过滤字符，在输入过程中会进行语法检查。如果您输入的格式不正确，或者未输入完成，则背景显示为红色。直到您输入合法的表达式，背景会变为绿色。你可以点击下拉列表选择您先前键入的过滤字符。列表会一直保留，即使您重新启动程序。

例如：下图所示，直接输入 2 个 MAC 作为过滤条件，点击 *Apply*（即图中的蓝色箭头），则表示只抓取 2 个此 MAC 地址之间的交互的包。

5) 封包列表

若想查看包的具体的信息只需要选中要查看的包，在界面的下方会显示出包的具体的格式和包的内容。

如上图所示，我要查看第 1 个包，选中此包，图中红色框中即为包的具体内容。

6) 停止/开始包的捕捉

若要停止当前抓包，点击下图的红色按钮即可。

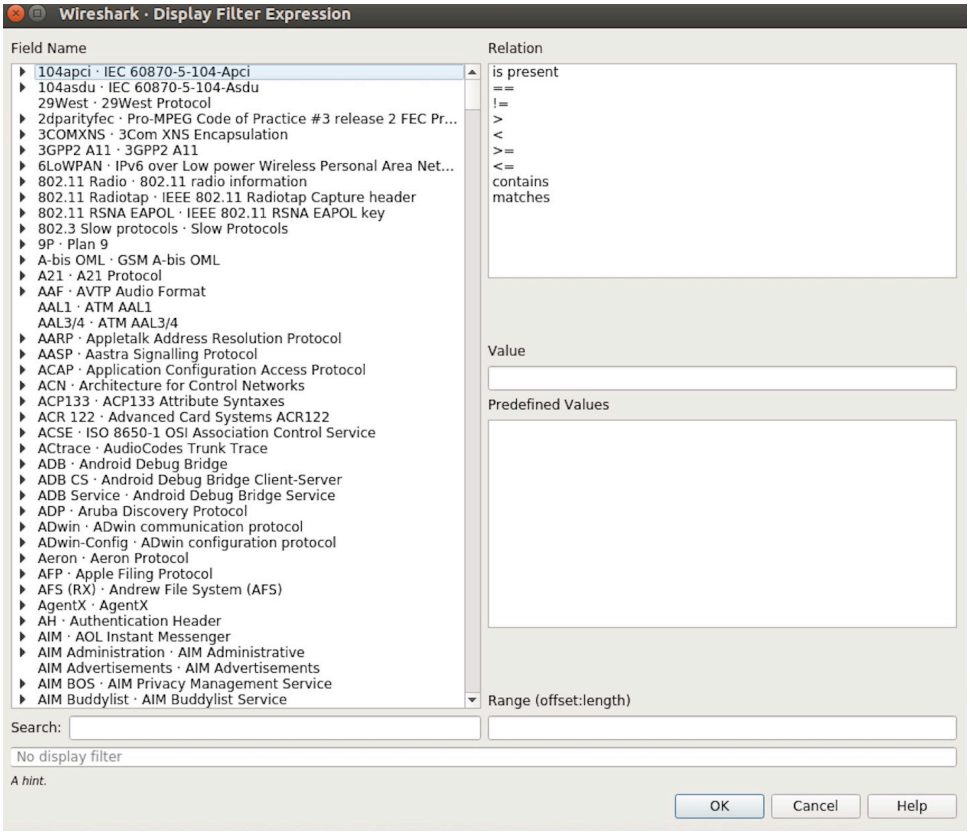


图 39: *Filter Expression* 对话框

图 40: 过滤条件工具栏

图 41: 在过滤条件工具栏中运用 MAC 地址过滤示例

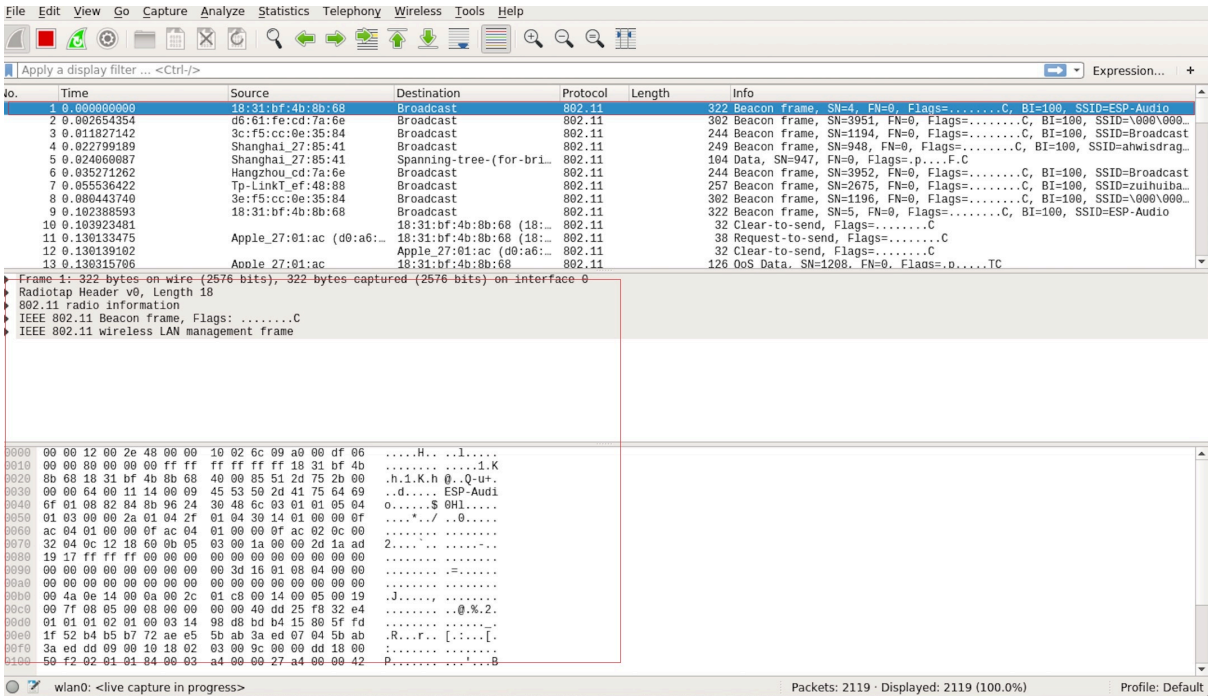


图 42: 封包列表具体信息示例

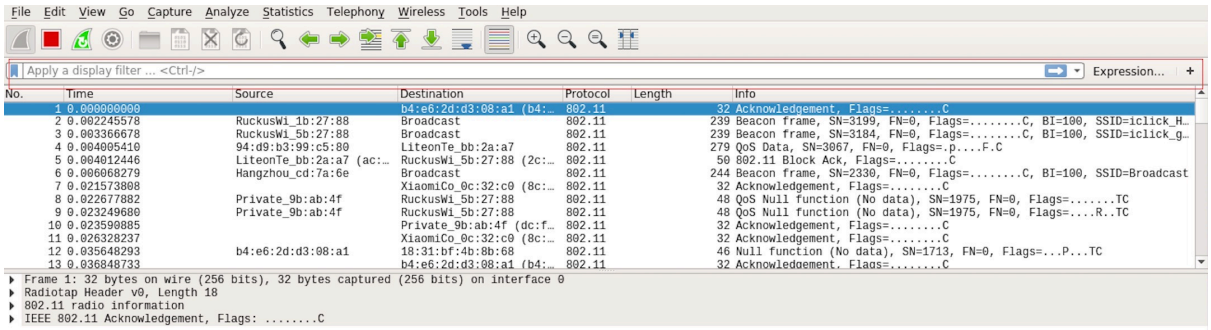


图 43: 停止包的捕捉

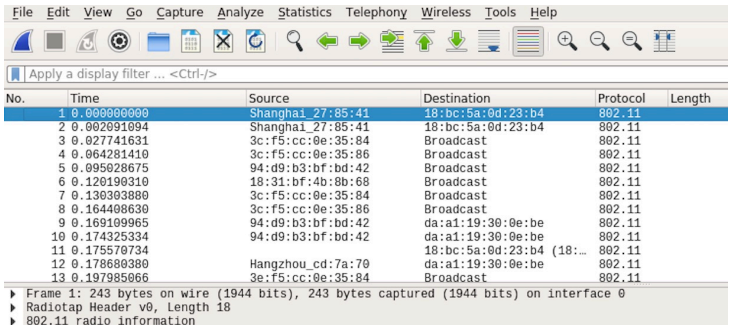


图 44: 开始或继续包的捕捉

若要重新开始抓包，点击下图左上角的蓝色按钮即可。

7) 保存当前捕捉包

Linux 下，可以通过依次点击 “File” -> “Export Packet Dissections” -> “As Plain Text File” 进行保存。

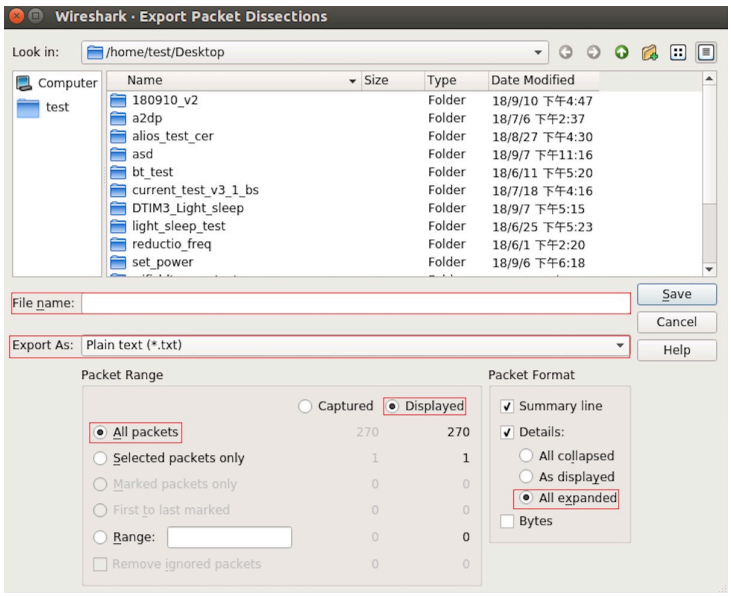


图 45: 保存捕捉包

上图中，需要注意的是，选择 *All packets*、*Displayed* 以及 *All expanded* 三项。

Wireshark 捕捉的包可以保存为其原生格式文件（libpcap），也可以保存为其他格式（如.txt 文件）供其他工具进行读取分析。

4.28 Wi-Fi Security

4.28.1 ESP32-C2 Wi-Fi Security Features

- Support for Protected Management Frames (PMF)
- Support for WPA3-Personal

In addition to traditional security methods (WEP/WPA-TKIP/WPA2-CCMP), ESP32-C2 Wi-Fi supports state-of-the-art security protocols, namely Protected Management Frames based on 802.11w standard and Wi-Fi Protected Access 3 (WPA3-Personal). Together, PMF and WPA3 provide better privacy and robustness against known attacks on traditional modes.

4.28.2 Protected Management Frames (PMF)

Introduction

In Wi-Fi, management frames such as beacons, probes, (de)authentication, (dis)association are used by non-AP stations to scan and connect to an AP. Unlike data frames, these frames are sent unencrypted. An attacker can use eavesdropping and packet injection to send spoofed (de)authentication/(dis)association frames at the right time, leading to the following attacks in case of unprotected management frame exchanges.

- DOS attack on one or all clients in the range of the attacker.
- Tearing down existing association on AP side by sending association request.
- Forcing a client to perform 4-way handshake again in case PSK is compromised in order to get PTK.
- Getting SSID of hidden network from association request.

- Launching man-in-the-middle attack by forcing clients to deauth from legitimate AP and associating to a rogue one.

PMF provides protection against these attacks by encrypting unicast management frames and providing integrity checks for broadcast management frames. These include deauthentication, disassociation and robust management frames. It also provides Secure Association (SA) teardown mechanism to prevent spoofed association/authentication frames from disconnecting already connected clients.

There are 3 types of PMF configuration modes on both station and AP side -

- PMF Optional
- PMF Required
- PMF Disabled

Depending on PMF configurations on Station and AP side, the resulting connection will behave differently. The table below summarises all possible outcomes -

STA Setting	AP Setting	Outcome
PMF Optional	PMF Optional/Required	Mgmt Frames Protected
PMF Optional	PMF Disabled	Mgmt Frames Not Protected
PMF Required	PMF Optional/Required	Mgmt Frames Protected
PMF Required	PMF Disabled	STA refuses Connection
PMF Disabled	PMF Optional/Disabled	Mgmt Frames Not Protected
PMF Disabled	PMF Required	AP refuses Connection

API & Usage

ESP32-C2 supports PMF in both Station and SoftAP mode. For both, the default mode is PMF Optional. For even higher security, PMF required mode can be enabled by setting the `required` flag in *pmf_cfg* while using the `esp_wifi_set_config()` API. This will result in the device only connecting to a PMF enabled device and rejecting others. PMF optional can be disabled using `esp_wifi_disable_pmf_config()` API. If softAP is started in WPA3 or WPA2/WPA3 mixed mode trying to disable PMF will result in error.

> **注意:** `capable` flag in *pmf_cfg* is deprecated and set to true internally. This is to take the additional security benefit of PMF whenever possible.

4.28.3 WPA3-Personal

Introduction

Wi-Fi Protected Access-3 (WPA3) is a set of enhancements to Wi-Fi access security intended to replace the current WPA2 standard. It includes new features and capabilities that offer significantly better protection against different types of attacks. It improves upon WPA2-Personal in following ways:

- WPA3 uses Simultaneous Authentication of Equals (SAE), which is password-authenticated key agreement method based on Diffie-Hellman key exchange. Unlike WPA2, the technology is resistant to offline-dictionary attack, where the attacker attempts to determine shared password based on captured 4-way handshake without any further network interaction.
- Disallows outdated protocols such as TKIP, which is susceptible to simple attacks like MIC key recovery attack.
- Mandates Protected Management Frames (PMF), which provides protection for unicast and multicast robust management frames which include Disassoc and Deauth frames. This means that the attacker cannot disrupt an established WPA3 session by sending forged Assoc frames to the AP or Deauth/Disassoc frames to the Station.
- Provides forward secrecy, which means the captured data cannot be decrypted even if password is compromised after data transmission.

Please refer to Security section of Wi-Fi Alliance ' s official website for further details.

Setting up WPA3 with ESP32-C2

In IDF Menuconfig under Wi-Fi component, a config option "Enable WPA3-Personal" is provided to Enable/Disable WPA3 for station. By default it is kept enabled, if disabled ESP32-C2 will not be able to establish a WPA3 connection. Also under WI-FI component a config option "ESP_WIFI_SOFTAP_SAE_SUPPORT" is provided to Enable/Disable WPA3 for softAP. Additionally, since PMF is mandated by WPA3 protocol, PMF Mode Optional is set by default for station and softAP. PMF Required can be configured using WiFi config. For WPA3 softAP, PMF required is mandatory and will be configured and stored in NVS implicitly if not specified by user.

Refer to *Protected Management Frames (PMF)* on how to set this mode.

After configuring all required settings for WPA3-Personal station, application developers need not worry about the underlying security mode of the AP. WPA3-Personal is now the highest supported protocol in terms of security, so it will be automatically selected for the connection whenever available. For example, if an AP is configured to be in WPA3 Transition Mode, where it will advertise as both WPA2 and WPA3 capable, Station will choose WPA3 for the connection with above settings. Note that Wi-Fi stack size requirement will increase 3kB when "Enable WPA3-Personal" is used.

After configuring all required setting for WPA3-Personal softAP, application developers have to set `WIFI_AUTH_WPA3_PSK` as WiFi config authmode to start AP in softAP. SoftAP can be also configured to use `WIFI_AUTH_WPA2_WPA3_PSK` mixed mode. Note that flash size will be increased by 6kB after enabling "ESP_WIFI_SOFTAP_SAE_SUPPORT" .

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

低功耗蓝牙

Bluedroid

以下 Bluedroid 宏、类型和函数已被重命名：

- bt/host/bluedroid/api/include/api/esp_gap_ble_api.h
  - *esp_gap_ble_cb_event_t* 中：
    - ESP_GAP_BLE_SET_PREFERED_DEFAULT_PHY_COMPLETE_EVT 改名为 ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT
    - ESP_GAP_BLE_SET_PREFERED_PHY_COMPLETE_EVT 改名为 ESP_GAP_BLE_SET_PREFERRED_PHY_COMPLETE_EVT
    - ESP_GAP_BLE_CHANNEL_SELETE_ALGORITHM_EVT 改名为 ESP_GAP_BLE_CHANNEL_SELECT_ALGORITHM_EVT
  - esp_ble_wl_opration_t 改名为*esp_ble_wl_operation_t*
  - esp_ble_gap_cb_param_t.pkt_data_lenth_cmpl 改名为 pkt_data_length_cmpl
  - esp_ble_gap_cb_param_t.update_whitelist_cmpl.wl_opration 改名为 wl_operation
  - esp_ble_gap_set_prefered_default_phy 改名为*esp_ble_gap_set_preferred_default_phy()*
  - esp_ble_gap_set_prefered_phy 改名为*esp_ble_gap_set_preferred_phy()*
- bt/host/bluedroid/api/include/api/esp_gatt_defs.h
  - *esp_gatt_status_t* 中：
    - ESP_GATT_ENCRYPED_MITM 改名为 ESP_GATT_ENCRYPTED_MITM
    - ESP_GATT_ENCRYPED_NO_MITM 改名为 ESP_GATT_ENCRYPTED_NO_MITM

Nimble

以下 Nimble API 已被移除：

- bt/host/nimble/esp-hci/include/esp_nimble_hci.h
  - 移除 `esp_err_t esp_nimble_hci_and_controller_init(void)`
    - 控制器初始化、使能以及 HCI 初始化的调用已经被移到 *nimble_port_init* 中。可直接删除该函数。
  - 移除 `esp_err_t esp_nimble_hci_and_controller_deinit(void)`
    - 控制器去初始化、禁用以及 HCI 去初始化的调用已经被移到 *nimble_port_deinit* 中。可直接删除该函数。

ESP-BLE-MESH

以下 ESP-BLE-MESH 宏已被重命名：

- bt/esp_ble_mesh/api/esp_ble_mesh_defs.h
  - `esp_ble_mesh_prov_cb_event_t` 中:
    - `ESP_BLE_MESH_PROVISIONER_DRIECT_ERASE_SETTINGS_COMP_EVT` 改名为 `ESP_BLE_MESH_PROVISIONER_DIRECT_ERASE_SETTINGS_COMP_EVT`

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 请参考将链接器脚本片段文件语法迁移至 *ESP-IDF v5.0 适应版本* 对 v3.x 的语法进行更新。

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中被添加至每个组件中，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。如果您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。若需要将您的代码从 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

上述变化主要影响到使用 <inttypes.h> 提供的类型来格式化字符串的代码。请使用 PRIi32、PRIxx 等占位符来分别替换 %i、%x 等。

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，因此该选项已被移除。

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，实参将被强制转换为 `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。如您在注册以太网/IP 事件处理程序之前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，您需要在实例化 SPI 以太网 MAC 之前，使用*spi_bus_add_device()* 手动分配 SPI 设备。

现在，由于 SPI 设备已在内部分配，您无需再调用*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*。

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

- 您只需用 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。
- 您还需用 #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

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 "\n", IP2STR(&event->ip_info.ip));
```

不建议使用下述方式：

```
ESP_LOGI(TAG, "got ip:%s\n", 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。

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。

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。

LEDC

删除/弃用项目	替代	备注
ledc_timer_config_t 中的 bit_num	*ledc_timer_config_t::duty_resolution*	设置占空比分辨率。

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*。

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_color_rgb_endian_t* 来描述 RGB 颜色的排列顺序。

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。

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, 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, 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 IDF Component Registry 的组件 以下组件已经从 ESP-IDF 中迁出至 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 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。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，我们将协助您进行代码迁移。

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，您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);
```

(下页继续)

(续上页)

```
        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 被提取至 `esp_rom_spiflash.h`。尽管这不能算作重大变更，我们强烈建议您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，以下函数也被重命名：

- `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，您可以直接将实际时钟频率值传递给 flash 配置结构。下为配置 80MHz 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` 配置选项均被删除，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。您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，现已被包含至 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）已被更新为私有 API。因此，原先的包含路径 #include "ESP32-C2/clk.h" 和 #include "esp_clk.h" 已被移除。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）已被更新为私有 API。因此，原先的包含路径 #include "ESP32-C2/cache_err_int.h" 已被移除。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() 已被移除。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() 已被移除，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，并解释您需要此函数的原因。

断电 断电 API（即以 esp_brownout 为前缀的函数、类型或宏）已被更新为私有 API。因此，原先的包含路径 #include "brownout.h" 已被移除。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）已被更新为私有 API。因此，原先的包含路径 #include "trax.h" 已被移除。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 相关头文件已被移除（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-C2/`` （新的包含路径为 ``ESP32-C2/rom/*.h）。

esp_hw_support

- 头文件 soc/cpu.h 及弃用的 CPU util 函数都已被移除。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 已被移除。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 已被移除。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）已被更新为私有 API。因此，原先的包含路径 #include "esp_panic.h" 已被移除。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 被扩展为 uint16_t。

PSRAM

- 针对特定目标的头文件 spiram.h 及头文件 esp_spiram.h 已被移除，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 已被移除，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，后者总是被设置为 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，因此已被移除。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 已被弃用。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）已被更新为私有 API。因此，原先的包含路径 #include "eh_frame_parser.h" 已被移除。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）已被更新为私有 API。因此，原先的包含路径 `#include "esp_int_wdt.h"` 已被移除。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` 也已被移除。
- 头文件 `soc/soc_memory_types.h` 已被弃用。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，现在已被移除，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` 已被重新命名为 `ESP_IMAGE_SPI_SPEED_DIV_1`。
- `ESP_IMAGE_SPI_SPEED_40M` 已被重新命名为 `ESP_IMAGE_SPI_SPEED_DIV_2`。
- `ESP_IMAGE_SPI_SPEED_26M` 已被重新命名为 `ESP_IMAGE_SPI_SPEED_DIV_3`。
- `ESP_IMAGE_SPI_SPEED_20M` 已被重新命名为 `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` 被重新命名为*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。该选项已被移除，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()` 已被移除，请使用 `portSET_INTERRUPT_MASK_FROM_ISR()` 宏。
- `portEXIT_CRITICAL_NESTED()` 已被移除，请使用 `portCLEAR_INTERRUPT_MASK_FROM_ISR()` 宏。
- `vPortCPUInitializeMutex()` 已被移除，请使用 `spinlock_initialize()` 函数。
- `vPortCPUAcquireMutex()` 已被移除，请使用 `spinlock_acquire()` 函数。
- `vPortCPUAcquireMutexTimeout()` 已被移除，请使用 `spinlock_acquire()` 函数。
- `vPortCPUReleaseMutex()` 已被移除，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()`* 已被弃用，请分别使用 *`esp_app_get_description()`* 和 *`esp_app_get_elf_sha256()`* 函数来代替。这些函数已被移至新组件 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_ESP32C2_REV_MIN* 和 < `CONFIG_ESP32C2_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_ESP32C2_REV_MIN* 和 < `CONFIG_ESP32C2_REV_MAX_FULL` 时，应用程序才能成功更新。

工具

IDF 监视器 IDF 监视器在波特率方面的改动如下：

- 目前，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。若您需要将代码从 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 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 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 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() 已被弃用。您可以使用*esp_vfs_fat_sdcard_unmount()* 代替。此接口在更早的 IDF 版本中已被弃用，但是尚未添加弃用警告。自 IDF v5.1 起，调用这个 esp_vfs_fat_sdmmc_unmount() 接口将会产生 deprecation 警告。

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

电源管理

- `esp_pm_config_esp32xx_t` 已被弃用，应使用 `esp_pm_config_t` 替代。
- `esp32xx/pm.h` 已被弃用，应使用 `esp_pm.h` 替代。

Chapter 6

Libraries and Frameworks

6.1 Cloud Frameworks

ESP32-C2 supports multiple cloud frameworks using agents built on top of ESP-IDF. Here are the pointers to various supported cloud frameworks' agents and examples:

6.1.1 ESP RainMaker

ESP RainMaker is a complete solution for accelerated AIoT development. ESP RainMaker on GitHub.

6.1.2 AWS IoT

https://github.com/espressif/esp-aws-iot is an open source repository for ESP32-C2 based on Amazon Web Services' aws-iot-device-sdk-embedded-C.

6.1.3 Azure IoT

https://github.com/espressif/esp-azure is an open source repository for ESP32-C2 based on Microsoft Azure' s azure-iot-sdk-c SDK.

6.1.4 Google IoT Core

https://github.com/espressif/esp-google-iot is an open source repository for ESP32-C2 based on Google' s iot-device-sdk-embedded-c SDK.

6.1.5 Aliyun IoT

https://github.com/espressif/esp-aliyun is an open source repository for ESP32-C2 based on Aliyun' s iotkit-embedded SDK.

6.1.6 Joylink IoT

https://github.com/espressif/esp-joylink is an open source repository for ESP32-C2 based on Joylink' s joylink_dev_sdk SDK.

6.1.7 Tencent IoT

https://github.com/espressif/esp-welink is an open source repository for ESP32-C2 based on Tencent's welink SDK.

6.1.8 Tencentyun IoT

https://github.com/espressif/esp-qcloud is an open source repository for ESP32-C2 based on Tencentyun's qcloud-iot-sdk-embedded-c SDK.

6.1.9 Baidu IoT

https://github.com/espressif/esp-baidu-iot is an open source repository for ESP32-C2 based on Baidu's iot-sdk-c SDK.

6.2 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。

6.2.1 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。

6.2.2 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。

6.2.3 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)
- 无线脉冲响应 (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。

6.2.4 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。

6.2.5 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。

6.2.6 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。

6.2.7 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。

6.2.8 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。

6.2.9 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。

6.2.10 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 7

Contributions Guide

We welcome contributions to the esp-idf project!

7.1 How to Contribute

Contributions to esp-idf - fixing bugs, adding features, adding documentation - are welcome. We accept contributions via Github Pull Requests.

7.2 Before Contributing

Before sending us a Pull Request, please consider this list of points:

- Is the contribution entirely your own work, or already licensed under an Apache License 2.0 compatible Open Source License? If not then we unfortunately cannot accept it. Please check the *Copyright Header Guide* for additional information.
- Does any new code conform to the esp-idf *Style Guide*?
- Have you installed the *pre-commit hook* for esp-idf project?
- Does the code documentation follow requirements in *Documenting Code*?
- Is the code adequately commented for people to understand how it is structured?
- Is there documentation or examples that go with code contributions? There are additional suggestions for writing good examples in examples readme.
- Are comments and documentation written in clear English, with no spelling or grammar errors?
- Example contributions are also welcome. Please check the 创建示例项目 guide for these.
- If the contribution contains multiple commits, are they grouped together into logical changes (one major change per pull request)? Are any commits with names like "fixed typo" squashed into previous commits?
- If you're unsure about any of these points, please open the Pull Request anyhow and then ask us for feedback.

7.3 Pull Request Process

After you open the Pull Request, there will probably be some discussion in the comments field of the request itself.

Once the Pull Request is ready to merge, it will first be merged into our internal git system for in-house automated testing.

If this process passes, it will be merged onto the public github repository.

7.4 Legal Part

Before a contribution can be accepted, you will need to sign our *Contributor Agreement*. You will be prompted for this automatically as part of the Pull Request process.

7.5 Related Documents

7.5.1 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. Don't 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. Don't begin or end a function with an empty line.

```
void function1()
{
    do_one_thing();
    do_another_thing();
                                    // INCORRECT, don't place empty line here
}
                                    // place empty line here
void function2()
{
                                    // INCORRECT, don't use an empty line here
    int var = 0;
    while (var < SOME_CONSTANT) {
        do_stuff(&var);
```

(下页继续)

(续上页)

```
    }
}
```

The maximum line length is 120 characters as long as it doesn't 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)
{

}
```

(下页继续)

(续上页)

```
// 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.

- Don't 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. Don't use `#if 0 ... #endif` or comments to store code snippets which you may need in the future.
- Don't 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's possible to run `dos2unix FILENAME` and then run `git commit --amend`

Formatting Your Code You can use `astyle` program to format your code according to the above recommendations.

If you are writing a file from scratch, or doing a complete rewrite, feel free to re-format the entire file. If you are changing a small portion of file, don't re-format the code you didn't change. This will help others when they review your changes.

To re-format a file, run:

```
tools/format.sh components/my_component/file.c
```

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's 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's 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's 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

- Don't 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;
}
```

(下页继续)

(续上页)

```
SpaceShuttle::SpaceShuttle() : SpaceShip(7)
{
    // doing further initialization
}

Sojuz::Sojuz() : SpaceShip(3)
{
    // 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).
- Don't 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: *Documenting Code*.

Structure

To be written.

Language Features

To be written.

7.5.2 Install pre-commit Hook for ESP-IDF Project

Required Dependency

Python 3.7.* or above. This is our recommended python version for IDF developers.

If you still have python versions not compatible, update your python versions before installing the pre-commit hook.

Install pre-commit

Run `pip install pre-commit`

Install pre-commit hook

1. Go to the IDF Project Directory
2. Run `pre-commit install --allow-missing-config`. Install hook by this approach will let you commit successfully even in branches without the `.pre-commit-config.yaml`
3. pre-commit hook will run automatically when you're running `git commit` command

Uninstall pre-commit

Run `pre-commit uninstall`

What's More?

For detailed usage, please refer to the documentation of pre-commit.

Common Problems For Windows Users

`/usr/bin/env: python: Permission denied.`

> If you're in Git Bash, please check the python executable location by run `which python`.
>
> If the executable is under `~/AppData/Local/Microsoft/WindowsApps/`, then it's a link to Windows AppStore, not a real one.
>
> Please install python manually and update this in your `PATH` environment variable.

Your %USERPROFILE% contains non-ASCII characters

> `pre-commit` may fail when initializing an environment for a particular hook when the path of `pre-commit`'s cache contains non-ASCII characters. The solution is to set `PRE_COMMIT_HOME` to a path containing only standard characters before running 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"`

7.5.3 Documenting Code

The purpose of this description is to provide quick summary on documentation style used in espressif/esp-idf repository and how to add new documentation.

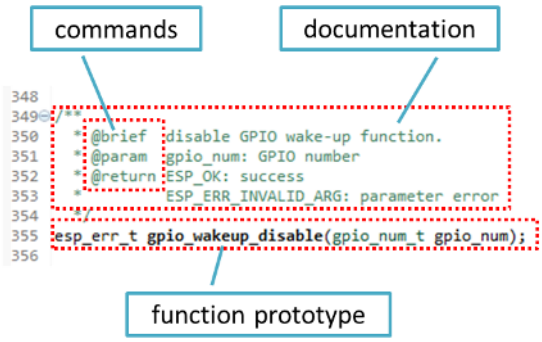
Introduction

When documenting code for this repository, please follow Doxygen style. You are doing it by inserting special commands, for instance `@param`, into standard comments blocks, for example:

```
/**
 * @param ratio this is oxygen to air ratio
 */
```

Doxygen is phrasing the code, extracting the commands together with subsequent text, and building documentation out of it.

Typical comment block, that contains documentation of a function, looks like below.

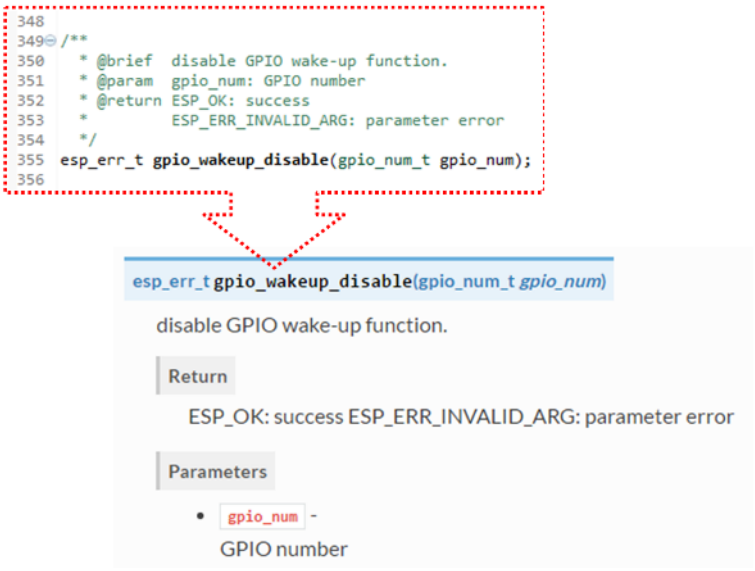


Doxygen supports couple of formatting styles. It also gives you great flexibility on level of details to include in documentation. To get familiar with available features, please check data rich and very well organized Doxygen Manual.

Why we need it?

The ultimate goal is to ensure that all the code is consistently documented, so we can use tools like Sphinx and Breathe to aid preparation and automatic updates of API documentation when the code changes.

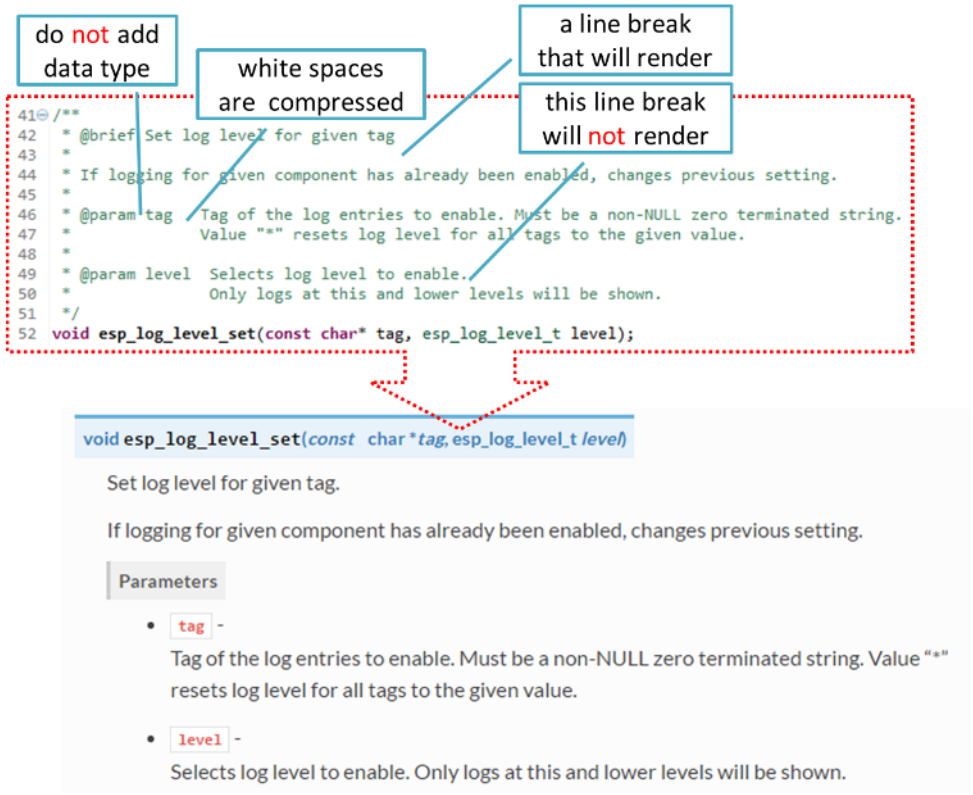
With these tools the above piece of code renders like below:



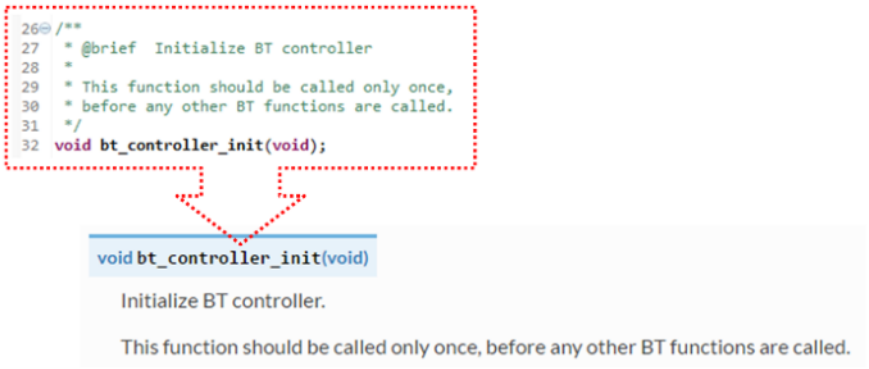
Go for it!

When writing code for this repository, please follow guidelines below.

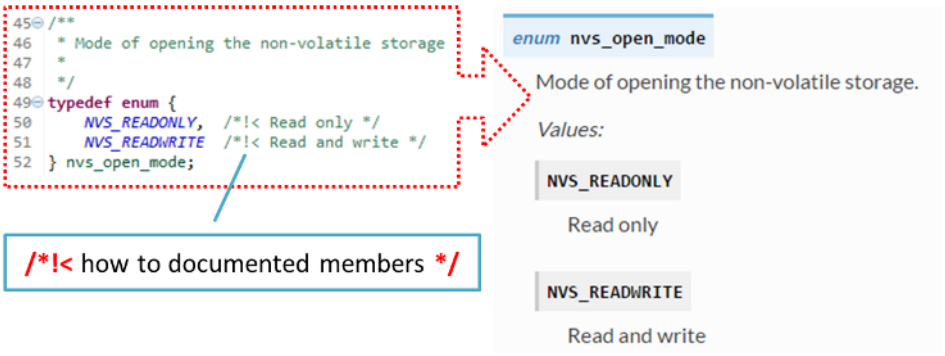
1. Document all building blocks of code: functions, structs, typedefs, enums, macros, etc. Provide enough information about purpose, functionality and limitations of documented items, as you would like to see them documented when reading the code by others.
2. Documentation of function should describe what this function does. If it accepts input parameters and returns some value, all of them should be explained.
3. Do not add a data type before parameter or any other characters besides spaces. All spaces and line breaks are compressed into a single space. If you like to break a line, then break it twice.



4. If function has void input or does not return any value, then skip `@param` or `@return`



5. When documenting a `define` as well as members of a `struct` or `enum`, place specific comment like below after each member.



6. To provide well formatted lists, break the line after command (like `@return` in example below).

```
*
* @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. Overview of functionality of documented header file, or group of files that make a library, should be placed in a separate `README.rst` file of the same directory. If this directory contains header files for different APIs, then the file name should be `apiname-readme.rst`.

Go one extra mile

Here are a couple of tips on how you can make your documentation even better and more useful to the reader and writer.

When writing codes, please follow the guidelines below:

1. Add code snippets to illustrate implementation. To do so, enclose snippet using `@code{c}` and `@endcode` commands.

```
*
* @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.
```

(下页继续)

(续上页)

```
* @endcode
*
```

The code snippet should be enclosed in a comment block of the function that it illustrates.

2. To highlight some important information use command `@attention` or `@note`.

```
*
* @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.
*
```

Above example also shows how to use a numbered list.

3. To provide common description to a group of similar functions, enclose them using `/**@{*/` and `/**@}*/` markup commands:

```
/**@{*/
/**
 * @brief common description of similar functions
 *
 */
void  first_similar_function (void);
void second_similar_function (void);
/**@}*/
```

For practical example see nvs_flash/include/nvs.h.

4. You may want to go even further and skip some code like repetitive defines or enumerations. In such case, enclose the code within `/** @cond */` and `/** @endcond */` commands. Example of such implementation is provided in driver/gpio/include/driver/gpio.h.
5. Use markdown to make your documentation even more readable. You will add headers, links, tables and more.

```
*
* [ESP32-C2 Technical Reference Manual](https://www.espressif.com/sites/
↪default/files/documentation/esp8684_technical_reference_manual_en.pdf)
*
```

备注： Code snippets, notes, links, etc. will not make it to the documentation, if not enclosed in a comment block associated with one of documented objects.

---

6. Prepare one or more complete code examples together with description. Place description to a separate file `README.md` in specific folder of examples directory.

Standardize Document Format

When it comes to text, please follow guidelines below to provide well formatted Markdown (.md) or reST (.rst) documents.

1. Please ensure that one paragraph is written in one line. Don't break lines like below. Breaking lines to enhance readability is only suitable for writing codes. To make the text easier to read, it is recommended to place an empty line to separate the paragraph.

2. Please make the line number of CN and EN documents consistent like below. The benefit of this approach is that it can save time for both writers and translators. When non-bilingual writers need to update text, they only need to update the same line in the corresponding CN or EN document. For translators, if documents are updated in English, then translators can quickly locate where to update in the corresponding CN document later. Besides, by comparing the total number of lines in EN and CN documents, you can quickly find out whether the CN version lags behind the EN version.

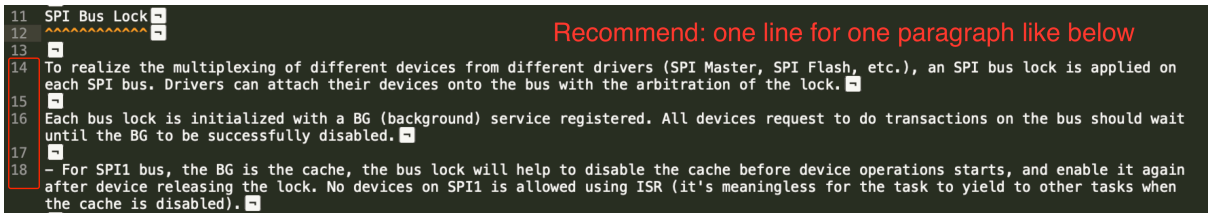


图 1: One line for one paragraph (click to enlarge)

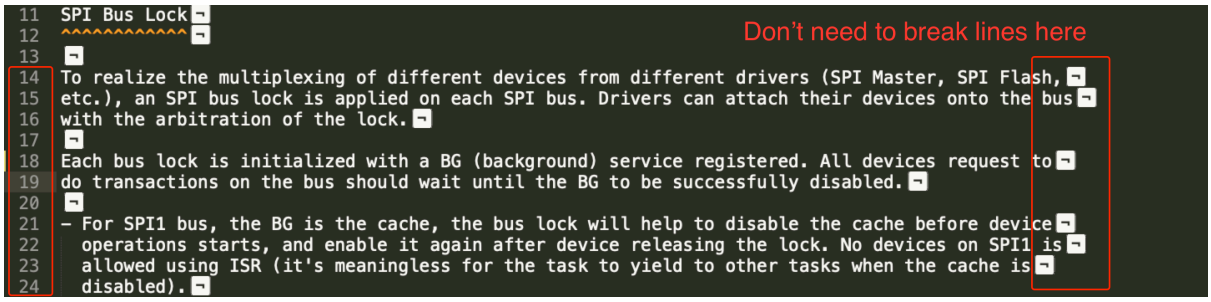


图 2: No line breaks within the same paragraph (click to enlarge)

图 3: Keep the line number for EN and CN documents consistent (click to enlarge)

Building Documentation

The documentation is built with the *esp-docs* Python package, which is a wrapper around Sphinx

To install it simply do:

```
pip install esp-docs
```

After a successful install then the documentation can be built from the docs folder with:

```
build-docs build
```

or for specific target and language with:

```
build-docs -t esp32 -l en build
```

For more in-depth documentation about *esp-docs* features please see the documentation at esp-docs.

Wrap up

We love good code that is doing cool things. We love it even better, if it is well documented, so we can quickly make it run and also do the cool things.

Go ahead, contribute your code and documentation!

Related Documents

- *API Documentation Template*

7.5.4 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：

- 示例项目的名字（包括 `README.md` 中）应使用 `example`，而不要写 “demo”，“test” 等词汇。
- 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。

7.5.5 API Documentation Template

备注： *INSTRUCTIONS*

1. Use this file (docs/en/api-reference/template.rst) as a template to document API.
2. Change the file name to the name of the header file that represents documented API.
3. Include respective files with descriptions from the API folder using `..include::`
   - README.rst
   - example.rst
   - ...
4. Optionally provide description right in this file.
5. Once done, remove all instructions like this one and any superfluous headers.

Overview

备注： *INSTRUCTIONS*

1. Provide overview where and how this API may be used.
2. Where applicable include code snippets to illustrate functionality of particular functions.
3. To distinguish between sections, use the following heading levels:
   - # with overline, for parts
   - * with overline, for chapters
   - =, for sections
   - -, for subsections
   - ^, for subsubsections
   - ", for paragraphs

Application Example

备注： *INSTRUCTIONS*

1. Prepare one or more practical examples to demonstrate functionality of this API.
2. Each example should follow pattern of projects located in `esp-idf/examples/` folder.
3. Place example in this folder complete with `README.md` file.
4. Provide overview of demonstrated functionality in `README.md`.
5. With good overview reader should be able to understand what example does without opening the source code.
6. Depending on complexity of example, break down description of code into parts and provide overview of functionality of each part.
7. Include flow diagram and screenshots of application output if applicable.
8. Finally add in this section synopsis of each example together with link to respective folder in `esp-idf/examples/`.

API Reference

备注： *INSTRUCTIONS*

1. This repository provides for automatic update of API reference documentation using *code markup retrieved by Doxygen from header files*.
2. Update is done on each documentation build by invoking Sphinx extension *:esp_extensions/run_doxygen.py* for all header files listed in the `INPUT` statement of docs/doxygen/Doxyfile.
3. Each line of the `INPUT` statement (other than a comment that begins with `##`) contains a path to header file `*.h` that will be used to generate corresponding `*.inc` files:

```
##
## Wi-Fi - API Reference
##
../components/esp32/include/esp_wifi.h \
../components/esp32/include/esp_smartconfig.h \
```

4. When the headers are expanded, any macros defined by default in `sdkconfig.h` as well as any macros defined in SOC-specific `include/soc/*_caps.h` headers will be expanded. This allows the headers to include/exclude material based on the `IDF_TARGET` value.
5. The `*.inc` files contain formatted reference of API members generated automatically on each documentation build. All `*.inc` files are placed in Sphinx `_build` directory. To see directives generated for e.g. `esp_wifi.h`, run `python gen-dxd.py esp32/include/esp_wifi.h`.
6. To show contents of `*.inc` file in documentation, include it as follows:

```
.. include-build-file:: inc/esp_wifi.inc
```

   For example see docs/en/api-reference/network/esp_wifi.rst
7. Optionally, rather that using `*.inc` files, you may want to describe API in you own way. See docs/en/api-reference/storage/fatfs.rst for example.
   Below is the list of common `.. doxygen...::` directives:
   - Functions - `.. doxygenfunction:: name_of_function`
   - Unions - `.. doxygenunion:: name_of_union`
   - Structures - `.. doxygenstruct:: name_of_structure` together with `:members:`
   - Macros - `.. doxygendefine:: name_of_define`
   - Type Definitions - `.. doxygentypedef:: name_of_type`
   - Enumerations - `.. doxygenenum:: name_of_enumeration`

   See Breathe documentation for additional information.
   To provide a link to header file, use the *link custom role* directive as follows:

```
* :component_file:`path_to/header_file.h`
```

8. In any case, to generate API reference, the file docs/doxygen/Doxyfile should be updated with paths to `*.h` headers that are being documented.
9. When changes are committed and documentation is build, check how this section has been rendered. *Correct annotations* in respective header files, if required.

7.5.6 Contributor Agreement

Individual Contributor Non-Exclusive License Agreement including the Traditional Patent License OPTION

Thank you for your interest in contributing to this Espressif project hosted on GitHub ("We" or "Us").

The purpose of this contributor agreement ("Agreement") is to clarify and document the rights granted by contributors to Us. To make this document effective, please follow the instructions in the *Contributions Guide*.

1. DEFINITIONS **"You"** means the Individual Copyright owner who submits a Contribution to Us. If You are an employee and submit the Contribution as part of your employment, You have had Your employer approve this Agreement or sign the Entity version of this document.

"Contribution" means any original work of authorship (software and/or documentation) including any modifications or additions to an existing work, Submitted by You to Us, in which You own the Copyright. If You do not own the Copyright in the entire work of authorship, please contact Us by submitting a comment on GitHub.

"Copyright" means all rights protecting works of authorship owned or controlled by You, including copyright, moral and neighboring rights, as appropriate, for the full term of their existence including any extensions by You.

"Material" means the software or documentation made available by Us to third parties. When this Agreement covers more than one software project, the Material means the software or documentation to which the Contribution was Submitted. After You Submit the Contribution, it may be included in the Material.

"Submit" means any form of physical, electronic, or written communication sent to Us, including but not limited to electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, Us, but excluding communication that is conspicuously marked or otherwise designated in writing by You as "Not a Contribution."

"Submission Date" means the date You Submit a Contribution to Us.

"Documentation" means any non-software portion of a Contribution.

2. LICENSE GRANT 2.1 Copyright License to Us

Subject to the terms and conditions of this Agreement, You hereby grant to Us a worldwide, royalty-free, NON-exclusive, perpetual and irrevocable license, with the right to transfer an unlimited number of non-exclusive licenses or to grant sublicenses to third parties, under the Copyright covering the Contribution to use the Contribution by all means, including, but not limited to:

- to publish the Contribution,
- to modify the Contribution, to prepare derivative works based upon or containing the Contribution and to combine the Contribution with other software code,
- to reproduce the Contribution in original or modified form,
- to distribute, to make the Contribution available to the public, display and publicly perform the Contribution in original or modified form.

2.2 Moral Rights remain unaffected to the extent they are recognized and not waivable by applicable law. Notwithstanding, You may add your name in the header of the source code files of Your Contribution and We will respect this attribution when using Your Contribution.

3. PATENTS 3.1 Patent License

Subject to the terms and conditions of this Agreement You hereby grant to us a worldwide, royalty-free, non-exclusive, perpetual and irrevocable (except as stated in Section 3.2) patent license, with the right to transfer an unlimited number of non-exclusive licenses or to grant sublicenses to third parties, to make, have made, use, sell, offer for sale, import and otherwise transfer the Contribution and the Contribution in combination with the Material (and portions of such combination). This license applies to all patents owned or controlled by You, whether already acquired or hereafter acquired, that would be infringed by making, having made, using, selling, offering for sale, importing or otherwise transferring of Your Contribution(s) alone or by combination of Your Contribution(s) with the Material.

3.2 Revocation of Patent License

You reserve the right to revoke the patent license stated in section 3.1 if we make any infringement claim that is targeted at your Contribution and not asserted for a Defensive Purpose. An assertion of claims of the Patents shall be considered for a "Defensive Purpose" if the claims are asserted against an entity that has filed, maintained, threatened, or voluntarily participated in a patent infringement lawsuit against Us or any of Our licensees.

4. DISCLAIMER THE CONTRIBUTION IS PROVIDED "AS IS". MORE PARTICULARLY, ALL EXPRESS OR IMPLIED WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED BY YOU TO US AND BY US TO YOU. TO THE EXTENT THAT ANY SUCH WARRANTIES CANNOT BE DISCLAIMED, SUCH WARRANTY IS LIMITED IN DURATION TO THE MINIMUM PERIOD PERMITTED BY LAW.

5. Consequential Damage Waiver TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL YOU OR US BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF ANTICIPATED SAVINGS, LOSS OF DATA, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL AND EXEMPLARY DAMAGES ARISING OUT OF THIS AGREEMENT REGARDLESS OF THE LEGAL OR EQUITABLE THEORY (CONTRACT, TORT OR OTHERWISE) UPON WHICH THE CLAIM IS BASED.

6. Approximation of Disclaimer and Damage Waiver IF THE DISCLAIMER AND DAMAGE WAIVER MENTIONED IN SECTION 4 AND SECTION 5 CANNOT BE GIVEN LEGAL EFFECT UNDER APPLICABLE LOCAL LAW, REVIEWING COURTS SHALL APPLY LOCAL LAW THAT MOST CLOSELY APPROXIMATES AN ABSOLUTE WAIVER OF ALL CIVIL LIABILITY IN CONNECTION WITH THE CONTRIBUTION.

7. Term 7.1 This Agreement shall come into effect upon Your acceptance of the terms and conditions.

7.2 In the event of a termination of this Agreement Sections 4, 5, 6, 7 and 8 shall survive such termination and shall remain in full force thereafter. For the avoidance of doubt, Contributions that are already licensed under a free and open source license at the date of the termination shall remain in full force after the termination of this Agreement.

8. Miscellaneous 8.1 This Agreement and all disputes, claims, actions, suits or other proceedings arising out of this agreement or relating in any way to it shall be governed by the laws of People's Republic of China excluding its private international law provisions.

8.2 This Agreement sets out the entire agreement between You and Us for Your Contributions to Us and overrides all other agreements or understandings.

8.3 If any provision of this Agreement is found void and unenforceable, such provision will be replaced to the extent possible with a provision that comes closest to the meaning of the original provision and that is enforceable. The terms and conditions set forth in this Agreement shall apply notwithstanding any failure of essential purpose of this Agreement or any limited remedy to the maximum extent possible under law.

8.4 You agree to notify Us of any facts or circumstances of which you become aware that would make this Agreement inaccurate in any respect.

You

Date:	
Name:	
Title:	
Address:	

Us

Date:	
Name:	
Title:	
Address:	

7.5.7 Copyright Header Guide

ESP-IDF is released under the Apache License 2.0 with some additional third-party copyrighted code released under various licenses. For further information please refer to *the list of copyrights and licenses*.

This page explains how the source code should be properly marked with a copyright header. ESP-IDF uses The Software Package Data Exchange (SPDX) format which is short and can be easily read by humans or processed by automated tools for copyright checks.

How to Check the Copyright Headers

Please make sure you have installed *the pre-commit hooks* which contain a copyright header checker as well. The checker can suggest a header if it is not able to detect a properly formatted SPDX header.

What if the Checker's Suggestion is Incorrect?

No automated checker (no matter how good is) can replace humans. So the developer's responsibility is to modify the offered header to be in line with the law and the license restrictions of the original code on which the work is based on. Certain licenses are not compatible between each other. Such corner cases will be covered by the following examples.

The checker can be configured with the `tools/ci/check_copyright_config.yaml` configuration file. Please check the options it offers and consider updating it in order to match the headers correctly.

Common Examples of Copyright Headers

The simplest case is when the code is not based on any licensed previous work, e.g. it was written completely from scratch. Such code can be decorated with the following copyright header and put under the license of ESP-IDF:

```
/*
 * SPDX-FileCopyrightText: 2015-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: Apache-2.0
 */
```

Less restrictive parts of ESP-IDF Some parts of ESP-IDF are deliberately under less restrictive licenses in order to ease their re-use in commercial closed source projects. This is the case for ESP-IDF examples which are in Public domain or under the Creative Commons Zero Universal (CC0) license. The following header can be used in such source files:

```
/*
 * SPDX-FileCopyrightText: 2015-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: Unlicense OR CC0-1.0
 */
```

The option allowing multiple licenses joined with the `OR` keyword from the above example can be achieved with the definition of multiple allowed licenses in the `tools/ci/check_copyright_config.yaml` configuration file. Please use this option with care and only selectively for a limited part of ESP-IDF.

Third party licenses Code licensed under different licenses, modified by Espressif Systems and included in ESP-IDF cannot be licensed under Apache License 2.0 not even if the checker suggests it. It is advised to keep the original copyright header and add an SPDX before it.

The following example is a suitable header for a code licensed under the "GNU General Public License v2.0 or later" held by John Doe with some additional modifications done by Espressif Systems:

```
/*
 * SPDX-FileCopyrightText: 1991 John Doe
 *
 * SPDX-License-Identifier: GPL-2.0-or-later
 *
 * SPDX-FileContributor: 2019-2023 Espressif Systems (Shanghai) CO LTD
 */
```

The licenses can be identified and the short SPDX identifiers can be found in the official SPDX license list. Other very common licenses are the GPL-2.0-only, the BSD-3-Clause, and the BSD-2-Clause.

In exceptional case, when a license is not present on the SPDX license list, it can be expressed by using the LicenseRef-[idString] custom license identifier, for example `LicenseRef-Special-License`. The full license text must be added into the `LICENSES` directory under `Special-License` filename.

```
/*
 * SPDX-FileCopyrightText: 2015-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: LicenseRef-Special-License
 */
```

Dedicated `LicenseRef-Included` custom license identifier can be used to express a situation when the custom license is included directly in the source file.

```
/*
 * SPDX-FileCopyrightText: 2015-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: LicenseRef-Included
 *
 * <Full custom license text>
 */
```

The configuration stored in `tools/ci/check_copyright_config.yaml` offers features useful for third party licenses:

- A different license can be defined for the files part of a third party library.
- The check for a selected set of files can be permanently disabled. Please use this option with care and only in cases when none of the other options are suitable.

7.5.8 ESP-IDF Tests with Pytest Guide

This documentation is a guide that introduces the following aspects:

1. The basic idea of different test types in ESP-IDF
2. How to apply the pytest framework to the test python scripts to make sure the apps are working as expected.
3. ESP-IDF CI target test process
4. Run ESP-IDF tests with pytest locally
5. Tips and tricks on pytest

Disclaimer

In ESP-IDF, we use the following plugins by default:

- pytest-embedded with default services `esp,idf`
- pytest-rerunfailures

All the introduced concepts and usages are based on the default behavior in ESP-IDF. Not all of them are available in vanilla pytest.

Installation

All dependencies could be installed by running the install script with the `--enable-pytest` argument, e.g., `$ install.sh --enable-pytest`.

Common Issues During Installation

No Package 'dbus-1' found If you're facing an error message like:

```
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.
```

If you're running under ubuntu system, you may need to run:

```
sudo apt-get install libdbus-glib-1-dev
```

or

```
sudo apt-get install libdbus-1-dev
```

For other linux distros, you may Google the error message and find the solution. This issue could be solved by installing the related header files.

Invalid command 'bdist_wheel' If you're facing an error message like:

```
error: invalid command 'bdist_wheel'
```

You may need to run:

```
python -m pip install -U pip
```

Or

```
python -m pip install wheel
```

Before running the pip commands, please make sure you're using the IDF python virtual environment.

Basic Concepts

Component-based Unit Tests Component-based unit tests are our recommended way to test your component. All the test apps should be located under `${IDF_PATH}/components/<COMPONENT_NAME>/test_apps`.

For example:

```
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
```

Example Tests Example Tests are tests for examples that are intended to demonstrate parts of the ESP-IDF functionality to our customers.

All the test apps should be located under `${IDF_PATH}/examples`. For more information please refer to the Examples Readme .

For example:

```
examples/
└── parent_folder/
    └── example_1/
        ├── main/
        │   └── ...
        ├── CMakeLists.txt
        └── pytest_example_1.py
```

Custom Tests Custom Tests are tests that aim to run some arbitrary test internally. They are not intended to demonstrate the ESP-IDF functionality to our customers in any way.

All the test apps should be located under `${IDF_PATH}/tools/test_apps`. For more information please refer to the Custom Test Readme .

Pytest in ESP-IDF

Pytest Execution Process

1. Bootstrapping Phase
   Create session-scoped caches:
   - port-target cache
   - port-app cache
2. Collection Phase
   1. Get all the python files with the prefix `pytest_`
   2. Get all the test functions with the prefix `test_`
   3. Apply the params, and duplicate the test functions.
   4. Filter the test cases with CLI options. Introduced detailed usages *here*
3. Test Running Phase
   1. Construct the fixtures. In ESP-IDF, the common fixtures are initialized in this order:
      1. `pexpect_proc`: pexpect instance

2. `app`: IdfApp instance
   The information of the app, like sdkconfig, flash_files, partition_table, etc., would be parsed at this phase.
3. `serial`: IdfSerial instance
   The port of the host which connected to the target type parsed from the app would be auto-detected. The flash files would be auto flashed.
4. `dut`: IdfDut instance

2. Run the real test function
3. Deconstruct the fixtures in this order:
   1. `dut`
      1. close the `serial` port
      2. (Only for apps with unity test framework) generate junit report of the unity test cases
   2. `serial`
   3. `app`
   4. `pexpect_proc`: Close the file descriptor
4. (Only for apps with unity test framework)
   Raise `AssertionError` when detected unity test failed if you call `dut.expect_from_unity_output()` in the test function.

4. Reporting Phase
   1. Generate junit report of the test functions
   2. Modify the junit report test case name into ESP-IDF test case ID format: `<target>.<config>.<test function name>`
5. Finalizing Phase (Only for apps with unity test framework)
   Combine the junit reports if the junit reports of the unity test cases are generated.

Getting Started Example This code example is taken from 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')
```

Let's go through this simple test case line by line in the following subsections.

Use Markers to Specify the Supported Targets

```
@pytest.mark.esp32      # <-- support esp32
@pytest.mark.esp32c3    # <-- support esp32c3
@pytest.mark.generic    # <-- test env "generic"
```

The above lines indicate that this test case supports target esp32 and esp32c3, the target board type should be "generic" . If you want to know what is the "generic" type refers to, you may run `pytest --markers` to get the detailed information of all markers.

备注: If the test case supports all officially ESP-IDF supported targets (You may check the value via "idf.py –list-targets"), you can use a special marker `supported_targets` to apply all of them in one line.

Use Params to Specify the sdkconfig Files You can use `pytest.mark.parametrize` with "config" to apply the same test to different apps with different sdkconfig files. For more information about `sdkconfig.ci.`

`xxx` files, please refer to the Configuration Files section under this readme .

```
@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
```

Overall, this test function would be replicated to 4 test cases:

- esp32.history.test_console_advanced
- esp32.nohistory.test_console_advanced
- esp32c3.history.test_console_advanced
- esp32c3.nohistory.test_console_advanced

Expect From the Serial output

```
def test_console_advanced(config: str, dut: IdfDut) -> None:  # The value of␣
↪argument ``config`` is assigned by the parametrization.
    if config == 'history':
        dut.expect('Command history enabled')
    elif config == 'nohistory':
        dut.expect('Command history disabled')
```

When we're using `dut.expect(...)`, the string would be compiled into regex at first, and then seeks through the serial output until the compiled regex is matched, or a timeout is exceeded. You may have to pay extra attention when the string contains regex keyword characters, like parentheses, or square brackets.

Actually using `dut.expect_exact(...)` here is better, since it would seek until the string is matched. For further reading about the different types of `expect` functions, please refer to the pytest-embedded Expecting documentation.

Advanced Examples

Multi Dut Tests with the Same App

```
@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
    ...
```

After setting the param `count` to 2, all these fixtures are changed into tuples.

Multi Dut Tests with Different Apps

This code example is taken from 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:
```

(下页继续)

(续上页)

```
    softap = dut[0]
    station = dut[1]
    ...
```

Here the first dut was flashed with the app softap , and the second dut was flashed with the app station .

备注: Here the `app_path` should be set with absolute path. the `__file__` macro in python would return the absolute path of the test script itself.

Multi Dut Tests with Different Apps, and Targets This code example is taken from pytest_wifi_getting_started.py . As the comment says, for now it's not running in the ESP-IDF CI.

```
@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]
    ...
```

Overall, this test function would be replicated to 2 test cases:

- softap with esp32 target, and station with esp32s2 target
- softap with esp32s2 target, and station with esp32 target

Support different targets with different sdkconfig files This code example is taken from pytest_panic.py as an advanced example.

```
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]),
    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)
...
```

Use Custom Class Usually, you can write a custom class in these conditions:

1. Add more reusable functions for a certain number of DUTs

2. Add custom setup and teardown functions in different phases described *here*

This code example is taken from 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` provide a module-scoped monkeypatch fixture.

`replace_dut_class` is a module-scoped autouse fixture. This function replaces the `IdfDut` class with your custom class.

Mark Flaky Tests Sometimes, our test is based on ethernet or wifi. The network may cause the test flaky. We could mark the single test case within the code repo.

This code example is taken from pytest_esp_eth.py

```
@pytest.mark.flaky(reruns=3, reruns_delay=5)
def test_esp_eth_ip101(dut: IdfDut) -> None:
    ...
```

This flaky marker means that if the test function failed, the test case would rerun for a maximum of 3 times with 5 seconds delay.

Mark Known Failure Cases Sometimes a test couldn't pass for the following reasons:

- Has a bug
- The success ratio is too low because of environment issue, such as network issue. Retry couldn't help

Now you may mark this test case with marker xfail with a user-friendly readable reason.

This code example is taken from 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:
```

This marker means that if the test would be a known failure one on esp32s2.

Mark Nightly Run Test Cases Some test cases are only triggered in nightly run pipelines due to a lack of runners.

```
@pytest.mark.nightly_run
```

This marker means that the test case would only be run with env var `NIGHTLY_RUN` or `INCLUDE_NIGHTLY_RUN`.

Mark Temp Disabled in CI Some test cases which can pass locally may need to be temporarily disabled in CI due to a lack of runners.

```
@pytest.mark.temp_skip_ci(targets=['esp32', 'esp32s2'], reason='lack of runners')
```

This marker means that the test case could still be run locally with `pytest --target esp32`, but will not run in CI.

Run Unity Test Cases For component-based unit test apps, one line could do the trick to run all single-board test cases, including normal test cases and multi-stage test cases:

```
def test_component_ut(dut: IdfDut):
    dut.run_all_single_board_cases()
```

It would also skip all the test cases with `[ignore]` mark.

If you need to run a group of test cases, you may run:

```
def test_component_ut(dut: IdfDut):
    dut.run_all_single_board_cases(group='psram')
```

It would trigger all test cases with module name `[psram]`.

You may also see that there are some test scripts with the following statements, which are deprecated. Please use the suggested one as above.

```
def test_component_ut(dut: IdfDut):
    dut.expect_exact('Press ENTER to see the list of tests')
    dut.write('*')
    dut.expect_unity_test_output()
```

For further reading about our unit testing in ESP-IDF, please refer to *our unit testing guide*.

Run the Tests in CI

The workflow in CI is simple, build jobs -> target test jobs.

Build Jobs

Build Job Names

- Component-based Unit Tests: `build_pytest_components_<target>`
- Example Tests: `build_pytest_examples_<target>`
- Custom Tests: `build_pytest_test_apps_<target>`

Build Job Commands The command used by CI to build all the relevant tests is: `python $IDF_PATH/tools/ci/ci_build_apps.py <parent_dir> --target <target> -vv --pytest-apps`

All apps which supported the specified target would be built with all supported sdkconfig files under `build_<target>_<config>`.

For example, If you run `python $IDF_PATH/tools/ci/ci_build_apps.py $IDF_PATH/examples/system/console/basic --target esp32 --pytest-apps`, the folder structure would be like this:

```
basic
├── build_esp32_history/
│   └── ...
├── build_esp32_nohistory/
│   └── ...
├── main/
├── CMakeLists.txt
├── pytest_console_basic.py
└── ...
```

All the binaries folders would be uploaded as artifacts under the same directories.

Target Test Jobs

Target Test Job Names

- Component-based Unit Tests: `component_ut_pytest_<target>_<test_env>`
- Example Tests: `example_test_pytest_<target>_<test_env>`
- Custom Tests: `test_app_test_pytest_<target>_<test_env>`

Target Test Job Commands The command used by CI to run all the relevant tests is: `pytest <parent_dir> --target <target> -m <test_env_marker>`

All test cases with the specified target marker and the test env marker under the parent folder would be executed.

The binaries in the target test jobs are downloaded from build jobs, the artifacts would be placed under the same directories.

Run the Tests Locally

First you need to install ESP-IDF with additional python requirements:

```
$ cd $IDF_PATH
$ bash install.sh --enable-pytest
$ . ./export.sh
```

By default, the pytest script will look for the build directory in this order:

- `build_<target>_<sdkconfig>`
- `build_<target>`
- `build_<sdkconfig>`
- `build`

Which means, the simplest way to run pytest is calling `idf.py build`.

For example, if you want to run all the esp32 tests under the `$IDF_PATH/examples/get-started/hello_world` folder, you should run:

```
$ cd examples/get-started/hello_world
$ idf.py build
$ pytest --target esp32
```

If you have multiple sdkconfig files in your test app, like those `sdkconfig.ci.*` files, the simple `idf.py build` won't apply the extra sdkconfig files. Let's take `$IDF_PATH/examples/system/console/basic` as an example.

If you want to test this app with config `history`, and build with `idf.py build`, you should run

```
$ cd examples/system/console/basic
$ idf.py -DSDKCONFIG_DEFAULTS="sdkconfig.defaults;sdkconfig.ci.history" build
$ pytest --target esp32 --sdkconfig history
```

If you want to build and test with all sdkconfig files at the same time, you should use our CI script as an helper script:

```
$ cd examples/system/console/basic
$ python $IDF_PATH/tools/ci/ci_build_apps.py . --target esp32 -vv --pytest-apps
$ pytest --target esp32
```

The app with `sdkconfig.ci.history` will be built in `build_esp32_history`, and the app with `sdkconfig.ci.nohistory` will be built in `build_esp32_nohistory`. `pytest --target esp32` will run tests on both apps.

Tips and Tricks

Filter the Test Cases

- filter by target with `pytest --target <target>`
  pytest would run all the test cases that support specified target.
- filter by sdkconfig file with `pytest --sdkconfig <sdkconfig>`
  if `<sdkconfig>` is `default`, pytest would run all the test cases with the sdkconfig file `sdkconfig.defaults`.
  In other cases, pytest would run all the test cases with sdkconfig file `sdkconfig.ci.<sdkconfig>`.

Add New Markers We're using two types of custom markers, target markers which indicate that the test cases should support this target, and env markers which indicate that the test case should be assigned to runners with these tags in CI.

You can add new markers by adding one line under the `${IDF_PATH}/conftest.py`. If it's a target marker, it should be added into `TARGET_MARKERS`. If it's a marker that specifies a type of test environment, it should be added into `ENV_MARKERS`. The grammar should be: `<marker_name>: <marker_description>`.

Generate JUnit Report You can call pytest with `--junitxml <filepath>` to generate the JUnit report. In ESP-IDF, the test case name would be unified as "<target>.<config>.<function_name>".

Skip Auto Flash Binary Skipping auto-flash binary every time would be useful when you're debugging your test script.

You can call pytest with `--skip-autoflash y` to achieve it.

Record Statistics Sometimes you may need to record some statistics while running the tests, like the performance test statistics.

You can use record_xml_attribute fixture in your test script, and the statistics would be recorded as attributes in the JUnit report.

Logging System Sometimes you may need to add some extra logging lines while running the test cases.

You can use python logging module to achieve this.

Useful Logging Functions (as Fixture)

`log_performance`

```
def test_hello_world(
    dut: IdfDut,
    log_performance: Callable[[str, object], None],
) -> None:
    log_performance('test', 1)
```

The above example would log the performance item with pre-defined format: "[performance][test]: 1" and record it under the `properties` tag in the junit report if `--junitxml <filepath>` is specified. The junit test case node would look like:

```
<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"/>
```

(下页继续)

(续上页)

```
    </properties>
</testcase>
```

check_performance We provide C macros `TEST_PERFORMANCE_LESS_THAN` and `TEST_PERFORMANCE_GREATER_THAN` to log the performance item and check if the value is in the valid range. Sometimes the performance item value could not be measured in C code, so we also provide a python function for the same purpose. Please note that using C macros is the preferred approach, since the python function couldn't recognize the threshold values of the same performance item under different ifdef blocks well.

```
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')
```

The above example would first get the threshold values of the performance item `RSA_2048KEY_PUBLIC_OP` from components/idf_test/include/idf_performance.h and the target-specific one components/idf_test/include/esp32/idf_performance_target.h, then check if the value reached the minimum limit or exceeded the maximum limit.

Let's assume the value of `IDF_PERFORMANCE_MAX_RSA_2048KEY_PUBLIC_OP` is 19000. so the first `check_performance` line would pass and the second one would fail with warning: `[Performance] RSA_2048KEY_PUBLIC_OP value is 19001, doesn\'t meet pass standard 19000.0`

Further Readings

- pytest documentation: https://docs.pytest.org/en/latest/contents.html
- pytest-embedded documentation: https://docs.espressif.com/projects/pytest-embedded/en/latest/

Chapter 8

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。

8.1 发布版本

您可以通过以下链接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。您可以在该页面查看各个版本的发布说明、配套文档及相应获取方式。

8.2 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。
- 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，您可以将一个最适合您的稳定版本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。

8.3 版本管理

ESP-IDF 采用了 语义版本管理方法，即您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`）意味着您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`）意味着您可能不需要更新您的工程代码，但需重新测试您的工程，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`）意味着您不需要更新您的工程代码，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。

8.4 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：

- 如您刚开始一个新项目，建议使用最新稳定版本。
- 如您有 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通知您。当您所使用的版本有 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。
- 请确保您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。

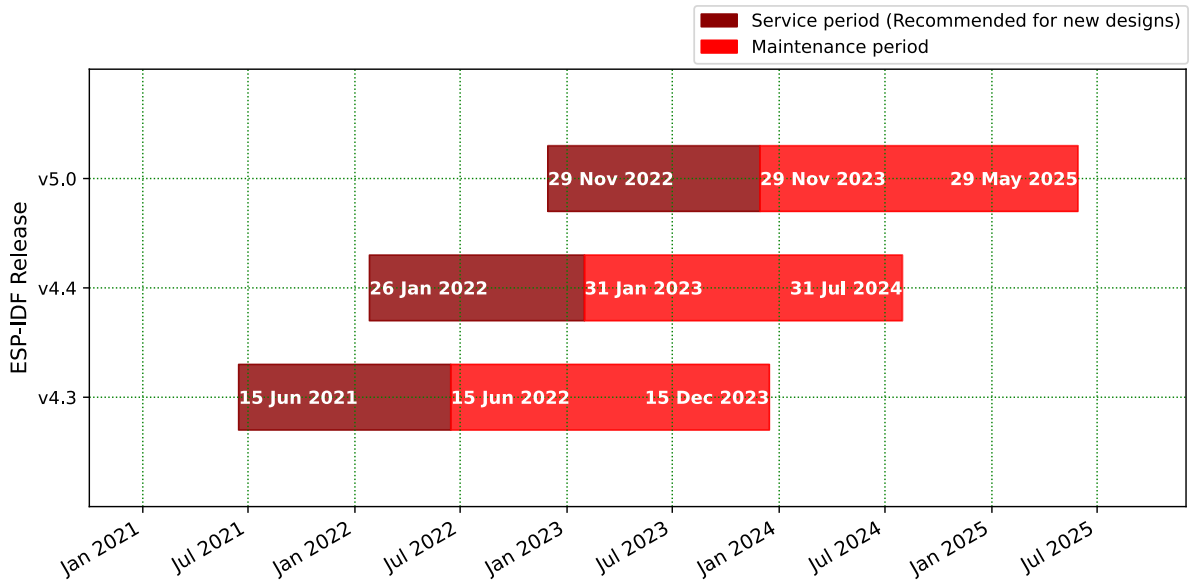
在维护期内，该版本仍受支持，但只会对严重性较高的问题或安全问题进行 bug 修复。

当开始一个新项目时，建议使用在服务期内的版本。

鼓励用户在您所用的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。



8.5 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，因此您也可以使用宏 `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。

8.6 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`）。您可以在 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 。

8.7 更新 ESP-IDF

请根据您的实际情况，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。

8.7.1 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 版本更新至您的 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，并决定是否升级您的版本。具体命令与上方描述一致。

备注： 如果您之前在安装 ESP-IDF 时使用了 zip 文件包，而非通过 Git 命令，则您将无法使用 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。

8.7.2 更新至一个预发布版本

您也可以将您的本地副本切换（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。

8.7.3 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，您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，您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： 强烈建议您定期使用 `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。

8.7.4 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，您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%`。

您每次在该分支上使用 `git pull` 时都相当于把最新的 Bugfix 版本发布更新至您的本地副本中。

备注： 发布分支并不会有专门的配套文档，建议您使用与本分支最接近版本的文档。

Chapter 9

资源

9.1 PlatformIO

9.1.1 What is PlatformIO?

PlatformIO is a cross-platform embedded development environment with out-of-the-box support for ESP-IDF.

Since ESP-IDF support within PlatformIO is not maintained by the Espressif team, please report any issues with PlatformIO directly to its developers in the official PlatformIO repositories.

A detailed overview of the PlatformIO ecosystem and its philosophy can be found in the official PlatformIO documentation.

9.1.2 Installation

- PlatformIO IDE is a toolset for embedded C/C++ development available on Windows, macOS and Linux platforms
- PlatformIO Core (CLI) is a command-line tool that consists of multi-platform build system, platform and library managers and other integration components. It can be used with a variety of code development environments and allows integration with cloud platforms and web services

9.1.3 Configuration

Please go through the official PlatformIO configuration guide for ESP-IDF.

9.1.4 Tutorials

- ESP-IDF and ESP32-DevKitC: debugging, unit testing, project analysis

9.1.5 Project Examples

Please check ESP-IDF page in the official PlatformIO documentation

9.1.6 Next Steps

Here are some useful links for exploring the PlatformIO ecosystem:

- Learn more about integrations with other IDEs/Text Editors
- Get help from PlatformIO community

9.2 有用的链接

- 您可以在 ESP32 论坛 中提出您的问题，访问社区资源。
- 您可以通过 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。
- 您可以在 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。
- 如果您有兴趣参与到 ESP-IDF 的开发，请查阅*Contributions Guide*。
- 关于 ESP32-C2 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 10

Copyrights and Licenses

10.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2022 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.

10.1.1 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) 2003-2005 Jouni Malinen and licensed under the BSD 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.

10.1.2 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.

10.2 ROM Source Code Copyrights

ESP32, ESP32-S and ESP32-C Series 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.
- wpa_supplicant Copyright (c) 2003-2005 Jouni Malinen and licensed under the BSD license.
- TJpgDec Copyright (C) 2011, ChaN, all right reserved. See below for license.

10.3 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.

10.4 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.

10.5 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 11

关于本指南

本指南为 ESP32-C2 官方应用开发框架 ESP-IDF 的配套文档。

ESP32-C2 是一款支持 2.4 GHz Wi-Fi 和低功耗蓝牙的芯片，搭载 RISC-V RV32IMAC 32 位单核处理器。

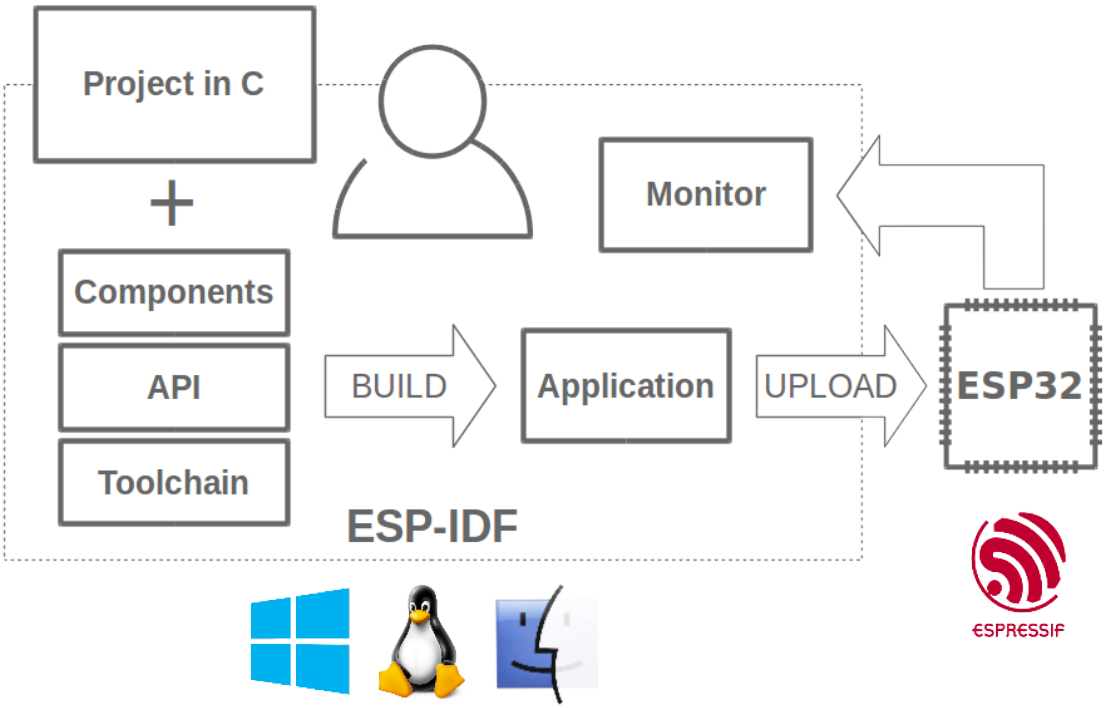


图 1: 乐鑫物联网综合开发框架

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-C2 应用程序提供工具链、API、组件和工作流程的支持。

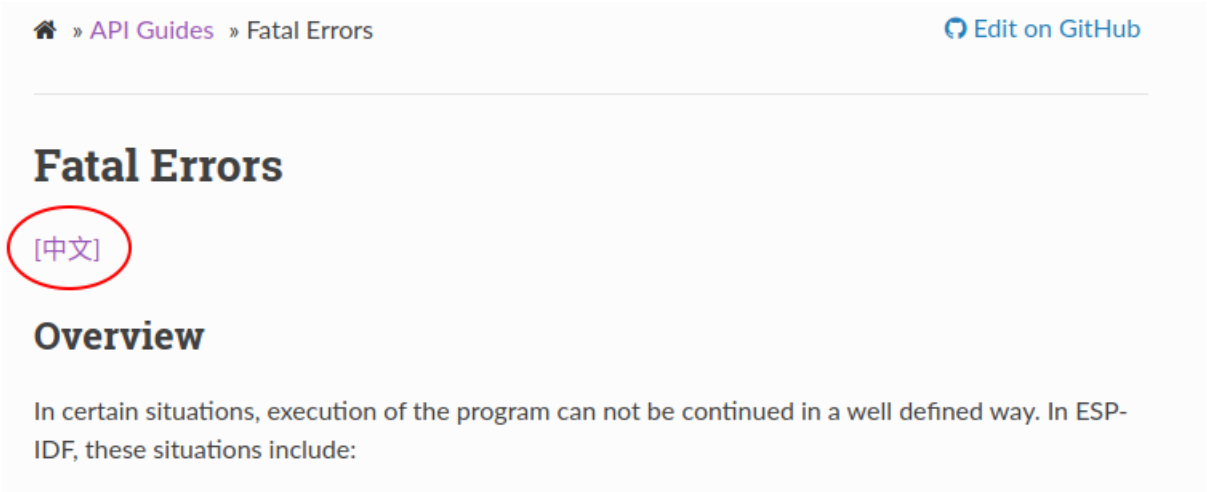
Chapter 12

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，您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

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X