

# ESP-IoT-Solution 用户指南



Release master  
乐鑫信息科技  
2024 年 09 月 20 日

# Table of contents

<b>Table of contents</b>	<b>i</b>
<b>1 快速入门</b>	<b>3</b>
1.1 ESP-IoT-Solution 简介	3
1.1.1 ESP-IoT-Solution 版本	3
1.2 ESP-IDF 简介	3
1.3 ESP 系列 SoC 简介	4
1.4 配置开发环境	4
1.4.1 1. 安装 ESP-IDF	4
1.4.2 2. 获取 ESP-IoT-Solution	4
1.5 使用 ESP-IoT-Solution 组件	4
1.6 编译和下载	5
1.6.1 1. 设置环境变量	5
1.6.2 2. 设置编译目标	5
1.6.3 3. 编译、下载程序	5
1.6.4 4. 串口打印 log	5
1.7 相关文档	6
<b>2 基础组件</b>	<b>7</b>
2.1 通信总线组件 (Bus)	7
2.1.1 i2c_bus 使用方法	7
2.1.2 spi_bus 使用方法	9
2.1.3 已适配 IDF 版本	10
2.1.4 已适配芯片	10
2.1.5 API 参考	10
2.2 I2S LCD 驱动	18
2.2.1 API 参考	18
2.3 板级支持组件 (Boards)	20
2.3.1 使用方法	21
2.3.2 开发板切换和配置	21
2.3.3 已支持的开发板	22
2.3.4 添加新的开发板	23
2.3.5 组件依赖	23
2.3.6 已适配 IDF 版本	23
2.3.7 已适配芯片	23
<b>3 蓝牙</b>	<b>25</b>
3.1 BLE 连接管理	25
3.1.1 应用示例	25
3.1.2 示例	25
3.1.3 API 参考	25
3.2 BLE 服务	34
3.2.1 警报通知服务	34
3.2.2 电池服务	37
3.2.3 设备信息服务	51
3.2.4 心率服务	55
3.2.5 健康温度计服务	57

3.2.6	TX 电源服务	61
3.3	BLE 配置文件	62
3.3.1	警报通知配置文件	62
3.3.2	心率配置文件	66
3.3.3	健康温度计配置文件	68
3.4	BLE HCI 组件	72
3.4.1	BLE HCI 使用方法	72
3.4.2	API 参考	72
<b>4</b>	<b>显示</b>	<b>79</b>
4.1	LCD 显示屏	79
4.1.1	LCD 概述	79
4.1.2	LCD 术语表	90
4.1.3	LCD 开发指南	90
4.1.4	SPI LCD 详解	94
4.1.5	RGB LCD 详解	101
4.1.6	LCD 屏幕撕裂详解	114
4.1.7	LCD Application Solution	117
4.2	数码管驱动	124
4.2.1	CH450 驱动	125
4.2.2	HT16C21 驱动	126
4.2.3	IS31FL3XXX 驱动	127
4.3	LED 指示灯	127
4.3.1	支持的指示灯类型	128
4.3.2	定义闪烁类型	128
4.3.3	预定义闪烁优先级	132
4.3.4	控制指示灯闪烁	132
4.3.5	自定义指示灯闪烁	133
4.3.6	gamma 曲线调光	133
4.3.7	驱动电平设置	133
4.3.8	API 参考	133
4.4	LCD 工具	139
4.4.1	ESP LV SPNG	140
4.4.2	ESP MMAP ASSETS	141
<b>5</b>	<b>USB 主机 &amp; 设备</b>	<b>145</b>
5.1	USB 外设综述	145
5.1.1	ESP USB 外设介绍	145
5.1.2	USB-OTG 外设介绍	147
5.1.3	USB-Serial-JTAG 外设介绍	149
5.1.4	USB PHY/Transceiver 介绍	151
5.1.5	USB VID 和 PID	152
5.1.6	USB Host 方案	153
5.1.7	USB Device 方案	155
5.1.8	自供电 USB 设备解决方案	157
5.1.9	阻止 Windows 依据 USB 设备序列号递增 COM 编号	159
5.1.10	TinyUSB 应用指南	159
5.1.11	使用原生的 tinyusb 进行开发	163
5.2	USB 主机驱动	166
5.2.1	USB Stream 组件说明	166
5.2.2	ESP MSC OTA	178
5.2.3	USB 主机 CDC	182
5.3	USB 设备驱动	186
5.3.1	USB Device UVC	186
5.3.2	ESP Device UAC	189
<b>6</b>	<b>音频</b>	<b>193</b>
6.1	PWM 音频	193
6.1.1	特性	193

6.1.2	结构	193
6.1.3	PWM 频率	194
6.1.4	应用示例	194
6.1.5	API 参考	194
6.2	DAC 音频	198
6.2.1	API 参考	198
<b>7</b>	<b>图形界面</b>	<b>201</b>
7.1	LVGL 图形库	201
7.1.1	主要特性	201
7.1.2	配置要求	201
7.1.3	在线工具	201
7.1.4	示例方案	202
<b>8</b>	<b>人工智能</b>	<b>205</b>
8.1	OpenAI	205
8.1.1	FAQ	205
8.1.2	API 参考	205
<b>9</b>	<b>输入设备</b>	<b>221</b>
9.1	按键	221
9.1.1	按键事件	222
9.1.2	配置项	223
9.1.3	应用示例	224
9.1.4	API Reference	228
9.2	键盘扫描	234
9.2.1	组件事件	234
9.2.2	应用示例	235
9.2.3	API Reference	236
9.3	旋钮	240
9.3.1	适用的场景	240
9.3.2	硬件设计	241
9.3.3	旋钮事件	241
9.3.4	配置项	242
9.3.5	应用示例	242
9.3.6	API Reference	244
9.4	触摸屏驱动	247
9.4.1	校准触摸屏	247
9.4.2	触摸屏的按下	247
9.4.3	触摸屏的旋转	247
9.4.4	应用示例	247
9.4.5	API 参考	248
<b>10</b>	<b>红外</b>	<b>255</b>
10.1	红外学习	255
10.1.1	应用示例	255
10.1.2	API Reference	257
<b>11</b>	<b>传感器集</b>	<b>261</b>
11.1	Sensor Hub 简介	261
11.1.1	Sensor Hub 使用方法	261
11.1.2	示例程序	262
11.1.3	API 参考	262
11.2	温湿度传感器	271
11.2.1	已适配列表	271
11.2.2	API 参考	271
11.3	惯性传感器 (IMU)	273
11.3.1	已适配列表	274
11.3.2	API 参考	274



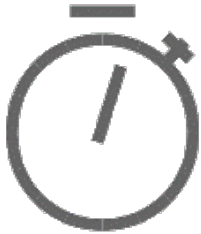







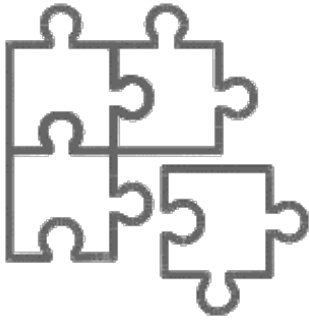
11.4	环境光传感器	276
11.4.1	已适配列表	276
11.4.2	API 参考	276
11.5	气压传感器	278
11.5.1	已适配列表	279
11.6	手势传感器	279
11.6.1	已适配列表	279
11.7	热敏电阻传感器	279
11.7.1	应用示例	280
11.7.2	API 参考	280
11.8	功率监视器	282
11.8.1	适配列表	282
11.8.2	API 参考	282
<b>12</b>	<b>触摸传感器</b>	<b>285</b>
12.1	触摸接近感应传感器	285
12.1.1	实现原理	285
12.1.2	测试硬件参考	286
12.1.3	配置参考	286
12.1.4	参数调节参考	288
12.1.5	示例程序	290
12.1.6	API Reference	290
<b>13</b>	<b>存储方案</b>	<b>297</b>
13.1	存储媒介	297
13.1.1	SPI Flash	297
13.1.2	SD Card	298
13.1.3	eMMC	298
13.1.4	EEPROM	298
13.2	文件系统	298
13.2.1	NVS 库	299
13.2.2	FAT 文件系统	299
13.2.3	SPIFFS 文件系统	300
13.2.4	LittleFS 文件系统	300
13.2.5	虚拟文件系统 (VFS)	300
<b>14</b>	<b>电机</b>	<b>301</b>
14.1	无刷电机	301
14.1.1	无刷电机控制概述	301
14.1.2	基于 ADC 采样的无感方波电机控制	305
14.1.3	基于比较器检测的无感方波电机控制	308
14.1.4	ESP Sensorless BLDC Control Components	310
14.2	舵机	320
14.2.1	使用方法	320
14.2.2	应用示例	321
14.2.3	API 参考	321
<b>15</b>	<b>安全 &amp; 加密</b>	<b>325</b>
15.1	Flash 加密	325
15.1.1	概述	325
15.1.2	使用步骤	325
15.1.3	加密过程（第一次 boot 时进行）	325
15.1.4	串口重烧 flash（3 次重烧机会）	326
15.1.5	FLASH_CRYPT_CNT	326
15.1.6	被加密的数据	326
15.1.7	哪些方式读到解密后的数据（真实数据）	326
15.1.8	哪些方式读到不解密的数据（无法使用的脏数据）	327
15.1.9	软件写入加密数据	327
15.2	安全启动	327

15.2.1	概述	327
15.2.2	所用资源	327
15.2.3	执行过程	327
15.2.4	使用步骤	328
15.2.5	注意事项	328
15.2.6	可重复烧写 bootloader	328
15.2.7	Secure Boot 与 Flash Encryption 流程图	328
15.2.8	开发阶段使用可重复烧写 flash 的 Secure Boot 与 Flash encryption	330
15.3	启用安全加密的生产方案	332
15.3.1	Windows 平台的下载工具	332
15.3.2	操作步骤	332
<b>16</b>	<b>其它资源</b>	<b>339</b>
16.1	GPIO 扩展	339
16.1.1	已适配列表	339
16.2	ADC 扩展量程方案	339
16.2.1	ESP32-S3 ADC 扩展量程方案	339
16.2.2	Patch 使用方法	340
16.2.3	API 使用说明	340
16.3	过零检测	340
16.3.1	过零检测事件	340
16.3.2	配置项	340
16.3.3	应用示例	341
16.3.4	API Reference	341
<b>17</b>	<b>Contributions Guide</b>	<b>347</b>
17.1	How to Contribute	347
17.2	Before Contributing	347
17.3	Pull Request Process	347
17.4	Legal Part	347
17.5	Related Documents	348
17.5.1	esp-iot-solution 编码规范	348
	<b>索引</b>	<b>355</b>
	<b>索引</b>	<b>355</b>



这里是乐鑫 [ESP-IoT-Solution](#) 开发框架的文档中心。

ESP-IoT-Solution 包含物联网系统开发中常用的外设驱动和代码框架，可作为 [ESP-IDF](#) 的补充组件，方便用户实现更简单的开发。

		
<a href="#">入门指南</a>	<a href="#">显示设备</a>	<a href="#">USB 主机 &amp; 设备</a>
		
<a href="#">图形界面</a>	<a href="#">输入设备</a>	<a href="#">传感器集</a>
		
<a href="#">音频设备</a>	<a href="#">安全 &amp; 加密</a>	<a href="#">贡献代码</a>



# Chapter 1

## 快速入门

本文档旨在指导用户搭建 ESP-IoT-Solution (Espressif IoT Solution) 开发环境，通过一个简单的示例展示如何使用 ESP-IoT-Solution 搭建环境、创建工程、编译和下载固件至 ESP 系列开发板等步骤。

### 1.1 ESP-IoT-Solution 简介

ESP-IoT-Solution 包含物联网系统开发中常用的外设驱动和代码框架，可提供 ESP-IDF 的补充组件，方便用户实现更简单的开发，其中包含的内容如下：

- 传感器、显示屏、音频设备、输入设备、执行机构等设备驱动；
- 低功耗、安全加密、存储方案等代码框架或说明文档；
- 从实际应用的角度出发，为乐鑫开源解决方案提供了入口指引。

#### 1.1.1 ESP-IoT-Solution 版本

不同版本的 ESP-IoT-Solution 说明如下：

ESP-IoT-Solution 版本	对应的 ESP-IDF 版本	主要变更	支持状态
master	>=v4.4	支持组件管理器，增加支持新的芯片支持	新功能开发分支
release/v1.1	v4.0.1	IDF 版本更新，删除已经移动到其它仓库的代码	停止维护
release/v1.0	v3.2.2	历史版本	停止维护

master 分支使用 ESP 组件管理器来管理组件，因此每个组件都是一个单独的软件包，每个包可能支持不同版本的 ESP-idf，这些版本将在组件的 idf\_component.yml 文件中声明。

### 1.2 ESP-IDF 简介

ESP-IDF 是乐鑫为 ESP 系列芯片提供的物联网开发框架：

- ESP-IDF 包含一系列库及头文件，提供了基于 ESP SoC 构建软件项目所需的核心组件；
- ESP-IDF 还提供了开发和量产过程中最常用的工具及功能，例如：构建、烧录、调试和测量等。

---

备注：详情请查阅：[ESP-IDF 编程指南](#)。

---

## 1.3 ESP 系列 SoC 简介

您可以选择任意 ESP 系列开发板使用 ESP-IoT-Solution，或者选择 [板级支持组件](#) 中支持的开发板快速开始。

ESP 系列 SoC 支持以下功能：

- 2.4 GHz Wi-Fi
- 蓝牙
- 高性能单核、双核处理器，运行频率可达 240 MHz
- 超低功耗协处理器
- 多种外设，包括 GPIO、I2C、I2S、SPI、UART、SDIO、RMT、LEDC PWM、Ethernet、TWAI、Touch、USB OTG 等
- 丰富的内存资源，内部 RAM 可达 520 KB，同时支持扩展 PSRAM
- 支持硬件加密等安全功能

ESP 系列 SoC 采用 40nm 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。

---

**备注：**不同系列 SoC 配置不同，详情请查阅 [ESP 产品选型工具](#)。

---

## 1.4 配置开发环境

### 1.4.1 1. 安装 ESP-IDF

由于 ESP-IoT-Solution 依赖 ESP-IDF 的基础功能和编译工具，因此首先需要参考 [ESP-IDF 详细安装步骤](#) 完成 ESP-IDF 开发环境的搭建。请注意，不同版本的 ESP-IoT-Solution 依赖的 ESP-IDF 版本可能不同，请参考 [ESP-IoT-Solution 版本](#) 进行选择。

### 1.4.2 2. 获取 ESP-IoT-Solution

若选择 master 版本，可使用以下指令获取代码：

```
git clone --recursive https://github.com/espressif/esp-iot-solution
```

对于 release/v1.1 版本，可使用以下指令获取代码：

```
git clone -b release/v1.1 --recursive https://github.com/espressif/esp-iot-solution
```

对于其它版本，请将 release/v1.1 替换成目标分支名。

## 1.5 使用 ESP-IoT-Solution 组件

如果您只想使用 ESP-IoT-Solution 中的组件，我们建议您从 ESP 组件注册表 [ESP Component Registry](#) 中使用它。

ESP-IoT-Solution 中注册的组件可在 [README\\_CN.md](#) 中查阅，您可以在项目根目录下使用 `idf.py add-dependency` 命令直接将组件从 [Component Registry](#) 添加到项目中。例如，执行 `idf.py add-dependency "espressif/usb_stream"` 命令添加 `usb_stream`，该组件将在 CMake 步骤中自动下载。

请参考 [IDF Component Manager](#) 查看更多关于组件管理器的细节。

## 1.6 编译和下载

### 1.6.1 1. 设置环境变量

以上步骤安装的代码和工具尚未添加至 `PATH` 环境变量，无法通过终端窗口使用这些工具。添加环境变量的步骤如下：

- 添加 ESP-IDF 环境变量：  
Windows 在 CMD 窗口运行：

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

Linux 和 macOS 在终端运行：

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

请将以上指令中的路径，替换成实际安装路径。

- 添加 IOT\_SOLUTION\_PATH 环境变量：  
Windows 在 CMD 窗口运行：

```
set IOT_SOLUTION_PATH=C:\esp\esp-iot-solution
```

Linux 和 macOS 在终端运行：

```
export IOT_SOLUTION_PATH=~/.esp/esp-iot-solution
```

**备注：** 以上方法设置的环境变量，仅对当前终端有效，如果打开新终端，请重新执行以上步骤。

### 1.6.2 2. 设置编译目标

ESP-IDF 同时支持 esp32、esp32s2 等多款芯片，因此需要在编译代码之前设置的编译目标芯片（默认编译目标为 esp32），例如设置编译目标为 esp32s2：

```
idf.py set-target esp32s2
```

对于 ESP-IoT-Solution 中基于 [板级支持组件](#) 开发的 example，还可以使用 menuconfig 在 Board Options -> Choose Target Board 中选择一个目标开发板：

```
idf.py menuconfig
```

### 1.6.3 3. 编译、下载程序

使用 idf.py 工具编译、下载程序，指令为：

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

请将 `PORT` 替换为当前使用的端口号，Windows 系统串口号一般为 COMx，Linux 系统串口号一般为 /dev/ttyUSBx，macOS 串口号一般为 /dev/cu..。

### 1.6.4 4. 串口打印 log

使用 idf.py 工具查看 log，指令为：

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

请将 `PORT` 替换为当前使用的端口号，Windows 系统串口号一般为 COMx，Linux 系统串口号一般为 /dev/ttyUSBx，macOS 串口号一般为 /dev/cu..。



## 1.7 相关文档

- [ESP-IDF 详细安装步骤](#)
- [ESP-IDF 编程指南](#)
- [ESP 产品选型工具](#)

## Chapter 2

# 基础组件

### 2.1 通信总线组件 (Bus)

通信总线组件 (Bus) 是建立在 ESP-IDF 外设驱动代码之上的一套应用层代码，包括 `i2c_bus`、`spi_bus` 等，主要用于 ESP 芯片与外置设备之间的总线通信。该组件从应用开发的角度出发，实现了以下功能：

1. 简化外设初始化步骤
2. 线程安全的设备操作
3. 简单灵活的读写操作

该组件对以下概念进行了抽象：

1. 总线 (Bus)：通信时设备之间共同拥有的资源和配置项
2. 设备 (Device)：通信时设备特有的资源和配置项

每个物理外设总线 (Bus) 在电气条件允许的情况下，均可挂载一到多个设备 (Device)，其中 SPI 总线根据 CS 引脚对设备进行寻址，I2C 总线则根据设备地址进行寻址，进而实现相同总线不同设备之间软件上的独立。

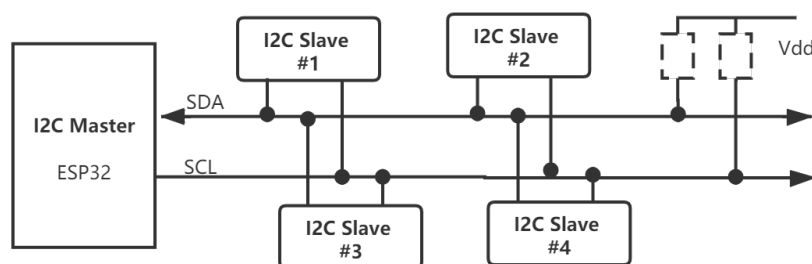


图 1: i2c\_bus 连接框图

#### 2.1.1 i2c\_bus 使用方法

1. 创建总线：使用 `i2c_bus_create()` 创建一个总线实例。创建时需要指定 I2C 端口号，以及总线配置项 `i2c_config_t`。配置项包括 SDA 和 SCL 引脚号、上下拉模式，因为这些配置项在系统设计时已经确定，一般不在运行时切换。总线配置项还包括总线默认的时钟频率，在设备不指定频率时使用。

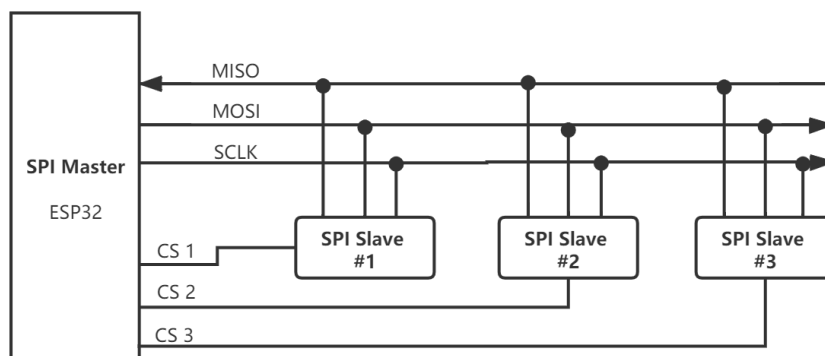


图 2: spi\_bus 连接框图

2. 创建设备：使用 `i2c_bus_device_create()` 在已创建的总线实例之上创建设备，创建时需要指定总线句柄、设备的 I2C 地址、设备运行的时钟频率，I2C 传输时将根据设备的配置项动态切换频率。设备时钟速率可配置为 0，表示默认使用当前的总线频率。
3. 数据读取：使用 `i2c_bus_read_byte()`、`i2c_bus_read_bytes()` 可直接进行 Byte 的读取操作；使用 `i2c_bus_read_bit()`、`i2c_bus_read_bits()` 可直接进行 bit 的读取操作。只需要传入设备句柄、设备寄存器地址、用于存放读取数据的 buf 和读取长度的等。寄存器地址可设为 `NULL_I2C_MEM_ADDR`，用于操作没有内部寄存器的设备。
4. 数据写入：使用 `i2c_bus_write_byte()`、`i2c_bus_write_bytes()` 可直接进行 Byte 的写入操作；使用 `i2c_bus_write_bit()`、`i2c_bus_write_bits()` 可直接进行 bit 的写入操作。只需要传入设备句柄、设备寄存器地址、将要写入的数据位置和写入长度等。寄存器地址可设为 `NULL_I2C_MEM_ADDR`，用于操作没有内部寄存器的设备。
5. 删除设备和总线：如果所有的 i2c\_bus 通信已经完成，可以通过删除设备和总线实例释放系统资源。可使用 `i2c_bus_device_delete()` 分别将已创建的设备删除，然后使用 `i2c_bus_delete()` 将总线资源删除。如果在设备未删除时删除总线，操作将不会被执行。

示例：

```
i2c_config_t conf = {
    .mode = I2C_MODE_MASTER,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .master.clk_speed = 100000,
}; // i2c_bus configurations

uint8_t data_rd[2] = {0};
uint8_t data_wr[2] = {0x01, 0x21};

i2c_bus_handle_t i2c0_bus = i2c_bus_create(I2C_NUM_0, &conf); // create i2c_bus
i2c_bus_device_handle_t i2c_device1 = i2c_bus_device_create(i2c0_bus, 0x28, 400000); // create device1, address: 0x28, clk_speed: 400000
i2c_bus_device_handle_t i2c_device2 = i2c_bus_device_create(i2c0_bus, 0x32, 0); // create device2, address: 0x32, clk_speed: no-specified

i2c_bus_read_bytes(i2c_device1, NULL_I2C_MEM_ADDR, 2, data_rd); // read bytes from device1 with no register address
i2c_bus_write_bytes(i2c_device2, 0x10, 2, data_wr); // write bytes to device2 register 0x10

i2c_bus_device_delete(&i2c_device1); //delete device1
i2c_bus_device_delete(&i2c_device2); //delete device2
i2c_bus_delete(i2c0_bus); //delete i2c_bus
```

**备注：**对于某些特殊应用场景，例如：

1. 当寄存器地址为 16 位时，可以使用 `i2c_bus_read_reg16()` 或 `i2c_bus_write_reg16()` 进行读写操作；
2. 对于需要跳过地址阶段或者需要增加命令阶段的设备，可以使用 `i2c_bus_cmd_begin()` 结合 [I2C command link](#) 进行操作。

## 2.1.2 spi\_bus 使用方法

1. 创建总线：使用 `spi_bus_create()` 创建一个总线实例，创建时需要指定 SPI 端口号（可选 SPI2\_HOST、SPI3\_HOST）以及总线配置项 `spi_config_t`。配置项包括 MOSI、MISO、SCLK 引脚号，因为这些引脚在系统设计时已经确定，一般不在运行时切换。总线配置项还包括 `max_transfer_sz`，用于配置一次传输时的最大数据量，设置为 0 将使用默认值 4096。
2. 创建设备：使用 `spi_bus_device_create()` 在已创建的总线实例之上创建设备，创建时需要指定总线句柄、设备的 CS 引脚号、设备运行模式、设备运行的时钟频率，SPI 传输时将根据设备的配置项动态切换模式和频率。
3. 数据传输：使用 `spi_bus_transfer_byte()`、`spi_bus_transfer_bytes()`、`spi_bus_transfer_reg16()` 以及 `spi_bus_transfer_reg32()` 可直接进行数据的传输操作。由于 SPI 是全双工通信，因此每次传输发送和接收可以同时进行，只需要传入设备句柄、待发送数据、存放读取数据的 `buf` 和传输长度。
4. 删除设备和总线：如果所有的 `spi_bus` 通信已经完成，可以通过删除设备和总线实例释放系统资源。可使用 `spi_bus_device_delete()` 分别将已创建的设备删除，然后使用 `spi_bus_delete()` 将总线资源删除。如果在设备未删除时删除总线，操作将不会被执行。

示例：

```
spi_bus_handle_t bus_handle = NULL;
spi_bus_device_handle_t device_handle = NULL;
uint8_t data8_in = 0;
uint8_t data8_out = 0xff;
uint16_t data16_in = 0;
uint32_t data32_in = 0;

spi_config_t bus_conf = {
    .miso_io_num = 19,
    .mosi_io_num = 23,
    .sclk_io_num = 18,
}; // spi_bus configurations

spi_device_config_t device_conf = {
    .cs_io_num = 19,
    .mode = 0,
    .clock_speed_hz = 20 * 1000 * 1000,
}; // spi_device configurations

bus_handle = spi_bus_create(SPI2_HOST, &bus_conf); // create spi bus
device_handle = spi_bus_device_create(bus_handle, &device_conf); // create spi_
↳device

spi_bus_transfer_bytes(device_handle, &data8_out, &data8_in, 1); // transfer 1_
↳byte with spi device
spi_bus_transfer_bytes(device_handle, NULL, &data8_in, 1); // only read 1 byte_
↳with spi device
spi_bus_transfer_bytes(device_handle, &data8_out, NULL, 1); // only write 1 byte_
↳with spi device
spi_bus_transfer_reg16(device_handle, 0x1020, &data16_in); // transfer 16-bit_
↳value with the device
spi_bus_transfer_reg32(device_handle, 0x10203040, &data32_in); // transfer 32-bit_
↳value with the device
```

(下页继续)

```
spi_bus_device_delete(&device_handle);
spi_bus_delete(&bus_handle);
```

**备注：**对于某些特殊应用场景，可以直接使用 `spi_bus_transmit_begin()` 结合 `spi_transaction_t` 进行操作。

### 2.1.3 已适配 IDF 版本

- ESP-IDF v4.0 及以上版本。

### 2.1.4 已适配芯片

- ESP32
- ESP32-S2

### 2.1.5 API 参考

#### i2c\_bus API 参考

##### Header File

- [components/i2c\\_bus/include/i2c\\_bus.h](#)

##### Functions

**`i2c_bus_handle_t i2c_bus_create`** (`i2c_port_t port`, `const i2c_config_t *conf`)

Create an I2C bus instance then return a handle if created successfully. Each I2C bus works in a singleton mode, which means for an i2c port only one group parameter works. When `i2c_bus_create` is called more than one time for the same i2c port, following parameter will override the previous one.

##### 参数

- **port** –I2C port number
- **conf** –Pointer to I2C bus configuration

**返回** `i2c_bus_handle_t` Return the I2C bus handle if created successfully, return NULL if failed.

**`esp_err_t i2c_bus_delete`** (`i2c_bus_handle_t *p_bus_handle`)

Delete and release the I2C bus resource.

**参数** **p\_bus\_handle** –Point to the I2C bus handle, if delete succeed handle will set to NULL.

##### 返回

- ESP\_OK Success
- ESP\_FAIL Fail

**`uint8_t i2c_bus_scan`** (`i2c_bus_handle_t bus_handle`, `uint8_t *buf`, `uint8_t num`)

Scan i2c devices attached on i2c bus.

##### 参数

- **bus\_handle** –I2C bus handle
- **buf** –Pointer to a buffer to save devices' address, if NULL no address will be saved.
- **num** –Maximum number of addresses to save, invalid if buf set to NULL, higher addresses will be discarded if num less-than the total number found on the I2C bus.

**返回** `uint8_t` Total number of devices found on the I2C bus

`uint32_t i2c_bus_get_current_clk_speed (i2c_bus_handle_t bus_handle)`

Get current active clock speed.

**参数** `bus_handle` –I2C bus handle

**返回** `uint32_t` current clock speed

`uint8_t i2c_bus_get_created_device_num (i2c_bus_handle_t bus_handle)`

Get created device number of the bus.

**参数** `bus_handle` –I2C bus handle

**返回** `uint8_t` created device number of the bus

`i2c_bus_device_handle_t i2c_bus_device_create (i2c_bus_handle_t bus_handle, uint8_t dev_addr, uint32_t clk_speed)`

Create an I2C device on specific bus. Dynamic configuration must be enable to achieve multiple devices with different configs on a single bus. menuconfig:Bus Options->I2C Bus Options->enable dynamic configuration.

**参数**

- **bus\_handle** –Point to the I2C bus handle
- **dev\_addr** –i2c device address
- **clk\_speed** –device specified clock frequency the i2c\_bus will switch to during each transfer. 0 if use current bus speed.

**返回** `i2c_bus_device_handle_t` return a device handle if created successfully, return NULL if failed.

`esp_err_t i2c_bus_device_delete (i2c_bus_device_handle_t *p_dev_handle)`

Delete and release the I2C device resource, `i2c_bus_device_delete` should be used in pairs with `i2c_bus_device_create`.

**参数** `p_dev_handle` –Point to the I2C device handle, if delete succeed handle will set to NULL.

**返回**

- `ESP_OK` Success
- `ESP_FAIL` Fail

`uint8_t i2c_bus_device_get_address (i2c_bus_device_handle_t dev_handle)`

Get device' s I2C address.

**参数** `dev_handle` –I2C device handle

**返回** `uint8_t` I2C address, return `NULL_I2C_DEV_ADDR` if `dev_handle` is invalid.

`esp_err_t i2c_bus_read_byte (i2c_bus_device_handle_t dev_handle, uint8_t mem_address, uint8_t *data)`

Read single byte from i2c device with 8-bit internal register/memory address.

**参数**

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal reg/mem address to read from, set to `NULL_I2C_MEM_ADDR` if no internal address.
- **data** –Pointer to a buffer to save the data that was read

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_FAIL` Sending command error, slave doesn' 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_bus_read_bytes (i2c_bus_device_handle_t dev_handle, uint8_t mem_address, size_t data_len, uint8_t *data)`

Read multiple bytes from i2c device with 8-bit internal register/memory address. If internal reg/mem address is 16-bit, please refer `i2c_bus_read_reg16`.

**参数**

- **dev\_handle** –I2C device handle

- **mem\_address** –The internal reg/mem address to read from, set to NULL\_I2C\_MEM\_ADDR if no internal address.
- **data\_len** –Number of bytes to read
- **data** –Pointer to a buffer to save the data that was read

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_FAIL Sending command error, slave doesn'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\_bus\_read\_bit** ([i2c\\_bus\\_device\\_handle\\_t](#) dev\_handle, uint8\_t mem\_address, uint8\_t bit\_num, uint8\_t \*data)

Read single bit of a byte from i2c device with 8-bit internal register/memory address.

参数

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal reg/mem address to read from, set to NULL\_I2C\_MEM\_ADDR if no internal address.
- **bit\_num** –The bit number 0 - 7 to read
- **data** –Pointer to a buffer to save the data that was read. \*data == 0 -> bit = 0, \*data !=0 -> bit = 1.

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_FAIL Sending command error, slave doesn'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\_bus\_read\_bits** ([i2c\\_bus\\_device\\_handle\\_t](#) dev\_handle, uint8\_t mem\_address, uint8\_t bit\_start, uint8\_t length, uint8\_t \*data)

Read multiple bits of a byte from i2c device with 8-bit internal register/memory address.

参数

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal reg/mem address to read from, set to NULL\_I2C\_MEM\_ADDR if no internal address.
- **bit\_start** –The bit to start from, 0 - 7, MSB at 0
- **length** –The number of bits to read, 1 - 8
- **data** –Pointer to a buffer to save the data that was read

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_FAIL Sending command error, slave doesn'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\_bus\_write\_byte** ([i2c\\_bus\\_device\\_handle\\_t](#) dev\_handle, uint8\_t mem\_address, uint8\_t data)

Write single byte to i2c device with 8-bit internal register/memory address.

参数

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal reg/mem address to write to, set to NULL\_I2C\_MEM\_ADDR if no internal address.
- **data** –The byte to write.

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_FAIL Sending command error, slave doesn'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_bus_write_bytes` (*[i2c\\_bus\\_device\\_handle\\_t](#)* dev\_handle, uint8\_t mem\_address, size\_t data\_len, const uint8\_t \*data)

Write multiple byte to i2c device with 8-bit internal register/memory address. If internal reg/mem address is 16-bit, please refer `i2c_bus_write_reg16`.

#### 参数

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal reg/mem address to write to, set to `NULL_I2C_MEM_ADDR` if no internal address.
- **data\_len** –Number of bytes to write
- **data** –Pointer to the bytes to write.

#### 返回

- `esp_err_t`
- **ESP\_OK** Success
  - **ESP\_ERR\_INVALID\_ARG** Parameter error
  - **ESP\_FAIL** Sending command error, slave doesn'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_bus_write_bit` (*[i2c\\_bus\\_device\\_handle\\_t](#)* dev\_handle, uint8\_t mem\_address, uint8\_t bit\_num, uint8\_t data)

Write single bit of a byte to an i2c device with 8-bit internal register/memory address.

#### 参数

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal reg/mem address to write to, set to `NULL_I2C_MEM_ADDR` if no internal address.
- **bit\_num** –The bit number 0 - 7 to write
- **data** –The bit to write, `data == 0` means set bit = 0, `data != 0` means set bit = 1.

#### 返回

- `esp_err_t`
- **ESP\_OK** Success
  - **ESP\_ERR\_INVALID\_ARG** Parameter error
  - **ESP\_FAIL** Sending command error, slave doesn'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_bus_write_bits` (*[i2c\\_bus\\_device\\_handle\\_t](#)* dev\_handle, uint8\_t mem\_address, uint8\_t bit\_start, uint8\_t length, uint8\_t data)

Write multiple bits of a byte to an i2c device with 8-bit internal register/memory address.

#### 参数

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal reg/mem address to write to, set to `NULL_I2C_MEM_ADDR` if no internal address.
- **bit\_start** –The bit to start from, 0 - 7, MSB at 0
- **length** –The number of bits to write, 1 - 8
- **data** –The bits to write.

#### 返回

- `esp_err_t`
- **ESP\_OK** Success
  - **ESP\_ERR\_INVALID\_ARG** Parameter error
  - **ESP\_FAIL** Sending command error, slave doesn'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_bus_cmd_begin` (*[i2c\\_bus\\_device\\_handle\\_t](#)* dev\_handle, *[i2c\\_cmd\\_handle\\_t](#)* cmd)

I2C master send queued commands create by `i2c_cmd_link_create`. This function will trigger sending all queued commands. The task will be blocked until all the commands have been sent out. If `I2C_BUS_DYNAMIC_CONFIG` enable, `i2c_bus` will dynamically check configs and re-install i2c driver before each transfer, hence multiple devices with different configs on a single bus can be supported.



---

**备注:** Only call this function when `i2c_bus_read/write_xx` do not meet the requirements

---

**参数**

- **dev\_handle** –I2C device handle
- **cmd** –I2C command handler

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_FAIL` Sending command error, slave doesn'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_bus_write_reg16` (*`i2c_bus_device_handle_t`* dev\_handle, `uint16_t` mem\_address, `size_t` data\_len, `const uint8_t *`data)

Write data to an i2c device with 16-bit internal reg/mem address.

**参数**

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal 16-bit reg/mem address to write to, set to `NULL_I2C_MEM_ADDR` if no internal address.
- **data\_len** –Number of bytes to write
- **data** –Pointer to the bytes to write.

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_FAIL` Sending command error, slave doesn'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_bus_read_reg16` (*`i2c_bus_device_handle_t`* dev\_handle, `uint16_t` mem\_address, `size_t` data\_len, `uint8_t *`data)

Read data from i2c device with 16-bit internal reg/mem address.

**参数**

- **dev\_handle** –I2C device handle
- **mem\_address** –The internal 16-bit reg/mem address to read from, set to `NULL_I2C_MEM_ADDR` if no internal address.
- **data\_len** –Number of bytes to read
- **data** –Pointer to a buffer to save the data that was read

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_FAIL` Sending command error, slave doesn'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.

**Macros****`NULL_I2C_MEM_ADDR`**

set mem\_address to `NULL_I2C_MEM_ADDR` if i2c device has no internal address during read/write

**`NULL_I2C_DEV_ADDR`**

invalid i2c device address

**`gpio_pad_select_gpio`**

**port**TICK\_RATE\_MS

### Type Definitions

typedef void \*i2c\_bus\_handle\_t

i2c bus handle

typedef void \*i2c\_bus\_device\_handle\_t

i2c device handle

### spi\_bus API 参考

#### Header File

- [components/bus/include/spi\\_bus.h](#)

#### Functions

*spi\_bus\_handle\_t* **spi\_bus\_create** (spi\_host\_device\_t host\_id, const *spi\_config\_t* \*bus\_conf)

Create and initialize a spi bus and return the spi bus handle.

##### 参数

- **host\_id** –SPI peripheral that controls this bus, SPI2\_HOST or SPI3\_HOST
- **bus\_conf** –spi bus configurations details in *spi\_config\_t*

**返回** *spi\_bus\_handle\_t* handle for spi bus operation, NULL if failed.

esp\_err\_t **spi\_bus\_delete** (*spi\_bus\_handle\_t* \*p\_bus\_handle)

Deinitialize and delete the spi bus.

**参数** **p\_bus\_handle** –pointer to spi bus handle, if delete succeed handle will set to NULL.

**返回** esp\_err\_t

- ESP\_ERR\_INVALID\_ARG if parameter is invalid
- ESP\_FAIL Fail
- ESP\_OK Success

*spi\_bus\_device\_handle\_t* **spi\_bus\_device\_create** (*spi\_bus\_handle\_t* bus\_handle, const *spi\_device\_config\_t* \*device\_conf)

Create and add a device on the spi bus.

##### 参数

- **bus\_handle** –handle for spi bus operation.
- **device\_conf** –spi device configurations details in *spi\_device\_config\_t*

**返回** *spi\_bus\_device\_handle\_t* handle for device operation, NULL if failed.

esp\_err\_t **spi\_bus\_device\_delete** (*spi\_bus\_device\_handle\_t* \*p\_dev\_handle)

Deinitialize and remove the device from spi bus.

**参数** **p\_dev\_handle** –pointer to device handle, if delete succeed handle will set to NULL.

**返回** esp\_err\_t

- ESP\_ERR\_INVALID\_ARG if parameter is invalid
- ESP\_FAIL Fail
- ESP\_OK Success

esp\_err\_t **spi\_bus\_transfer\_byte** (*spi\_bus\_device\_handle\_t* dev\_handle, uint8\_t data\_out, uint8\_t \*data\_in)

Transfer one byte with the device.

##### 参数

- **dev\_handle** –handle for device operation.
- **data\_out** –data will send to device.

- **data\_in** –pointer to receive buffer, set NULL to skip receive phase.
- 返回 esp\_err\_t
- ESP\_ERR\_INVALID\_ARG if parameter is invalid
  - ESP\_ERR\_TIMEOUT if bus is busy
  - ESP\_OK on success

esp\_err\_t **spi\_bus\_transfer\_bytes** ([spi\\_bus\\_device\\_handle\\_t](#) dev\_handle, const uint8\_t \*data\_out, uint8\_t \*data\_in, uint32\_t data\_len)

Transfer multi-bytes with the device.

#### 参数

- **dev\_handle** –handle for device operation.
- **data\_out** –pointer to sent buffer, set NULL to skip sent phase.
- **data\_in** –pointer to receive buffer, set NULL to skip receive phase.
- **data\_len** –number of bytes will transfer.

返回 esp\_err\_t

- ESP\_ERR\_INVALID\_ARG if parameter is invalid
- ESP\_ERR\_TIMEOUT if bus is busy
- ESP\_OK on success

esp\_err\_t **spi\_bus\_transmit\_begin** ([spi\\_bus\\_device\\_handle\\_t](#) dev\_handle, spi\_transaction\_t \*p\_trans)

Send a polling transaction, wait for it to complete, and return the result.

---

**备注:** Only call this function when `spi_bus_transfer_xx` do not meet the requirements

---

#### 参数

- **dev\_handle** –handle for device operation.
- **p\_trans** –Description of transaction to execute

返回 esp\_err\_t

- ESP\_ERR\_INVALID\_ARG if parameter is invalid
- ESP\_ERR\_TIMEOUT if bus is busy
- ESP\_OK on success

esp\_err\_t **spi\_bus\_transfer\_reg16** ([spi\\_bus\\_device\\_handle\\_t](#) dev\_handle, uint16\_t data\_out, uint16\_t \*data\_in)

Transfer one 16-bit value with the device. using msb by default. For example 0x1234, 0x12 will send first then 0x34.

#### 参数

- **dev\_handle** –handle for device operation.
- **data\_out** –data will send to device.
- **data\_in** –pointer to receive buffer, set NULL to skip receive phase.

返回 esp\_err\_t

- ESP\_ERR\_INVALID\_ARG if parameter is invalid
- ESP\_ERR\_TIMEOUT if bus is busy
- ESP\_OK on success

esp\_err\_t **spi\_bus\_transfer\_reg32** ([spi\\_bus\\_device\\_handle\\_t](#) dev\_handle, uint32\_t data\_out, uint32\_t \*data\_in)

Transfer one 32-bit value with the device. using msb by default. For example 0x12345678, 0x12 will send first, 0x78 will send in the end.

#### 参数

- **dev\_handle** –handle for device operation.
- **data\_out** –data will send to device.
- **data\_in** –pointer to receive buffer, set NULL to skip receive phase.

返回 esp\_err\_t

- ESP\_ERR\_INVALID\_ARG if parameter is invalid

- ESP\_ERR\_TIMEOUT if bus is busy
- ESP\_OK on success

## Structures

struct **spi\_config\_t**

spi bus initialization parameters.

### Public Members

gpio\_num\_t **miso\_io\_num**

GPIO pin for Master In Slave Out (=spi\_q) signal, or -1 if not used.

gpio\_num\_t **mosi\_io\_num**

GPIO pin for Master Out Slave In (=spi\_d) signal, or -1 if not used.

gpio\_num\_t **sclk\_io\_num**

GPIO pin for Spi CLock signal, or -1 if not used

int **max\_transfer\_sz**

<Maximum length of bytes available to send, if < 4096, 4096 will be set

struct **spi\_device\_config\_t**

spi device initialization parameters.

### Public Members

gpio\_num\_t **cs\_io\_num**

GPIO pin to select this device (CS), or -1 if not used

uint8\_t **mode**

modes (0,1,2,3) that correspond to the four possible clocking configurations

int **clock\_speed\_hz**

spi clock speed, divisors of 80MHz, in Hz. See 'SPI\_MASTER\_FREQ\_\*

## Macros

**NULL\_SPI\_CS\_PIN**

set cs\_io\_num to NULL\_SPI\_CS\_PIN if spi device has no CP pin

## Type Definitions

typedef void \***spi\_bus\_handle\_t**

spi bus handle

typedef void \***spi\_bus\_device\_handle\_t**

spi device handle

## 2.2 I2S LCD 驱动

### 2.2.1 API 参考

#### Header File

- [components/bus/include/i2s\\_lcd\\_driver.h](#)

#### Functions

*i2s\_lcd\_handle\_t* **i2s\_lcd\_driver\_init** (const *i2s\_lcd\_config\_t* \*config)

Initialize i2s lcd driver.

参数 **config** –configuration of i2s

返回 A handle to the created i2s lcd driver, or NULL in case of error.

esp\_err\_t **i2s\_lcd\_driver\_deinit** (*i2s\_lcd\_handle\_t* handle)

Deinit i2s lcd driver.

参数 **handle** –i2s lcd driver handle to deinitialize

返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG handle is invalid

esp\_err\_t **i2s\_lcd\_write\_data** (*i2s\_lcd\_handle\_t* handle, uint16\_t data)

Write a data to LCD.

参数

- **handle** –i2s lcd driver handle
- **data** –Data to write

返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG handle is invalid

esp\_err\_t **i2s\_lcd\_write\_cmd** (*i2s\_lcd\_handle\_t* handle, uint16\_t cmd)

Write a command to LCD.

参数

- **handle** –Handle of i2s lcd driver
- **cmd** –command to write

返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG handle is invalid

esp\_err\_t **i2s\_lcd\_write\_command** (*i2s\_lcd\_handle\_t* handle, const uint8\_t \*cmd, uint32\_t length)

Write a command to LCD.

参数

- **handle** –Handle of i2s lcd driver
- **cmd** –command to write
- **length** –length of command

返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG handle is invalid

esp\_err\_t **i2s\_lcd\_write** (*i2s\_lcd\_handle\_t* handle, const uint8\_t \*data, uint32\_t length)

Write block data to LCD.

参数

- **handle** –Handle of i2s lcd driver
- **data** –Pointer of data

- **length** –length of data

返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG handle is invalid

esp\_err\_t **i2s\_lcd\_acquire** (*i2s\_lcd\_handle\_t* handle)

acquire a lock

参数 **handle** –Handle of i2s lcd driver

返回 Always return ESP\_OK

esp\_err\_t **i2s\_lcd\_release** (*i2s\_lcd\_handle\_t* handle)

release a lock

参数 **handle** –Handle of i2s lcd driver

返回 Always return ESP\_OK

## Structures

struct **i2s\_lcd\_config\_t**

Configuration of i2s lcd mode.

Handle of i2s lcd driver

## Public Members

int8\_t **data\_width**

Parallel data width, 16bit or 8bit available

int8\_t **pin\_data\_num**[16]

Parallel data output IO

int8\_t **pin\_num\_cs**

CS io num

int8\_t **pin\_num\_wr**

Write clk io

int8\_t **pin\_num\_rs**

RS io num

int **clk\_freq**

I2s clock frequency

i2s\_port\_t **i2s\_port**

I2S port number

bool **swap\_data**

Swap the 2 bytes of RGB565 color

uint32\_t **buffer\_size**

DMA buffer size

## Macros

LCD\_CMD\_LEV

LCD\_DATA\_LEV

## Type Definitions

```
typedef void *i2s_lcd_handle_t
```

## 2.3 板级支持组件 (Boards)

本文档主要介绍板级支持组件 (Boards) 的使用方法，该组件作为示例程序的公共组件，可向应用程序提供统一的引脚宏定义和与硬件无关的初始化操作，基于板级支持开发的应用程序可以同时兼容不同的开发板，具体功能如下：

1. 提供统一的引脚资源宏定义
2. 提供默认的外设配置参数
3. 提供统一的板级初始化接口
4. 提供开发板的硬件控制接口

Boards 组件的结构如下：

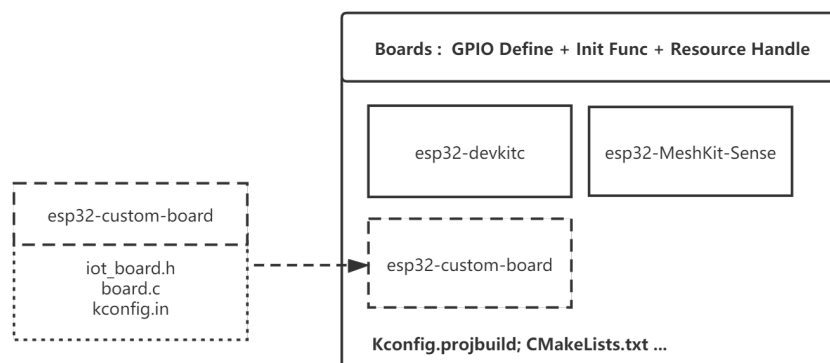


图 3: Boards 组件结构框图

- Boards 组件中包含以下内容：
  - board\_common.h, 包含了公有 API 的函数声明
  - board\_common.c, 包含了对公有 API 的函数实现（虚函数）
  - Kconfig.projbuild, 包含了公有的配置项
- 以开发板名称命名的子文件夹包含以下内容：
  - iot\_board.h 提供了开发板的引脚资源定义，该开发板特有的自定义 API 函数声明
  - board.c 提供了公有 API 的用户实现（默认虚函数），自定义 API 函数实现
  - kconfig.in, 提供了该开发板特有的自定义配置项。

**备注：** Boards 在 examples/common\_components/boards 中提供。

### 2.3.1 使用方法

1. 初始化开发板：在 app\_main 使用 `iot_board_init` 初始化开发板，用户亦可在 menuconfig 中使用 [开发板切换和配置](#) 进行初始化配置；
2. 获取外设句柄：使用 `iot_board_get_handle` 和 `board_res_id_t` 获取外设资源的句柄，如果该外设未被初始化将返回 `NULL`；
3. 使用句柄进行外设操作。

示例：

```
void app_main(void)
{
    /*initialize board with default parameters,
    you can use menuconfig to choose a target board*/
    esp_err_t err = iot_board_init();
    if (err != ESP_OK) {
        goto error;
    }

    /*get the i2c0 bus handle with a board_res_id,
    BOARD_I2C0_ID is declared in board_res_id_t in each iot_board.h*/
    bus_handle_t i2c0_bus_handle = (bus_handle_t)iot_board_get_handle(BOARD_I2C0_
↪ID);
    if (i2c0_bus_handle == NULL) {
        goto error;
    }

    /*
    * use initialized peripheral with handles directly,
    * no configurations required anymore.
    */
}
```

### 2.3.2 开发板切换和配置

基于 Boards 开发的应用程序，可以使用以下方法切换和配置开发板：

1. 选择目标开发板：在 menuconfig->Board Options->Choose Target Board 中选择一个开发板；
2. 配置开发板参数：Board Common Options 中包含公有的开发板配置项，例如配置是否在开发板初始化期间初始化 `i2c_bus`；XXX Board Options 中包含了该开发板特有的配置项，例如切换开发板供电状态等。
3. 使用 `idf.py build flash monitor` 重新编译并下载代码。

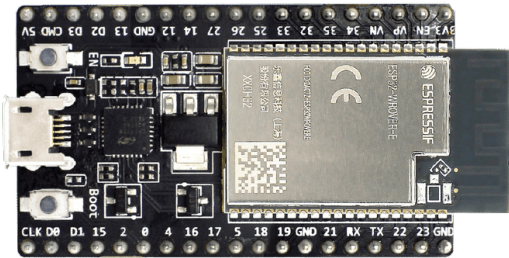
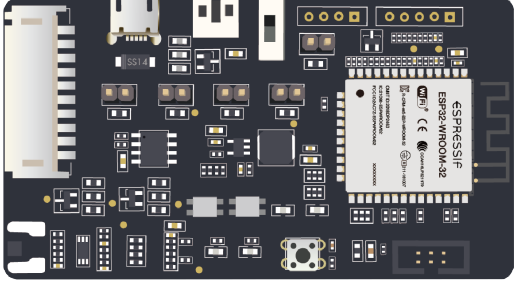
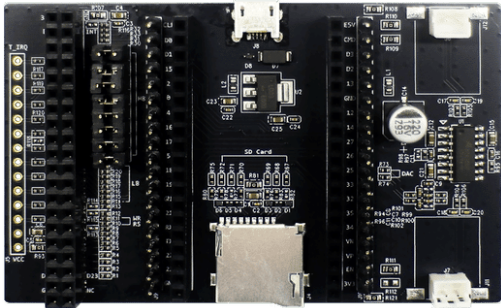
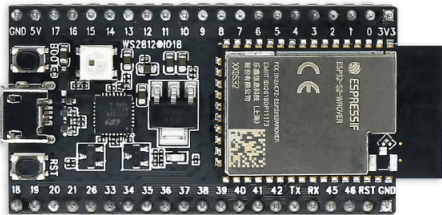

---

**备注：**编译系统编译目标默认为 ESP32，如使用 ESP32-S2，请在编译之前使用 `idf.py set-target esp32s2` 配置目标。

---



2.3.3 已支持的开发板

ESP32 开发板	
	
<a href="#">esp32-devkitc</a>	<a href="#">esp32-meshkit-sense</a>
	
<a href="#">esp32-lcdkit</a>	
ESP32-S2 开发板	
	
<a href="#">esp32s2-saola</a>	
ESP32-S3 开发板	
	

### 2.3.4 添加新的开发板

通过添加新的开发板，可以快速适配基于 Boards 组件开发的应用程序。

添加开发板过程：

1. 按照[组件文件结构](#)准备必要的 `iot_board.h`；
2. 按照需求在 `board_XXX.c` 添加该开发板特有的函数实现，或对公有的弱函数进行覆盖；
3. 按照需求在 `kconfig.in` 添加该开发板特有的配置项；
4. 将开发板信息添加到 `Kconfig.projbuild`，供用户选择；
5. 将开发板目录添加到 `CMakeLists.txt`，使其能被编译系统索引。如果需要支持老的 `make` 编译系统，请同时修改 `component.mk`。

---

**备注：**可复制 Boards 中已添加的开发板文件夹，通过简单修改完成开发板的添加。

---

### 2.3.5 组件依赖

- 公共依赖项：bus, button, led\_strip

### 2.3.6 已适配 IDF 版本

- ESP-IDF v4.4 及以上版本。

### 2.3.7 已适配芯片

- ESP32
- ESP32-S2
- ESP32-S3



## Chapter 3

# 蓝牙

### 3.1 BLE 连接管理

支持的芯片	ESP32	ESP32-C2	ESP32-C3	ESP32-S3
-------	-------	----------	----------	----------

BLE 连接管理为访问常用 BLE 功能提供了简化的 API 接口。它支持外围、中心等常见场景。

#### 3.1.1 应用示例

```
esp_ble_conn_config_t config = {
    .device_name = CONFIG_EXAMPLE_BLE_ADV_NAME,
    .broadcast_data = CONFIG_EXAMPLE_BLE_SUB_ADV
};

ESP_ERROR_CHECK(esp_ble_conn_init(&config));
ESP_ERROR_CHECK(esp_ble_conn_start());
```

#### 3.1.2 示例

1. BLE 周期广告示例: [bluetooth/ble\\_conn\\_mgr/ble\\_periodic\\_adv](#).
2. BLE 周期同步示例: [bluetooth/ble\\_conn\\_mgr/ble\\_periodic\\_sync](#).
3. BLE 串口配置文件示例: [bluetooth/ble\\_conn\\_mgr/ble\\_spp](#).

#### 3.1.3 API 参考

##### Header File

- [components/bluetooth/ble\\_conn\\_mgr/include/esp\\_ble\\_conn\\_mgr.h](#)

##### Functions

esp\_err\_t **esp\_ble\_conn\_init** ([esp\\_ble\\_conn\\_config\\_t](#) \*config)

Initialization *BLE* connection management based on the configuration This function must be the first function to call, This call MUST have a corresponding call to esp\_ble\_conn\_deinit when the operation is complete.

**参数 config** –[in] The configurations, see [esp\\_ble\\_conn\\_config\\_t](#).

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

esp\_err\_t **esp\_ble\_conn\_deinit** (void)

Deinitialization *BLE* connection management This function must be the last function to call, It is the opposite of the esp\_ble\_conn\_init function.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong deinitialization
- ESP\_FAIL on error

esp\_err\_t **esp\_ble\_conn\_start** (void)

Starts *BLE* session.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong start
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_stop** (void)

Stop *BLE* session.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong stop
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_set\_mtu** (uint16\_t mtu)

This api is typically used to update maximum transmission unit value.

**参数 mtu** –[in] The maximum transmission unit value to update

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong update
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_connect** (void)

This api is typically used to connect actively.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG on wrong connect
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_disconnect** (void)

This api is typically used to disconnect actively.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG on wrong disconnect
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_notify** (const [esp\\_ble\\_conn\\_data\\_t](#) \*inbuff)

This api is typically used to notify actively.

**参数 inbuff** –[in] The pointer to store notify data.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG on wrong notify
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_read** (*esp\_ble\_conn\_data\_t* \*outbuf)

This api is typically used to read actively.

**参数 outbuf** –[in] The pointer to store read data.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG on wrong read
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_write** (const *esp\_ble\_conn\_data\_t* \*inbuff)

This api is typically used to write actively.

**参数 inbuff** –[in] The pointer to store write data.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG on wrong write
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_subscribe** (*esp\_ble\_conn\_desc\_t* desc, const *esp\_ble\_conn\_data\_t* \*inbuff)

This api is typically used to subscribe actively.

**参数**

- **desc** –[in] The declarations of descriptors
- **inbuff** –[in] The pointer to store subscribe data.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG on wrong subscribe
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_add\_svc** (const *esp\_ble\_conn\_svc\_t* \*svc)

This api is typically used to add service actively.

**参数 svc** –[in] The pointer to store service.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG on wrong add service
- ESP\_FAIL on other error

esp\_err\_t **esp\_ble\_conn\_remove\_svc** (const *esp\_ble\_conn\_svc\_t* \*svc)

This api is typically used to remove service actively.

**参数 svc** –[in] The pointer to store service.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG on wrong remove service
- ESP\_FAIL on other error

## Unions

union **esp\_ble\_conn\_uuid\_t**

*#include <esp\_ble\_conn\_mgr.h>* Universal UUID, to be used for any-UUID static allocation.

## Public Members

`uint16_t uuid16`

16 bit UUID of the service

`uint32_t uuid32`

32 bit UUID of the service

`uint8_t uuid128[BLE_UUID128_VAL_LEN]`

128 bit UUID of the service

## Structures

struct **esp\_ble\_conn\_character\_t**

This structure maps handler required by UUID which are used to BLE characteristics.

### Public Members

`const char *name`

Name of the handler

`uint8_t type`

Type of the UUID

`uint8_t flag`

Flag for visiting right

*esp\_ble\_conn\_uuid\_t* **uuid**

Universal UUID, to be used for any-UUID static allocation

*esp\_ble\_conn\_cb\_t* **uuid\_fn**

BLE UUID callback

struct **esp\_ble\_conn\_svc\_t**

This structure maps handler required by UUID which are used to BLE services.

### Public Members

`uint8_t type`

Type of the UUID

`uint16_t nu_lookup_count`

Number of entries in the Name-UUID lookup table

*esp\_ble\_conn\_uuid\_t* **uuid**

Universal UUID, to be used for any-UUID static allocation

*esp\_ble\_conn\_character\_t* \***nu\_lookup**

Pointer to the Name-UUID lookup table

struct **esp\_ble\_conn\_config\_t**

This structure maps handler required which are used to configure.

### Public Members

uint8\_t **device\_name**[MAX\_BLE\_DEVNAME\_LEN]

BLE device name being broadcast

uint8\_t **broadcast\_data**[BROADCAST\_PARAM\_LEN]

BLE device manufacturer being announce

uint16\_t **extended\_adv\_len**

Extended advertising data length

uint16\_t **periodic\_adv\_len**

Periodic advertising data length

uint16\_t **extended\_adv\_rsp\_len**

Extended advertising responses data length

const char \***extended\_adv\_data**

Extended advertising data

const char \***periodic\_adv\_data**

Periodic advertising data

const char \***extended\_adv\_rsp\_data**

Extended advertising responses data

uint16\_t **include\_service\_uuid**

If include service UUID in advertising

struct **esp\_ble\_conn\_data\_t**

This structure maps handler required by UUID which are used to data.

### Public Members

uint8\_t **type**

Type of the UUID

uint16\_t **write\_conn\_id**

Connection handle ID



*esp\_ble\_conn\_uuid\_t* **uuid**

Universal UUID, to be used for any-UUID static allocation

**uint8\_t \*data**

Data buffer

**uint16\_t data\_len**

Data size

struct **esp\_ble\_conn\_periodic\_sync\_t**

This structure represents a periodic advertising sync established during discovery procedure.

### **Public Members**

**uint8\_t status**

Periodic sync status

**uint16\_t sync\_handle**

Periodic sync handle

**uint8\_t sid**

Advertising Set ID

**uint8\_t adv\_addr[6]**

Advertiser address

**uint8\_t adv\_phy**

Advertising PHY

**uint16\_t per\_adv\_ival**

Periodic advertising interval

**uint8\_t adv\_clk\_accuracy**

Advertiser clock accuracy

struct **esp\_ble\_conn\_periodic\_report\_t**

This structure represents a periodic advertising report received on established sync.

### **Public Members**

**uint16\_t sync\_handle**

Periodic sync handle

**int8\_t tx\_power**

Advertiser transmit power in dBm (127 if unavailable)

int8\_t **rss\_i**

Received signal strength indication in dBm (127 if unavailable)

uint8\_t **data\_status**

Advertising data status

uint8\_t **data\_length**

Advertising Data length

const uint8\_t \***data**

Advertising data

struct **esp\_ble\_conn\_periodic\_sync\_lost\_t**

This structure represents a periodic advertising sync lost of established sync.

### Public Members

uint16\_t **sync\_handle**

Periodic sync handle

int **reason**

Reason for sync lost

### 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 device name length

**BLE\_UUID128\_VAL\_LEN**

128 bit UUID length

**BROADCAST\_PARAM\_LEN**

BLE device broadcast param length

**BLE\_CONN\_GATT\_CHR\_BROADCAST**

Characteristic broadcast properties

**BLE\_CONN\_GATT\_CHR\_READ**

Characteristic read properties

**BLE\_CONN\_GATT\_CHR\_WRITE\_NO\_RSP**

Characteristic write properties

**BLE\_CONN\_GATT\_CHR\_WRITE**

Characteristic write properties

**BLE\_CONN\_GATT\_CHR\_NOTIFY**

Characteristic notify properties

**BLE\_CONN\_GATT\_CHR\_INDICATE**

Characteristic indicate properties

**BLE\_UUID\_CMP** (type, src, dst)

**BLE\_UUID\_TYPE** (type)

**MIN** (a, b)

## Type Definitions

```
typedef esp_err_t (*esp_ble_conn_cb_t)(const uint8_t *inbuf, uint16_t inlen, uint8_t **outbuf, uint16_t *outlen, void *priv_data)
```

This is type of function that will handle the registered characteristic.

**Param inbuf [in]** The pointer to store data: read operation if NULL or write operation if not NULL

**Param inlen [in]** The store data length

**Param outbuf [out]** Variable to store data, it' ll free by connection management component

**Param outlen [out]** Variable to store data length

**Param priv\_data [in]** Private data context

**Return**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong parameter
- ESP\_FAIL on error

## Enumerations

```
enum esp_ble_conn_event_t
```

Type of event.

*Values:*

enumerator **ESP\_BLE\_CONN\_EVENT\_UNKNOWN**

Unknown event

enumerator **ESP\_BLE\_CONN\_EVENT\_STARTED**

When BLE connection management start, the event comes

enumerator **ESP\_BLE\_CONN\_EVENT\_STOPPED**

When BLE connection management stop, the event comes

enumerator **ESP\_BLE\_CONN\_EVENT\_CONNECTED**

When a new connection was established, the event comes

enumerator **ESP\_BLE\_CONN\_EVENT\_DISCONNECTED**

When a connection was terminated, the event comes

enumerator **ESP\_BLE\_CONN\_EVENT\_DATA\_RECEIVE**

When receive a notification or indication data, the event comes

enumerator **ESP\_BLE\_CONN\_EVENT\_DISC\_COMPLETE**

When the ble discover service complete, the event comes

enumerator **ESP\_BLE\_CONN\_EVENT\_PERIODIC\_REPORT**

When the periodic adv report, the event comes

enumerator **ESP\_BLE\_CONN\_EVENT\_PERIODIC\_SYNC\_LOST**

When the periodic sync lost, the event comes

enumerator **ESP\_BLE\_CONN\_EVENT\_PERIODIC\_SYNC**

When the periodic sync, the event comes

enum **esp\_ble\_conn\_desc\_t**

Type of descriptors.

*Values:*

enumerator **ESP\_BLE\_CONN\_DESC\_UNKNOWN**

Unknown descriptors

enumerator **ESP\_BLE\_CONN\_DESC\_EXTENDED**

Characteristic Extended Properties

enumerator **ESP\_BLE\_CONN\_DESC\_USER**

Characteristic User Description

enumerator **ESP\_BLE\_CONN\_DESC\_CIENT\_CONFIG**

Client Characteristic Configuration

enumerator **ESP\_BLE\_CONN\_DESC\_SERVER\_CONFIG**

Server Characteristic Configuration

enumerator **ESP\_BLE\_CONN\_DESC\_PRE\_FORMAT**

Characteristic Presentation Format

enumerator **ESP\_BLE\_CONN\_DESC\_AGG\_FORMAT**

Characteristic Aggregate Format

enumerator **ESP\_BLE\_CONN\_DESC\_VALID\_RANGE**

Valid Range

enumerator **ESP\_BLE\_CONN\_DESC\_EXTREPORT**

External Report Reference

enumerator **ESP\_BLE\_CONN\_DESC\_REPORT**

Report Reference

enumerator **ESP\_BLE\_CONN\_DESC\_DIGITAL**

Number of Digitals

enumerator **ESP\_BLE\_CONN\_DESC\_VALUE\_TRIGGER**

Value Trigger Setting

enumerator **ESP\_BLE\_CONN\_DESC\_ENV\_SENS\_CONFIG**

Environmental Sensing Configuration

enumerator **ESP\_BLE\_CONN\_DESC\_ENV\_SENS\_MEASURE**

Environmental Sensing Measurement

enumerator **ESP\_BLE\_CONN\_DESC\_ENV\_TRIGGER**

Environmental Sensing Trigger Setting

enumerator **ESP\_BLE\_CONN\_DESC\_TIME\_TRIGGER**

Time Trigger Setting

enumerator **ESP\_BLE\_CONN\_DESC\_COMPLETE\_BLOCK**

Complete BR-EDR Transport Block Data

enum **esp\_ble\_conn\_uuid\_type\_t**

Type of UUID.

*Values:*

enumerator **BLE\_CONN\_UUID\_TYPE\_16**

16-bit UUID (BT SIG assigned)

enumerator **BLE\_CONN\_UUID\_TYPE\_32**

32-bit UUID (BT SIG assigned)

enumerator **BLE\_CONN\_UUID\_TYPE\_128**

128-bit UUID

## 3.2 BLE 服务

### 3.2.1 警报通知服务

警报通知服务公开设备中的警报信息。这些信息包括以下内容:

1. 设备中发生的警报类型
2. 其他文本信息，如来电显示或发件人 ID
3. 新警报的计数
4. 未读警报项的计数。

## 示例

[bluetooth/ble\\_services/ble\\_ans.](#)

## API 参考

### Header File

- [components/bluetooth/ble\\_services/ans/include/esp\\_ans.h](#)

### Functions

`esp_err_t esp_ble_ans_get_new_alert (uint8_t cat_id, uint8_t *cat_val)`

Read the value of or check supported new alert category.

**Attention** 1. When `cat_id` is `0xFF`, read the value of supported new alert category.

**Attention** 2. When `cat_id` isn't `0xFF`, check supported new alert category is enable or disable.

#### 参数

- **cat\_id** –[in] The ID of the category to read or check
- **cat\_val** –[out] The value of read or check supported new alert category

#### 返回

- `ESP_OK` on successful
- `ESP_ERR_INVALID_ARG` on wrong category of the new alert

`esp_err_t esp_ble_ans_set_new_alert (uint8_t cat_id, const char *cat_info)`

Send a new alert notification to the given category with the given info string.

#### 参数

- **cat\_id** –The ID of the category to send the notification to
- **cat\_info** –The info string to send with the notification

#### 返回

- `ESP_OK` on successful
- `ESP_ERR_INVALID_ARG` on wrong category of the new alert

`esp_err_t esp_ble_ans_get_unread_alert (uint8_t cat_id, uint8_t *cat_val)`

Read the value of or check supported unread alert status category.

**Attention** 1. When `cat_id` is `0xFF`, read the value of supported unread alert status category.

**Attention** 2. When `cat_id` isn't `0xFF`, check supported unread alert status category is enable or disable.

#### 参数

- **cat\_id** –[in] The ID of the category to read or check
- **cat\_val** –[out] The value of read or check supported unread alert status category

#### 返回

- `ESP_OK` on successful
- `ESP_ERR_INVALID_ARG` on wrong category of the unread alert

`esp_err_t esp_ble_ans_set_unread_alert (uint8_t cat_id)`

Send an unread alert to the given category then notifies the client if the given category is valid and enabled.

**参数** **cat\_id** –The ID of the category which should be incremented and notified

#### 返回

- `ESP_OK` on successful
- `ESP_ERR_INVALID_ARG` on wrong category of the unread alert

esp\_err\_t **esp\_ble\_ans\_init** (void)

Initialization GATT Alert Notification Service.

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

## Macros

**BLE\_ANS\_UUID16**

**BLE\_ANS\_CHR\_UUID16\_SUP\_NEW\_ALERT\_CAT**

**BLE\_ANS\_CHR\_UUID16\_NEW\_ALERT**

**BLE\_ANS\_CHR\_UUID16\_SUP\_UNR\_ALERT\_CAT**

**BLE\_ANS\_CHR\_UUID16\_UNR\_ALERT\_STAT**

**BLE\_ANS\_CHR\_UUID16\_ALERT\_NOT\_CTRL\_PT**

**BLE\_ANS\_CAT\_BM\_NONE**

**BLE\_ANS\_CAT\_BM\_SIMPLE\_ALERT**

**BLE\_ANS\_CAT\_BM\_EMAIL**

**BLE\_ANS\_CAT\_BM\_NEWS**

**BLE\_ANS\_CAT\_BM\_CALL**

**BLE\_ANS\_CAT\_BM\_MISSED\_CALL**

**BLE\_ANS\_CAT\_BM\_SMS**

**BLE\_ANS\_CAT\_BM\_VOICE\_MAIL**

**BLE\_ANS\_CAT\_BM\_SCHEDULE**

**BLE\_ANS\_CAT\_ID\_SIMPLE\_ALERT**

**BLE\_ANS\_CAT\_ID\_EMAIL**

**BLE\_ANS\_CAT\_ID\_NEWS**

**BLE\_ANS\_CAT\_ID\_CALL**

`BLE_ANS_CAT_ID_MISSED_CALL`

`BLE_ANS_CAT_ID_SMS`

`BLE_ANS_CAT_ID_VOICE_MAIL`

`BLE_ANS_CAT_ID_SCHEDULE`

`BLE_ANS_CAT_NUM`

`BLE_ANS_CMD_EN_NEW_ALERT_CAT`

`BLE_ANS_CMD_EN_UNR_ALERT_CAT`

`BLE_ANS_CMD_DIS_NEW_ALERT_CAT`

`BLE_ANS_CMD_DIS_UNR_ALERT_CAT`

`BLE_ANS_CMD_NOT_NEW_ALERT_IMMEDIATE`

`BLE_ANS_CMD_NOT_UNR_ALERT_IMMEDIATE`

`BLE_ANS_INFO_STR_MAX_LEN`

`BLE_ANS_NEW_ALERT_MAX_LEN`

## 3.2.2 电池服务

电池服务在电池与设备的电气连接上下文中公开电池的电量和其他信息。

### 示例

[bluetooth/ble\\_services/ble\\_bas](#).

### API 参考

#### Header File

- [components/bluetooth/ble\\_services/bas/include/esp\\_bas.h](#)

#### Functions

`esp_err_t esp_ble_bas_get_battery_level (uint8_t *level)`

Get the current battery level of the device.

**参数** `level` –[in] The pointer to store the current battery level

**返回**

- `ESP_OK` on successful
- `ESP_ERR_INVALID_ARG` on wrong battery level



`esp_err_t esp_ble_bas_set_battery_level (uint8_t *level)`

Set the current battery level of the device.

**参数** `level` –[in] The pointer to store the current battery level

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery level

`esp_err_t esp_ble_bas_get_level_status (esp_ble_bas_level_status_t *status)`

Get the summary information related to the battery status of the device.

**参数** `status` –[in] The pointer to store the the summary information related to the battery status

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery status

`esp_err_t esp_ble_bas_set_level_status (esp_ble_bas_level_status_t *status)`

Set the summary information related to the battery status of the device.

**参数** `status` –[in] The pointer to store the the summary information related to the battery status

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery status

`esp_err_t esp_ble_bas_get_estimated_date (uint32_t *estimated_date)`

Get the current estimated date when the battery is likely to require service or replacement.

**参数** `estimated_date` –[in] The pointer to store the current estimated date

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery estimated date

`esp_err_t esp_ble_bas_set_estimated_date (uint32_t *estimated_date)`

Set the current estimated date when the battery is likely to require service or replacement.

**参数** `estimated_date` –[in] The pointer to store the current estimated date

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery estimated date

`esp_err_t esp_ble_bas_get_critical_status (esp_ble_bas_critical_status_t *status)`

Get the status bits related to potential battery malfunction.

**参数** `status` –[in] The pointer to store the status bits related to potential battery malfunction

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery malfunction

`esp_err_t esp_ble_bas_set_critical_status (esp_ble_bas_critical_status_t *status)`

Set the status bits related to potential battery malfunction.

**参数** `status` –[in] The pointer to store the status bits related to potential battery malfunction

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery malfunction

`esp_err_t esp_ble_bas_get_energy_status (esp_ble_bas_energy_status_t *status)`

Get the current energy status details of the battery.

**参数** `status` –[in] The pointer to store the status bits related to potential battery energy

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery energy

esp\_err\_t **esp\_ble\_bas\_set\_energy\_status** (*esp\_ble\_bas\_energy\_status\_t* \*status)

Set the current energy status details of the battery.

**参数 status** –[in] The pointer to store the status bits related to potential battery energy

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery energy

esp\_err\_t **esp\_ble\_bas\_get\_time\_status** (*esp\_ble\_bas\_time\_status\_t* \*status)

Get the current estimates on times for discharging and charging.

**参数 status** –[in] The pointer to store the current estimates on times

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery estimates

esp\_err\_t **esp\_ble\_bas\_set\_time\_status** (*esp\_ble\_bas\_time\_status\_t* \*status)

Set the current estimates on times for discharging and charging.

**参数 status** –[in] The pointer to store the current estimates on times

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery estimates

esp\_err\_t **esp\_ble\_bas\_get\_health\_status** (*esp\_ble\_bas\_health\_status\_t* \*status)

Get the aspects of battery health.

**参数 status** –[in] The pointer to store the aspects of battery health

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery health

esp\_err\_t **esp\_ble\_bas\_set\_health\_status** (*esp\_ble\_bas\_health\_status\_t* \*status)

Set the aspects of battery health.

**参数 status** –[in] The pointer to store the aspects of battery health

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery health

esp\_err\_t **esp\_ble\_bas\_get\_health\_info** (*esp\_ble\_bas\_health\_info\_t* \*info)

Get the static aspects of battery health.

**参数 info** –[in] The pointer to store the static aspects of battery health

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery health information

esp\_err\_t **esp\_ble\_bas\_set\_health\_info** (*esp\_ble\_bas\_health\_info\_t* \*info)

Set the static aspects of battery health.

**参数 info** –[in] The pointer to store the static aspects of battery health

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery health information

esp\_err\_t **esp\_ble\_bas\_get\_battery\_info** (*esp\_ble\_bas\_battery\_info\_t* \*info)

Get the physical characteristics of the battery.

**参数 info** –[in] The pointer to store the physical characteristics of the battery

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery information

esp\_err\_t **esp\_ble\_bas\_set\_battery\_info** (*esp\_ble\_bas\_battery\_info\_t* \*info)

Set the physical characteristics of the battery.

**参数** **info** –[in] The pointer to store the physical characteristics of the battery

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery information

esp\_err\_t **esp\_ble\_bas\_init** (void)

Initialization Battery Service.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

## Structures

struct **uint24\_t**

This structure represents a 24bits data type.

### Public Members

uint32\_t **u24**

A 24bits data

struct **esp\_ble\_bas\_level\_status\_t**

Battery Level Status Characteristic.

### Public Members

uint8\_t **en\_identifier**

Identifier Present

uint8\_t **en\_battery\_level**

Battery Level Present

uint8\_t **en\_additional\_status**

Additional Status Present

uint8\_t **flags\_reserved**

Reserve Feature Used

struct *esp\_ble\_bas\_level\_status\_t*::[anonymous] **flags**

Flags of Battery Level Status

uint16\_t **battery**

Battery Present

uint16\_t **wired\_external\_power\_source**

Wired External Power Source Connected:

uint16\_t **wireless\_external\_power\_source**

Wireless External Power Source Connected:

uint16\_t **battery\_charge\_state**

Battery Charge State:

uint16\_t **battery\_charge\_level**

Battery Charge Level:

uint16\_t **battery\_charge\_type**

Battery Charging Type

uint16\_t **charging\_fault\_reason**

Charging Fault Reason

uint16\_t **power\_state\_reserved**

Reserve Feature Used

struct *esp\_ble\_bas\_level\_status\_t*::[anonymous] **power\_state**

Power State of Battery Level Status

uint16\_t **identifier**

Used as an identifier for a service instance.

uint8\_t **battery\_level**

Refer to the Battery Level

uint8\_t **service\_required**

Service Required

uint8\_t **battery\_fault**

Battery Fault:

uint8\_t **reserved**

Reserve Feature Used

struct *esp\_ble\_bas\_level\_status\_t*::[anonymous] **additional\_status**

Contains additional status information such as whether or not service is required

struct **esp\_ble\_bas\_critical\_status\_t**

Battery Critical Status Characteristic.

### Public Members

uint8\_t **critical\_power\_state**

Critical Power State

uint8\_t **immediate\_service**

Immediate Service Required

uint8\_t **reserved**

Reserve Feature Used

struct *esp\_ble\_bas\_critical\_status\_t*::[anonymous] **status**

Battery Critical Status

struct **esp\_ble\_bas\_energy\_status\_t**

Battery Energy Status Characteristic.

### Public Members

uint8\_t **en\_external\_source\_power**

External Source Power Present

uint8\_t **en\_voltage**

Present Voltage Present

uint8\_t **en\_available\_energy**

Available Energy Present

uint8\_t **en\_available\_battery\_capacity**

Available Battery Capacity Present

uint8\_t **en\_charge\_rate**

Charge Rate Present

uint8\_t **en\_available\_energy\_last\_charge**

Available Energy at Last Charge Present

uint8\_t **reserved**

Reserve Feature Used

struct *esp\_ble\_bas\_energy\_status\_t*::[anonymous] **flags**

Flags of Battery Energy Status

uint16\_t **external\_source\_power**

The total power being consumed from an external power source

uint16\_t **voltage**

The present terminal voltage of the battery in volts.

uint16\_t **available\_energy**

The available energy of the battery in kilowatt-hours in its current charge state

uint16\_t **available\_battery\_capacity**

The capacity of the battery in kilowatt-hours at full charge in its current condition

uint16\_t **charge\_rate**

The energy flowing into the battery in watts

uint16\_t **available\_energy\_last\_charge**

The available energy of the battery in kilowatt-hours in its last charge state

struct **esp\_ble\_bas\_time\_status\_t**

Battery Time Status Characteristic.

### Public Members

uint8\_t **en\_discharged\_standby**

Time until Discharged on Standby Present

uint8\_t **en\_recharged**

Time until Recharged Present

uint8\_t **reserved**

Reserve Feature Used

struct *esp\_ble\_bas\_time\_status\_t*::[anonymous] **flags**

Flags of Battery Time Status

*uint24\_t* **discharged**

Estimated time in minutes until discharged

*uint24\_t* **discharged\_standby**

Estimated time in minutes until discharged assuming for the remaining time the device is in standby.

*uint24\_t* **recharged**

Estimated time in minutes until recharged

struct **esp\_ble\_bas\_health\_status\_t**

Battery Health Status Characteristic.

### Public Members

uint8\_t **en\_battery\_health\_summary**

Battery Health Summary Present

uint8\_t **en\_cycle\_count**

Cycle Count Present

uint8\_t **en\_current\_temperature**

Current Temperature Present

uint8\_t **en\_deep\_discharge\_count**

Deep Discharge Count Present

uint8\_t **reserved**

Reserve Feature Used

struct *esp\_ble\_bas\_health\_status\_t*::[anonymous] **flags**

Flags of Battery Health Status

uint8\_t **battery\_health\_summary**

Represents aggregation of the overall health of the battery

uint16\_t **cycle\_count**

Represents the count value of charge cycles

int8\_t **current\_temperature**

Represents the current temperature of the battery

uint16\_t **deep\_discharge\_count**

Represents the number of times the battery was completely discharged

struct **esp\_ble\_bas\_health\_info\_t**

Battery Health Information Characteristic.

### Public Members

uint8\_t **en\_cycle\_count\_designed\_lifetime**

Cycle Count Designed Lifetime Present

uint8\_t **min\_max\_designed\_operating\_temperature**

Min and Max Designed Operating Temperature Present

uint8\_t **reserved**

Reserve Feature Used

struct *esp\_ble\_bas\_health\_info\_t*::[anonymous] **flags**

Flags of Battery Health Information

uint16\_t **cycle\_count\_designed\_lifetime**

Represents the designed number of charge cycles supported by the device

int8\_t **min\_designed\_operating\_temperature**

Represents the minimum designed operating temperature of the battery

`int8_t max_designed_operating_temperature`

Represents the maximum designed operating temperature of the battery

struct `esp_ble_bas_battery_info_t`

Battery Information Characteristic.

### Public Members

`uint16_t en_manufacture_date`

Battery Manufacture Date Present

`uint16_t en_expiration_date`

Battery Expiration Date Present

`uint16_t en_designed_capacity`

Battery Designed Capacity Present

`uint16_t en_low_energy`

Battery Low Energy Present

`uint16_t en_critical_energy`

Battery Critical Energy Present

`uint16_t en_chemistry`

Battery Chemistry Present

`uint16_t en_nominalvoltage`

Nominal Voltage Present

`uint16_t en_aggregation_group`

Battery Aggregation Group Present

`uint16_t flags_reserved`

Reserve Feature Used

struct `esp_ble_bas_battery_info_t::[anonymous] flags`

Flags of Battery Information

`uint8_t replace_able`

Battery Replaceable

`uint8_t recharge_able`

Battery Rechargeable

`uint8_t reserved`

Reserve Feature Used



struct *esp\_ble\_bas\_battery\_info\_t*::[anonymous] **features**

Features of Battery Information

*uint24\_t* **manufacture\_date**

Battery date of manufacture specified as days

*uint24\_t* **expiration\_date**

Battery expiration date specified as days

*uint16\_t* **designed\_capacity**

The capacity of the battery in kilowatt-hours at full charge in original (new) condition.

*uint16\_t* **low\_energy**

The battery energy value in kilowatt-hours when the battery is low

*uint16\_t* **critical\_energy**

The battery energy value in kilowatt-hours when the battery is critical.

*uint8\_t* **chemistry**

The value of battery chemistry

*uint16\_t* **nominalvoltage**

Nominal voltage of the battery in units of volts

*uint8\_t* **aggregation\_group**

Indicates the Battery Aggregation Group to which this instance of the battery service is associated

struct **esp\_ble\_bas\_data\_t**

Battery Service.

### Public Members

*uint8\_t* **battery\_level**

The charge level of a battery

*esp\_ble\_bas\_level\_status\_t* **level\_status**

The power state of a battery

*uint24\_t* **estimated\_service\_date**

The estimated date when replacement or servicing is required.

*esp\_ble\_bas\_critical\_status\_t* **critical\_status**

The device will possibly not function as expected due to low energy or service required

*esp\_ble\_bas\_energy\_status\_t* **energy\_status**

Details about the energy status of the battery

*esp\_ble\_bas\_time\_status\_t* **time\_status**

Time estimates for discharging and charging

*esp\_ble\_bas\_health\_status\_t* **health\_status**

Several aspects of battery health

*esp\_ble\_bas\_health\_info\_t* **health\_info**

The health of a battery

*esp\_ble\_bas\_battery\_info\_t* **battery\_info**

The physical characteristics of a battery in the context of the battery's connection in a device

### Macros

**BLE\_BAS\_UUID16**

**BLE\_BAS\_CHR\_UUID16\_LEVEL**

**BLE\_BAS\_CHR\_UUID16\_LEVEL\_STATUS**

**BLE\_BAS\_CHR\_UUID16\_ESTIMATED\_SERVICE\_DATE**

**BLE\_BAS\_CHR\_UUID16\_CRITICAL\_STATUS**

**BLE\_BAS\_CHR\_UUID16\_ENERGY\_STATUS**

**BLE\_BAS\_CHR\_UUID16\_TIME\_STATUS**

**BLE\_BAS\_CHR\_UUID16\_HEALTH\_STATUS**

**BLE\_BAS\_CHR\_UUID16\_HEALTH\_INFO**

**BLE\_BAS\_CHR\_UUID16\_BATTERY\_INFO**

**BAS\_CHR\_LEVEL\_STATUS\_FLAGS\_BM\_NONE**

**BAS\_CHR\_LEVEL\_STATUS\_FLAGS\_BM\_IDENTIFY**

**BAS\_CHR\_LEVEL\_STATUS\_FLAGS\_BM\_BATTERY\_LEVEL**

**BAS\_CHR\_LEVEL\_STATUS\_FLAGS\_BM\_ADDITIONAL\_STATUS**

**BAS\_CHR\_LEVEL\_STATUS\_FLAGS\_BM\_RFU**

**BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_NOT**

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_SET

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_WIRED\_EXTERNAL\_POWER\_SOURCE\_NOTCONNECT

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_WIRED\_EXTERNAL\_POWER\_SOURCE\_CONNECTED

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_WIRED\_EXTERNAL\_POWER\_SOURCE\_UNKNOWN

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_WIRED\_EXTERNAL\_POWER\_SOURCE\_RFU

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_WIRELESS\_EXTERNAL\_POWER\_SOURCE\_NOTCONNECT

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_WIRELESS\_EXTERNAL\_POWER\_SOURCE\_CONNECTED

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_WIRELESS\_EXTERNAL\_POWER\_SOURCE\_UNKNOWN

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_WIRELESS\_EXTERNAL\_POWER\_SOURCE\_RFU

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_CHARGE\_STATE\_UNKNOWN

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_CHARGE\_STATE\_CHARGING

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_CHARGE\_STATE\_DISCHARGING\_ACTIVE

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_CHARGE\_STATE\_DISCHARGING\_INACTIVE

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_CHARGE\_LEVEL\_UNKNOWN

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_CHARGE\_LEVEL\_GOOD

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_CHARGE\_LEVEL\_LOW

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_BATTERY\_CHARGE\_LEVEL\_CRITICAL

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_TYPE\_UNKNOWN

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_TYPE\_CURRENT

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_TYPE\_VOLTAGE

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_TYPE\_TRICKLE

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_TYPE\_FLOAT

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_TYPE\_RFU

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_FAULT\_NONE

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_FAULT\_BATTERY

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_FAULT\_EXTERNAL\_POWER\_SOURCE

BAS\_CHR\_LEVEL\_STATUS\_POWER\_STATE\_CHARGE\_FAULT\_OTHER

BAS\_CHR\_LEVEL\_STATUS\_ASSITIONAL\_STATUS\_SERVICE\_FALSE

BAS\_CHR\_LEVEL\_STATUS\_ASSITIONAL\_STATUS\_SERVICE\_TRUE

BAS\_CHR\_LEVEL\_STATUS\_ASSITIONAL\_STATUS\_SERVICE\_UNKNOWN

BAS\_CHR\_LEVEL\_STATUS\_ASSITIONAL\_STATUS\_SERVICE\_RFU

BAS\_CHR\_LEVEL\_STATUS\_ASSITIONAL\_STATUS\_BATTERY\_FAULT\_UNKNOWN

BAS\_CHR\_LEVEL\_STATUS\_ASSITIONAL\_STATUS\_BATTERY\_FAULT\_TRUE

BAS\_CHR\_CRITICAL\_STATUS\_FLAGS\_BM\_NONE

BAS\_CHR\_CRITICAL\_STATUS\_FLAGS\_BM\_CRITICAL\_POWER\_STATE

BAS\_CHR\_CRITICAL\_STATUS\_FLAGS\_BM\_IMMEDIATE\_SERVICE

BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_NONE

BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_EXTERNAL\_SOURCE\_POWER

BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_VOLTAGE

BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_AVAILABLE\_ENERGY

BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_AVAILABLE\_BATTERY\_CAPACITY

BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_CHARGE\_RATE

BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_AVAILABLE\_ENERGY\_LAST\_CHARGE

BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_RFU

BAS\_CHR\_TIME\_STATUS\_FLAGS\_BM\_NONE

BAS\_CHR\_TIME\_STATUS\_FLAGS\_BM\_DISCHARGED\_STANDBY

BAS\_CHR\_TIME\_STATUS\_FLAGS\_BM\_RECHARGE

BAS\_CHR\_TIME\_STATUS\_FLAGS\_BM\_RFU

BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_NONE

BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_BATTERY\_HEALTH\_SUMMARY

BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_RCYCLE\_COUNT

BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_CURRENT\_TEMPERATURE

BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_DEEP\_DISCHARGE\_COUNT

BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_RFU

BAS\_CHR\_HEALTH\_INFO\_FLAGS\_BM\_NONE

BAS\_CHR\_HEALTH\_INFO\_FLAGS\_BM\_CYCLE\_COUNT\_DESIGNED\_LIFETIME

BAS\_CHR\_HEALTH\_INFO\_FLAGS\_BM\_DESIGNED\_OPERATING\_TEMPERATURE

BAS\_CHR\_HEALTH\_INFO\_FLAGS\_BM\_RFU

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_NONE

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_MANUFACTURE\_DATE

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_EXPIRATION\_DATE

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_DESIGNED\_CAPACITY

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_LOW\_ENERGY

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_CRITICAL\_ENERGY

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_CHEMISTRY

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_NOMINAL\_VOLTAGE

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_AGGREGATION\_GROUP

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_RFU

BAS\_CHR\_BATTERY\_INFO\_FEATURE\_BM\_NONE

**BAS\_CHR\_BATTERY\_INFO\_FEATURE\_BM\_REPLACE**

**BAS\_CHR\_BATTERY\_INFO\_FEATURE\_BM\_RECHARGE**

**BAS\_CHR\_BATTERY\_INFO\_FEATURE\_BM\_RFU**

### 3.2.3 设备信息服务

设备信息服务公开关于设备的制造商和/或供应商信息。

#### 示例

[bluetooth/ble\\_services/ble\\_dis.](#)

#### API 参考

##### Header File

- [components/bluetooth/ble\\_services/dis/include/esp\\_dis.h](#)

##### Functions

**esp\_err\_t esp\_ble\_dis\_get\_model\_number** (const char \*\*value)

Get the model number that is assigned by the device vendor.

**参数 value** –[out] The pointer to store the model number.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong model number

**esp\_err\_t esp\_ble\_dis\_set\_model\_number** (const char \*value)

Set the model number of the device.

**参数 value** –[in] The pointer to store the model number.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong model number

**esp\_err\_t esp\_ble\_dis\_get\_serial\_number** (const char \*\*value)

Get the serial number for a particular instance of the device.

**参数 value** –[out] The pointer to store the serial number.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong serial number

**esp\_err\_t esp\_ble\_dis\_set\_serial\_number** (const char \*value)

Set the serial number of the device.

**参数 value** –[in] The pointer to store the serial number.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong serial number

`esp_err_t esp_ble_dis_get_firmware_revision (const char **value)`

Get the firmware revision for the firmware within the device.

**参数** `value` –[out] The pointer to store the firmware revision.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong firmware revision

`esp_err_t esp_ble_dis_set_firmware_revision (const char *value)`

Set the firmware revision of the device.

**参数** `value` –[in] The pointer to store the firmware revision.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong firmware revision

`esp_err_t esp_ble_dis_get_hardware_revision (const char **value)`

Get the hardware revision for the hardware within the device.

**参数** `value` –[out] The pointer to store the hardware revision.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong hardware revision

`esp_err_t esp_ble_dis_set_hardware_revision (const char *value)`

Set the hardware revision of the device.

**参数** `value` –[in] The pointer to store the hardware revision.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong hardware revision

`esp_err_t esp_ble_dis_get_software_revision (const char **value)`

Get the software revision for the software within the device.

**参数** `value` –[out] The pointer to store the software revision.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong software revision

`esp_err_t esp_ble_dis_set_software_revision (const char *value)`

Set the software revision of the device.

**参数** `value` –[in] The pointer to store the software revision.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong software revision

`esp_err_t esp_ble_dis_get_manufacturer_name (const char **value)`

Get the name of the manufacturer of the device.

**参数** `value` –[out] The pointer to store the name of the manufacturer.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong name of the manufacturer

`esp_err_t esp_ble_dis_set_manufacturer_name (const char *value)`

Set the name of the manufacturer of the device.

**参数** `value` –[in] The pointer to store the name of the manufacturer.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong name of the manufacturer

esp\_err\_t **esp\_ble\_dis\_get\_system\_id** (const char \*\*value)

Get the System Id of the device.

**参数 value** –[out] The pointer to store the System Id.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong System Id

esp\_err\_t **esp\_ble\_dis\_set\_system\_id** (const char \*value)

Set the System Id of the device.

**参数 value** –[in] The pointer to store the System Id.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong System Id

esp\_err\_t **esp\_ble\_dis\_get\_pnp\_id** (*esp\_ble\_dis\_pnp\_t* \*pnp\_id)

Get the PnP Id of the device.

**参数 pnp\_id** –[in] The pointer to store the PnP Id.

**返回 :**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong PnP Id

esp\_err\_t **esp\_ble\_dis\_set\_pnp\_id** (*esp\_ble\_dis\_pnp\_t* \*pnp\_id)

Set the PnP Id of the device.

**参数 pnp\_id** –[in] The pointer to store the PnP Id.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong PnP Id

esp\_err\_t **esp\_ble\_dis\_init** (void)

Initialization GATT Device Information Service.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

## Structures

struct **esp\_ble\_dis\_pnp**

The structure represent a set of values that are used to create a device ID value. These values are used to identify all devices of a given type/model/version using numbers.

## Public Members

uint8\_t **src**

The vendor ID source

uint16\_t **vid**

The product vendor from the namespace in the vendor ID source

uint16\_t **pid**

Manufacturer managed identifier for this product



uint16\_t **ver**

Manufacturer managed version for this product

struct **esp\_ble\_dis\_data**

Structure holding data for the main characteristics

### Public Members

char **model\_number**[CONFIG\_BLE\_DIS\_STR\_MAX]

Model number. Represent the model number that is assigned by the device vendor.

char **serial\_number**[CONFIG\_BLE\_DIS\_STR\_MAX]

Serial number. Represent the serial number for a particular instance of the device.

char **firmware\_revision**[CONFIG\_BLE\_DIS\_STR\_MAX]

Firmware revision. Represent the firmware revision for the firmware within the device.

char **hardware\_revision**[CONFIG\_BLE\_DIS\_STR\_MAX]

Hardware revision. Represent the hardware revision for the hardware within the device.

char **software\_revision**[CONFIG\_BLE\_DIS\_STR\_MAX]

Software revision. Represent the software revision for the software within the device.

char **manufacturer\_name**[CONFIG\_BLE\_DIS\_STR\_MAX]

Manufacturer name. Represent the name of the manufacturer of the device.

char **system\_id**[CONFIG\_BLE\_DIS\_STR\_MAX]

System ID. Represent the System Id of the device.

*esp\_ble\_dis\_pnp\_t* **pnp\_id**

PnP ID. Represent the PnP Id of the device.

### Macros

**BLE\_DIS\_UUID16**

**BLE\_DIS\_CHR\_UUID16\_SYSTEM\_ID**

**BLE\_DIS\_CHR\_UUID16\_MODEL\_NUMBER**

**BLE\_DIS\_CHR\_UUID16\_SERIAL\_NUMBER**

**BLE\_DIS\_CHR\_UUID16\_FIRMWARE\_REVISION**

**BLE\_DIS\_CHR\_UUID16\_HARDWARE\_REVISION**

**BLE\_DIS\_CHR\_UUID16\_SOFTWARE\_REVISION**

**BLE\_DIS\_CHR\_UUID16\_MANUFACTURER\_NAME**

**BLE\_DIS\_CHR\_UUID16\_REG\_CERT**

**BLE\_DIS\_CHR\_UUID16\_PNP\_ID**

### Type Definitions

typedef struct *esp\_ble\_dis\_pnp* **esp\_ble\_dis\_pnp\_t**

The structure represent a set of values that are used to create a device ID value. These values are used to identify all devices of a given type/model/version using numbers.

typedef struct *esp\_ble\_dis\_data* **esp\_ble\_dis\_data\_t**

Structure holding data for the main characteristics

## 3.2.4 心率服务

心率服务公开心率和其他与用于健身应用的心率传感器相关的数据。

### 示例

[bluetooth/ble\\_services/ble\\_hrs.](#)

### API 参考

#### Header File

- [components/bluetooth/ble\\_services/hrs/include/esp\\_hrs.h](#)

#### Functions

**esp\_err\_t esp\_ble\_hrs\_get\_location** (uint8\_t \*location)

Get the sensor location value of the device.

**参数 location** –[in] The pointer to store the sensor location value

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong location

**esp\_err\_t esp\_ble\_hrs\_set\_location** (uint8\_t location)

Set the sensor location value of the device.

**参数 location** –[in] The sensor location value

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong location

**esp\_err\_t esp\_ble\_hrs\_get\_hrm** (*esp\_ble\_hrs\_hrm\_t* \*hrm)

Get the value of the heart rate measurement of the device.

**参数 hrm** –[in] The pointer to store the value of the heart rate measurement

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong heart rate measurement

`esp_err_t esp_ble_hrs_set_hrm(esp_ble_hrs_hrm_t *hrm)`

Set the value of the heart rate measurement of the device.

**参数** `hrm` –[in] The pointer to store the value of the heart rate measurement

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong heart rate measurement

`esp_err_t esp_ble_hrs_init(void)`

Initialization Heart Rate Service.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

## Structures

struct `esp_ble_hrs_hrm_t`

Heart Rate Measurement Characteristic.

## Public Members

`uint8_t format`

Heart rate value format flag

`uint8_t detected`

Sensor contact detected flag

`uint8_t supported`

Sensor contact supported flag

`uint8_t energy`

Energy expended present flag

`uint8_t interval`

RR-Interval present flag

`uint8_t reserved`

Reserved for future use flag

struct `esp_ble_hrs_hrm_t::[anonymous] flags`

Flags of heart rate measurement

`uint8_t u8`

8 bit resolution

`uint16_t u16`

16 bit resolution

union `esp_ble_hrs_hrm_t::[anonymous] heartrate`

Heart rate measurement value

uint16\_t **energy\_val**

Expended energy value

uint16\_t **interval\_buf**[BLE\_HRS\_CHR\_MERSUREMENT\_RR\_INTERVAL\_MAX\_NUM]

The RR-Interval value represents the time between two R-Wave detections

### Macros

**BLE\_HRS\_UUID16**

**BLE\_HRS\_CHR\_UUID16\_MEASUREMENT**

**BLE\_HRS\_CHR\_UUID16\_BODY\_SENSOR\_LOC**

**BLE\_HRS\_CHR\_UUID16\_HEART\_RATE\_CNTL\_POINT**

**BLE\_HRS\_CHR\_MERSUREMENT\_RR\_INTERVAL\_MAX\_NUM**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_OTHER**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_CHEST**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_WRIST**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_FINGER**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_HAND**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_EAR\_LOBE**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_FOOT**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_RFU**

**BLE\_HRS\_CHR\_BODY\_SENSOR\_LOC\_MAX**

## 3.2.5 健康温度计服务

健康温度计服务公开与用于医疗保健应用程序的温度计相关的温度和其他数据。

### 示例

[bluetooth/ble\\_services/ble\\_hts](#).

## API 参考

## Header File

- [components/bluetooth/ble\\_services/hts/include/esp\\_hts.h](#)

## Functions

`esp_err_t esp_ble_hts_get_temp_type (uint8_t *temp_type)`

Get the current temperature type value of the device.

参数 **temp\_type** –[in] The pointer to store the current temperature type value

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong temperature type

`esp_err_t esp_ble_hts_set_temp_type (uint8_t temp_type)`

Set the current temperature type value of the device.

参数 **temp\_type** –[in] The current temperature type value

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong temperature type

`esp_err_t esp_ble_hts_get_measurement_temp (esp_ble_hts_temp_t *temp_val)`

Get the value of the temperature measurement of the device.

参数 **temp\_val** –[in] The pointer to store the value of the temperature measurement

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong temperature measurement

`esp_err_t esp_ble_hts_set_measurement_temp (esp_ble_hts_temp_t *temp_val)`

Set the value of the temperature measurement of the device.

参数 **temp\_val** –[in] The pointer to store the value of the temperature measurement

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong temperature measurement

`esp_err_t esp_ble_hts_get_intermediate_temp (esp_ble_hts_temp_t *temp_val)`

Get the value of the intermediate temperature of the device.

参数 **temp\_val** –[in] The pointer to store the value of the intermediate temperature

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong intermediate temperature

`esp_err_t esp_ble_hts_set_intermediate_temp (esp_ble_hts_temp_t *temp_val)`

Set the value of the intermediate temperature of the device.

参数 **temp\_val** –[in] The pointer to store the value of the intermediate temperature

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong intermediate temperature

`esp_err_t esp_ble_hts_get_measurement_interval (uint16_t *interval_val)`

Get the measurement interval value of the device.

参数 **interval\_val** –[in] The pointer to store the measurement interval value

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong measurement interval

esp\_err\_t **esp\_ble\_hts\_set\_measurement\_interval** (uint16\_t interval\_val)

Set the measurement interval value of the device.

参数 **interval\_val** –[in] The measurement interval value

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong measurement interval

esp\_err\_t **esp\_ble\_hts\_init** (void)

Initialization Health Thermometer Service.

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

## Structures

struct **esp\_ble\_hts\_temp\_t**

Temperature Measurement and Intermediate Temperature Characteristic.

### Public Members

uint8\_t **temperature\_unit**

Temperature units flag

uint8\_t **time\_stamp**

Time stamp flag

uint8\_t **temperature\_type**

Temperature type flag

uint8\_t **reserved**

Reserved for future use

struct *esp\_ble\_hts\_temp\_t*::[anonymous] **flags**

Flags of temperature

uint32\_t **celsius**

Celsius unit

uint32\_t **fahrenheit**

Fahrenheit unit

union *esp\_ble\_hts\_temp\_t*::[anonymous] **temperature**

Temperature value

uint16\_t **year**

Year as defined by the Gregorian calendar, Valid range 1582 to 9999

uint8\_t **month**

Month of the year as defined by the Gregorian calendar, Valid range 1 (January) to 12 (December)

`uint8_t day`

Day of the month as defined by the Gregorian calendar, Valid range 1 to 31

`uint8_t hours`

Number of hours past midnight, Valid range 0 to 23

`uint8_t minutes`

Number of minutes since the start of the hour. Valid range 0 to 59

`uint8_t seconds`

Number of seconds since the start of the minute. Valid range 0 to 59

struct *esp\_ble\_hts\_temp\_t*::[anonymous] `timestamp`

The date and time

`uint8_t location`

The location of a temperature measurement

## Macros

`BLE-HTS-UUID16`

`BLE-HTS-CHR-UUID16-TEMPERATURE-MEASUREMENT`

`BLE-HTS-CHR-UUID16-TEMPERATURE-TYPE`

`BLE-HTS-CHR-UUID16-INTERMEDIATE-TEMPERATURE`

`BLE-HTS-CHR-UUID16-MEASUREMENT-INTERVAL`

`BLE-HTS-CHR-TEMPERATURE-UNITS-CELSIUS`

`BLE-HTS-CHR-TEMPERATURE-UNITS-FAHRENHEIT`

`BLE-HTS-CHR-TEMPERATURE-FLAGS-NOT`

`BLE-HTS-CHR-TEMPERATURE-FLAGS-SET`

`BLE-HTS-CHR-TEMPERATURE-TYPE-RFU`

`BLE-HTS-CHR-TEMPERATURE-TYPE-ARMPIT`

`BLE-HTS-CHR-TEMPERATURE-TYPE-BODY`

`BLE-HTS-CHR-TEMPERATURE-TYPE-EAR`

`BLE-HTS-CHR-TEMPERATURE-TYPE-FINGER`

`BLE_HTS_CHR_TEMPERATURE_TYPE_GAST_TRACT`

`BLE_HTS_CHR_TEMPERATURE_TYPE_MOUTH`

`BLE_HTS_CHR_TEMPERATURE_TYPE_RECTUM`

`BLE_HTS_CHR_TEMPERATURE_TYPE_TOE`

`BLE_HTS_CHR_TEMPERATURE_TYPE_TYMPANUM`

`BLE_HTS_CHR_TEMPERATURE_TYPE_MAX`

### 3.2.6 TX 电源服务

TX 功率服务在连接时公开设备的当前发射功率水平。

#### 示例

[bluetooth/ble\\_services/ble\\_tps.](#)

#### API 参考

##### Header File

- [components/bluetooth/ble\\_services/tps/include/esp\\_tps.h](#)

##### Functions

`int8_t esp_ble_tps_get_tx_power_level` (void)

Get the TX Power Level of the device.

返回 : The TX Power Level of the device

`esp_err_t esp_ble_tps_set_tx_power_level` (int8\_t tx\_power\_level)

Set the TX Power Level of the device.

参数 `tx_power_level` –[in] The TX Power Level of the device

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong TX Power Level

`esp_err_t esp_ble_tps_init` (void)

Initialization TX Power Service.

返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

##### Macros

`BLE_TPS_UUID16`

`BLE_TPS_CHR_UUID16_TX_POWER_LEVEL`



## 3.3 BLE 配置文件

### 3.3.1 警报通知配置文件

警报通知配置文件允许像手表这样的设备从设备中获取信息手机的来电、未接电话和短信/彩信信息。信息包括来电时的来电显示或电邮/短讯/彩信的发送人的显示，但不是信息。该配置文件还使客户端设备能够获取有关服务器设备上未读消息数。

#### 示例

[bluetooth/ble\\_profiles/ble\\_anp](#).

#### API 参考

##### Header File

- [components/bluetooth/ble\\_profiles/std/ble\\_anp/include/esp\\_anp.h](#)

##### Functions

`esp_err_t esp_ble_anp_get_new_alert` (uint8\_t cat\_id, uint8\_t \*cat\_val)

Read the value of or check supported new alert category.

**Attention** 1. When cat\_id is 0xFF, read the value of supported new alert category.

**Attention** 2. When cat\_id isn't 0xFF, check supported new alert category is enable or disable.

##### 参数

- **cat\_id** –[in] The ID of the category to read or check
- **cat\_val** –[out] The value of read or check supported new alert category

##### 返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong category of the alert

`esp_err_t esp_ble_anp_set_new_alert` (uint8\_t cat\_id, [esp\\_ble\\_anp\\_option\\_t](#) option)

Request or recovery supported new alert notification to the given category.

**Attention** 1. When cat\_id is 0xFF, recover for all supported new alert category to get the current message counts.

**Attention** 2. When cat\_id isn't 0xFF, request for a supported new alert category to get the current message counts.

##### 参数

- **cat\_id** –[in] The ID of the category to request or recover the notification to
- **option** –[in] Disable or enable supported new alert category

##### 返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong category of the alert

esp\_err\_t **esp\_ble\_anp\_get\_unr\_alert** (uint8\_t cat\_id, uint8\_t \*cat\_val)

Read the value of or check supported unread alert status category.

**Attention** 1. When cat\_id is 0xFF, read the value of supported unread alert status category.

**Attention** 2. When cat\_id isn't 0xFF, check supported unread alert status category is enable or disable.

#### 参数

- **cat\_id** –[in] The ID of the category to read or check
- **cat\_val** –[out] The value of read or check supported unread alert status category

#### 返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong category of the alert

esp\_err\_t **esp\_ble\_anp\_set\_unr\_alert** (uint8\_t cat\_id, *esp\_ble\_anp\_option\_t* option)

Request or recovery supported unread alert status notification to the given category.

**Attention** 1. When cat\_id is 0xFF, recover for all supported unread alert status category to get the current message counts.

**Attention** 2. When cat\_id isn't 0xFF, request for an supported unread alert status category to get the current message counts.

#### 参数

- **cat\_id** –[in] The ID of the category to request or recover the notification to
- **option** –[in] Disable or enable supported unread alert status category

#### 返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong category of the alert

esp\_err\_t **esp\_ble\_anp\_init** (void)

Initialization GATT Alert Notification Profile.

#### 返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

esp\_err\_t **esp\_ble\_anp\_deinit** (void)

Deinitialization GATT Alert Notification Profile.

#### 返回

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

## Structures

struct **esp\_ble\_anp\_data\_t**

The status of the new or unread alert.

## Public Members

uint8\_t **cat\_id**

The predefined categories of unread alerts and messages

The predefined categories of new alerts and messages

`uint8_t count`

The number of unread alerts in the server ranging from 0 to 255

The number of new alerts in the server ranging from 0 to 255

struct *esp\_ble\_anp\_data\_t*::[anonymous]::[anonymous] **unr\_alert\_stat**

The status of unread alerts

`uint8_t cat_info`[BLE\_ANP\_INFO\_STR\_MAX\_LEN]

The brief text information for the last alert

struct *esp\_ble\_anp\_data\_t*::[anonymous]::[anonymous] **new\_alert\_val**

The status of new alerts

union *esp\_ble\_anp\_data\_t*::[anonymous] [**anonymous**]

Alert notification status

## Macros

**BLE\_ANP\_UUID16**

**BLE\_ANP\_CHR\_UUID16\_SUP\_NEW\_ALERT\_CAT**

**BLE\_ANP\_CHR\_UUID16\_NEW\_ALERT**

**BLE\_ANP\_CHR\_UUID16\_SUP\_UNR\_ALERT\_CAT**

**BLE\_ANP\_CHR\_UUID16\_UNR\_ALERT\_STAT**

**BLE\_ANP\_CHR\_UUID16\_ALERT\_NOT\_CTRL\_PT**

**BLE\_ANP\_CAT\_BM\_NONE**

**BLE\_ANP\_CAT\_BM\_SIMPLE\_ALERT**

**BLE\_ANP\_CAT\_BM\_EMAIL**

**BLE\_ANP\_CAT\_BM\_NEWS**

**BLE\_ANP\_CAT\_BM\_CALL**

**BLE\_ANP\_CAT\_BM\_MISSED\_CALL**

**BLE\_ANP\_CAT\_BM\_SMS**

**BLE\_ANP\_CAT\_BM\_VOICE\_MAIL**

**BLE\_ANP\_CAT\_BM\_SCHEDULE**

**BLE\_ANP\_CAT\_ID\_SIMPLE\_ALERT**

**BLE\_ANP\_CAT\_ID\_EMAIL**

**BLE\_ANP\_CAT\_ID\_NEWS**

**BLE\_ANP\_CAT\_ID\_CALL**

**BLE\_ANP\_CAT\_ID\_MISSED\_CALL**

**BLE\_ANP\_CAT\_ID\_SMS**

**BLE\_ANP\_CAT\_ID\_VOICE\_MAIL**

**BLE\_ANP\_CAT\_ID\_SCHEDULE**

**BLE\_ANP\_CAT\_NUM**

**BLE\_ANP\_CMD\_EN\_NEW\_ALERT\_CAT**

**BLE\_ANP\_CMD\_EN\_UNR\_ALERT\_CAT**

**BLE\_ANP\_CMD\_DIS\_NEW\_ALERT\_CAT**

**BLE\_ANP\_CMD\_DIS\_UNR\_ALERT\_CAT**

**BLE\_ANP\_CMD\_NOT\_NEW\_ALERT\_IMMEDIATE**

**BLE\_ANP\_CMD\_NOT\_UNR\_ALERT\_IMMEDIATE**

**BLE\_ANP\_INFO\_STR\_MAX\_LEN**

**BLE\_ANP\_NEW\_ALERT\_MAX\_LEN**

### Enumerations

enum **esp\_ble\_anp\_option\_t**

The option of the new or unread alert.

*Values:*

enumerator **BLE\_ANP\_OPT\_ENABLE**

enumerator **BLE\_ANP\_OPT\_DISABLE**

enumerator **BLE\_ANP\_OPT\_RECOVER**

### 3.3.2 心率配置文件

心率配置文件用于使数据收集设备能够从公开心率服务的心率传感器获取数据。

#### 示例

[bluetooth/ble\\_profiles/ble\\_hrp.](#)

#### API 参考

##### Header File

- [components/bluetooth/ble\\_profiles/std/ble\\_hrp/include/esp\\_hrp.h](#)

##### Functions

`esp_err_t esp_ble_hrp_get_location (uint8_t *location)`

Get the sensor location value of the device.

**参数** `location` –[in] The pointer to store the sensor location value

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery level

`esp_err_t esp_ble_hrp_get_ctrl (uint8_t *cmd_id)`

Get the control point value of the device.

**参数** `cmd_id` –[in] The pointer to store the control point value

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery level

`esp_err_t esp_ble_hrp_set_ctrl (uint8_t cmd_id)`

Set the control point value of the device.

**参数** `cmd_id` –[in] The control point value

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery level

`esp_err_t esp_ble_hrp_init (void)`

Initialization Heart Rate Profile.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

`esp_err_t esp_ble_hrp_deinit (void)`

Deinitialization Heart Rate Profile.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

## Structures

struct **esp\_ble\_hrp\_data\_t**

Heart Rate Measurement Characteristic.

### Public Members

uint8\_t **format**

Heart rate value format flag

uint8\_t **detected**

Sensor contact detected flag

uint8\_t **supported**

Sensor contact supported flag

uint8\_t **energy**

Energy expended present flag

uint8\_t **interval**

RR-Interval present flag

uint8\_t **reserved**

Reserved for future use flag

struct *esp\_ble\_hrp\_data\_t*::[anonymous] **flags**

Flags of heart rate measurement

uint8\_t **u8**

8 bit resolution

uint16\_t **u16**

16 bit resolution

union *esp\_ble\_hrp\_data\_t*::[anonymous] **heartrate**

Heart rate measurement value

uint16\_t **energy\_val**

Expended energy value

uint16\_t **interval\_buf**[BLE\_HRP\_CHR\_MERSUREMENT\_RR\_INTERVAL\_MAX\_NUM]

The RR-Interval value represents the time between two R-Wave detections

## Macros

**BLE\_HRP\_UUID16**

**BLE\_HRP\_CHR\_UUID16\_MEASUREMENT**

BLE\_HRP\_CHR\_UUID16\_BODY\_SENSOR\_LOC

BLE\_HRP\_CHR\_UUID16\_HEART\_RATE\_CNTL\_POINT

BLE\_HRP\_CHR\_MERSUREMENT\_RR\_INTERVAL\_MAX\_NUM

BLE\_HRP\_FLAGS\_BM\_NONE

BLE\_HRP\_FLAGS\_BM\_FORMAT

BLE\_HRP\_FLAGS\_BM\_SENSOR\_CONTACT\_DETECTED

BLE\_HRP\_FLAGS\_BM\_SENSOR\_CONTACT\_SUPPOTRED

BLE\_HRP\_FLAGS\_BM\_ENERGY

BLE\_HRP\_FLAGS\_BM\_RR\_INTERVAL

BLE\_HRP\_FLAGS\_BM\_RFU

BLE\_HRP\_CHR\_MERSUREMENT\_FLAGS\_FORMAT\_U8

BLE\_HRP\_CHR\_MERSUREMENT\_FLAGS\_FORMAT\_U16

BLE\_HRP\_CHR\_MERSUREMENT\_FLAGS\_NOT

BLE\_HRP\_CHR\_MERSUREMENT\_FLAGS\_SET

BLE\_HRP\_CMD\_RFU

BLE\_HRP\_CMD\_RESET\_ENERGY\_EXPENDED

BLE\_HRP\_CMD\_MAX

### 3.3.3 健康温度计配置文件

健康温度计配置文件用于使数据收集设备能够从公开健康温度计服务的温度计传感器获取数据。

#### 示例

[bluetooth/ble\\_profiles/ble\\_http](#).

## API 参考

### Header File

- [components/bluetooth/ble\\_profiles/std/ble\\_http/include/esp\\_http.h](#)

### Functions

`esp_err_t esp_ble_http_get_temp_type (uint8_t *temp_type)`

Get the current temperature type value of the device.

**参数** `temp_type` –[in] The pointer to store the current temperature type value

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery level

`esp_err_t esp_ble_http_get_measurement_interval (uint16_t *interval_val)`

Get the measurement interval value of the device.

**参数** `interval_val` –[in] The pointer to store the measurement interval value

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery level

`esp_err_t esp_ble_http_set_measurement_interval (uint16_t interval_val)`

Set the measurement interval value of the device.

**参数** `interval_val` –[in] The measurement interval value

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong battery level

`esp_err_t esp_ble_http_init (void)`

Initialization Health Thermometer Profile.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

`esp_err_t esp_ble_http_deinit (void)`

Deinitialization Health Thermometer Profile.

**返回**

- ESP\_OK on successful
- ESP\_ERR\_INVALID\_ARG on wrong initialization
- ESP\_FAIL on error

### Structures

`struct esp_ble_http_data_t`

Temperature Measurement and Intermediate Temperature Characteristic.

### Public Members

`uint8_t temperature_unit`

Temperature units flag



uint8\_t **time\_stamp**

Time stamp flag

uint8\_t **temperature\_type**

Temperature type flag

uint8\_t **reserved**

Reserved for future use

struct *esp\_ble\_htp\_data\_t*::[anonymous] **flags**

Flags of temperature

uint32\_t **celsius**

Celsius unit

uint32\_t **fahrenheit**

Fahrenheit unit

union *esp\_ble\_htp\_data\_t*::[anonymous] **temperature**

Temperature value

uint16\_t **year**

Year as defined by the Gregorian calendar, Valid range 1582 to 9999

uint8\_t **month**

Month of the year as defined by the Gregorian calendar, Valid range 1 (January) to 12 (December)

uint8\_t **day**

Day of the month as defined by the Gregorian calendar, Valid range 1 to 31

uint8\_t **hours**

Number of hours past midnight, Valid range 0 to 23

uint8\_t **minutes**

Number of minutes since the start of the hour. Valid range 0 to 59

uint8\_t **seconds**

Number of seconds since the start of the minute. Valid range 0 to 59

struct *esp\_ble\_htp\_data\_t*::[anonymous] **timestamp**

The date and time

uint8\_t **location**

The location of a temperature measurement

## Macros

**BLE\_HTP\_UUID16**

BLE\_HTP\_CHR\_UUID16\_TEMPERATURE\_MEASUREMENT

BLE\_HTP\_CHR\_UUID16\_TEMPERATURE\_TYPE

BLE\_HTP\_CHR\_UUID16\_INTERMEDIATE\_TEMPERATURE

BLE\_HTP\_CHR\_UUID16\_MEASUREMENT\_INTERVAL

BLE\_HTP\_FLAGS\_BM\_NONE

BLE\_HTP\_FLAGS\_BM\_TEMPERATURE\_UNITS

BLE\_HTP\_FLAGS\_BM\_TIME\_STAMP

BLE\_HTP\_FLAGS\_BM\_TEMPERATURE\_TYPE

BLE\_HTP\_FLAGS\_BM\_RFU

BLE\_HTP\_CHR\_TEMPERATURE\_UNITS\_CELSIUS

BLE\_HTP\_CHR\_TEMPERATURE\_UNITS\_FAHRENHEIT

BLE\_HTP\_CHR\_TEMPERATURE\_FLAGS\_NOT

BLE\_HTP\_CHR\_TEMPERATURE\_FLAGS\_SET

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_RFU

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_ARMPIT

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_BODY

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_EAR

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_FINGER

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_GAST\_TRACT

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_MOUTH

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_RECTUM

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_TOE

BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_TYMPANUM

**BLE\_HTP\_CHR\_TEMPERATURE\_TYPE\_MAX**

## 3.4 BLE HCI 组件

BLE HCI 组件用于通过 VHCI 接口直接操作 BLE Controller 实现广播, 扫描等功能。相比于通过 Nimble 或 Bluedroid 协议栈发起广播和扫描, 使用该组件有如下优点: - 更少的内存占用 - 更小的固件尺寸 - 更快的初始化流程

### 3.4.1 BLE HCI 使用方法

对于广播应用: 1. 初始化 BLE HCI: 使用`ble_hci_init()` 函数进行初始化。2. 设定本机随机地址 (可选): 如果需要使用随机地址作为广播地址, 使用`cpp:func:ble_hci_set_random_address`函数进行设定。3. 配置广播参数: 使用`cpp:func:ble_hci_set_adv_param` 配置广播参数。4. 配置广播数据: 使用`ble_hci_set_adv_data()` 设定需要广播的数据内容。5. 开始广播: 使用`ble_hci_set_adv_enable()`。

对于扫描应用: 1. 初始化 BLE HCI: 使用`ble_hci_init()` 函数进行初始化。2. 配置扫描参数: 使用`ble_hci_set_scan_param()` 配置扫描参数。3. 使能 meta 事件: 使用`ble_hci_enable_meta_event()` 使能中断事件。4. 注册扫描事件函数: 使用`ble_hci_set_register_scan_callback()` 注册中断事件。5. 开始扫描: 使用`ble_hci_set_scan_enable()`。

### 3.4.2 API 参考

#### Header File

- [components/bluetooth/ble\\_hci/include/ble\\_hci.h](#)

#### Functions

`esp_err_t ble_hci_init (void)`

BLE HCI initialization.

返回 `esp_err_t`

- ESP\_OK: succeed
- others: fail

`esp_err_t ble_hci_deinit (void)`

BLE HCI de-initialization.

返回 `esp_err_t`

- ESP\_OK: succeed
- others: fail

`esp_err_t ble_hci_reset (void)`

BLE HCI reset controller.

返回 `esp_err_t`

- ESP\_OK: succeed
- others: fail

`esp_err_t ble_hci_enable_meta_event` (void)

Enable BLE HCI meta event.

返回 `esp_err_t`

`esp_err_t ble_hci_set_adv_param` ([ble\\_hci\\_adv\\_param\\_t](#) \*param)

Set BLE HCI advertising parameters.

参数 **param** -: advertising parameters

返回 `esp_err_t`

- `ESP_OK`: succeed
- others: fail

`esp_err_t ble_hci_set_adv_data` (uint8\_t len, uint8\_t \*data)

Set BLE HCI advertising data.

参数

- **len** -: advertising data length
- **data** -: advertising data

返回 `esp_err_t`

- `ESP_OK`: succeed
- others: fail

`esp_err_t ble_hci_set_adv_enable` (bool enable)

Set BLE HCI advertising enable.

参数 **enable** -: true for enable advertising

返回 `esp_err_t`

- `ESP_OK`: succeed
- others: fail

`esp_err_t ble_hci_set_scan_param` ([ble\\_hci\\_scan\\_param\\_t](#) \*param)

Set BLE HCI scan parameters.

参数 **param** -: scan parameters

返回 `esp_err_t`

- `ESP_OK`: succeed
- others: fail

`esp_err_t ble_hci_set_scan_enable` (bool enable, bool filter\_duplicates)

Set BLE HCI scan enable.

参数

- **enable** -: enable or disable scan
- **filter\_duplicates** -: filter duplicates or not

返回 `esp_err_t`

- `ESP_OK`: succeed
- others: fail

`esp_err_t ble_hci_set_register_scan_callback` ([ble\\_hci\\_scan\\_cb\\_t](#) cb)

Set BLE HCI scan callback.

参数 **cb** -: scan callback function pointer

返回 `esp_err_t`

- `ESP_OK`: succeed
- others: fail

`esp_err_t ble_hci_add_to_accept_list` ([ble\\_hci\\_addr\\_t](#) addr, [ble\\_hci\\_addr\\_type\\_t](#) addr\_type)

Add BLE white list.

参数

- **addr** -: address to be added to white list
- **addr\_type** -: address type to be added to white list

返回 `esp_err_t`

- ESP\_OK: succeed
- others: fail

esp\_err\_t **ble\_hci\_clear\_accept\_list** (void)

Clear BLE white list.

返回 esp\_err\_t

- ESP\_OK: succeed
- others: fail

esp\_err\_t **ble\_hci\_set\_random\_address** (*ble\_hci\_addr\_t* addr)

Set BLE owner address.

参数 **addr** -: owner address

返回

- ESP\_OK: succeed
- others: fail

## Structures

struct **ble\_hci\_scan\_result\_t**

BLE scan result struct.

## Public Members

*ble\_hci\_search\_evt\_t* **search\_evt**

Search event type

*ble\_hci\_dev\_type\_t* **dev\_type**

Device type

*ble\_hci\_addr\_t* **bda**

Bluetooth device address which has been searched

*ble\_hci\_addr\_type\_t* **ble\_addr\_type**

Ble device address type

uint8\_t **ble\_adv**[ESP\_BLE\_ADV\_DATA\_LEN\_MAX + ESP\_BLE\_SCAN\_RSP\_DATA\_LEN\_MAX]

Received EIR

uint8\_t **adv\_data\_len**

Adv data length

uint8\_t **scan\_rsp\_len**

Scan response length

int **rssi**

Searched device's RSSI

struct **ble\_hci\_adv\_param\_t**

Ble adv parameters.

**Public Members****uint16\_t adv\_int\_min**Time =  $N \times 0.625$  ms Range: 0x0020 to 0x4000**uint16\_t adv\_int\_max**Time =  $N \times 0.625$  ms Range: 0x0020 to 0x4000*ble\_hci\_adv\_type\_t* **adv\_type**

Advertising Type

*ble\_hci\_addr\_type\_t* **own\_addr\_type**

Own Address Type

*ble\_hci\_addr\_t* **peer\_addr**

Peer device bluetooth device address

*ble\_hci\_addr\_type\_t* **peer\_addr\_type**

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

*ble\_hci\_adv\_channel\_t* **channel\_map**

Advertising channel map

*ble\_hci\_adv\_filter\_t* **adv\_filter\_policy**

Advertising Filter Policy:

struct **ble\_hci\_scan\_param\_t**

Ble sscan parameters.

**Public Members***ble\_hci\_scan\_type\_t* **scan\_type**

Scan Type

**uint16\_t scan\_interval**Time =  $N \times 0.625$  ms Range: 0x0004 to 0x4000**uint16\_t scan\_window**Time =  $N \times 0.625$  ms Range: 0x0004 to 0x4000*ble\_hci\_addr\_type\_t* **own\_addr\_type**

Own Address Type

*ble\_hci\_adv\_filter\_t* **filter\_policy**

Scanning Filter Policy

## Macros

### **ESP\_BLE\_ADV\_DATA\_LEN\_MAX**

Advertising data maximum length.

### **ESP\_BLE\_SCAN\_RSP\_DATA\_LEN\_MAX**

Scan response data maximum length.

### **BLE\_HCI\_ADDR\_LEN**

BLE Address Length.

## Type Definitions

```
typedef uint8_t ble_hci_addr_t[BLE_HCI_ADDR_LEN]
```

Bluetooth device address.

```
typedef void (*ble_hci_scan_cb_t)(ble_hci_scan_result_t *scan_result, uint16_t result_len)
```

BLE HCI scan callback function type.

**Param scan\_result** : ble advertisement scan result

**Param result\_len** : length of scan result

## Enumerations

```
enum ble_hci_search_evt_t
```

Sub Event of BLE\_HCI\_BLE\_SCAN\_RESULT\_EVT.

*Values:*

enumerator **BLE\_HCI\_SEARCH\_INQ\_RES\_EVT**

Inquiry result for a peer device.

enumerator **BLE\_HCI\_SEARCH\_INQ\_CMPL\_EVT**

Inquiry complete.

enumerator **BLE\_HCI\_SEARCH\_DISC\_RES\_EVT**

Discovery result for a peer device.

enumerator **BLE\_HCI\_SEARCH\_DISC\_BLE\_RES\_EVT**

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

enumerator **BLE\_HCI\_SEARCH\_DISC\_CMPL\_EVT**

Discovery complete.

enumerator **BLE\_HCI\_SEARCH\_DI\_DISC\_CMPL\_EVT**

Discovery complete.

enumerator **BLE\_HCI\_SEARCH\_SEARCH\_CANCEL\_CMPL\_EVT**

Search cancelled

enumerator **BLE\_HCI\_SEARCH\_INQ\_DISCARD\_NUM\_EVT**

The number of pkt discarded by flow control

enum **ble\_hci\_addr\_type\_t**

BLE address type.

*Values:*

enumerator **BLE\_ADDR\_TYPE\_PUBLIC**

Public Device Address

enumerator **BLE\_ADDR\_TYPE\_RANDOM**

Random Device Address.

enumerator **BLE\_ADDR\_TYPE\_RPA\_PUBLIC**

Resolvable Private Address (RPA) with public identity address

enumerator **BLE\_ADDR\_TYPE\_RPA\_RANDOM**

Resolvable Private Address (RPA) with random identity address.

enum **ble\_hci\_dev\_type\_t**

Bluetooth device type.

*Values:*

enumerator **BLE\_HCI\_DEVICE\_TYPE\_BREDR**

enumerator **BLE\_HCI\_DEVICE\_TYPE\_BLE**

enumerator **BLE\_HCI\_DEVICE\_TYPE\_DUMO**

enum **ble\_hci\_adv\_type\_t**

BLE advertising type.

*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 **ble\_hci\_adv\_channel\_t**

Advertising channel mask.

*Values:*



enumerator **ADV\_CHNL\_37**

enumerator **ADV\_CHNL\_38**

enumerator **ADV\_CHNL\_39**

enumerator **ADV\_CHNL\_ALL**

enum **ble\_hci\_scan\_type\_t**

Ble scan type.

*Values:*

enumerator **BLE\_SCAN\_TYPE\_PASSIVE**

Passive scan

enumerator **BLE\_SCAN\_TYPE\_ACTIVE**

Active scan

enum **ble\_hci\_adv\_filter\_t**

Ble adv filter type.

*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.

## Chapter 4

# 显示

### 4.1 LCD 显示屏

#### 4.1.1 LCD 概述

通常所说的 LCD 是 TFT-LCD (薄膜晶体管液晶显示器) 的统称。它是一种常用的数字显示技术，常用于显示图像和文字。LCD 使用液晶材料和偏振光技术，当液晶分子受电场影响时，会改变光的偏振方向，从而控制光线强度，使图像或文字显示在屏幕上。

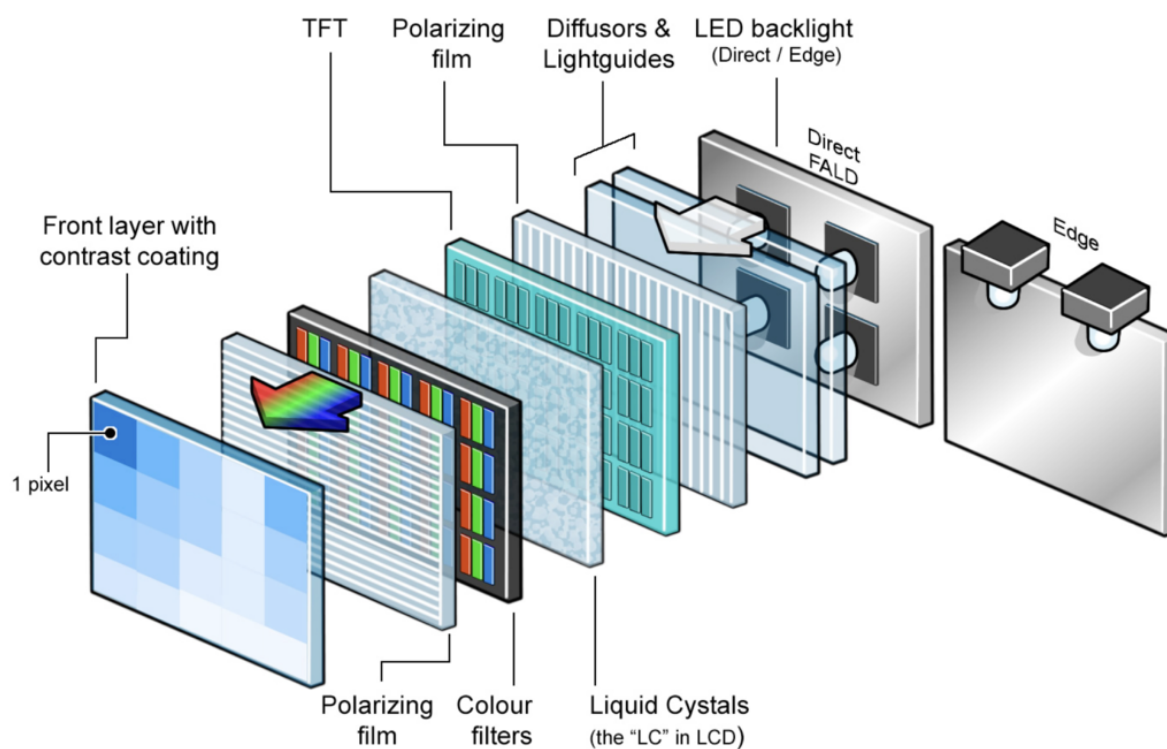


图 1: TFT-LCD 的硬件框图

LCD 具有很多优点，例如：能耗低、寿命长、可靠性高、清晰度高、占用空间小、颜色还原度高、抗眩光能力强等。因此，它已广泛应用于各种电子设备，如家电、便携式设备、可穿戴设备等。同时，LCD 技术持续进步和完善，其中包括不同的面板类型如 IPS、VA、TN，以及新的 LED 背光技术，这些都进一步提高了 LCD 的性能和用户体验。

本指南包含如下内容：

- **结构**：LCD 模块的主要结构，主要包含面板、背光源、驱动 IC 和 FPC。
- **形态**：LCD 模块的常见形态，主要有矩形屏和圆形屏。
- **驱动接口**：LCD 模块的驱动接口，包含 SPI、QSPI、I80、RGB 和 MIPI-DSI。
- **典型连接方式**：LCD 模块的典型连接方式，包含 LCD 的通用引脚以及不同类型的接口引脚。
- **帧率**：LCD 应用的帧率，包含渲染帧率、接口帧率和屏幕刷新率。

## 术语表

请参阅[LCD 术语表](#)。

## 结构

为了让 LCD 能够稳定工作并且方便开发，厂商通常将 LCD 封装成一个 **集成的模块**供用户使用，它主要由以下四个部分组成：

- **面板**：面板决定了 LCD 模块的色彩、可视角度、分辨率。面板的价格走势直接影响到模块的价格，面板的质量、技术的好坏关系到模块整体性能的高低。常见的面板类型有 IPS、VA、TN 等。
- **背光源**：液晶分子自身无法发光，因此若想出现画面，液晶屏需要专门的发光源来提供光线，然后经过液晶分子的偏转来产生不同的颜色。背光源起到的是提供光能的作用，一般可以通过 PWM 控制它的亮度。
- **驱动 IC**：驱动 IC 通过特定的接口对外通信，并控制输出电压让液晶扭转，使其发生色阶及明暗的变化。它通常包含控制电路和驱动电路两部分。控制电路负责接收来自主控芯片的信号，以及图像信号的转换与处理，驱动电路负责输出图像信号并显示到面板上。
- **FPC**：FPC 是 LCD 模块的对外接口，用于连接驱动 IC、外部驱动电路与主控芯片。FPC 由于其出色的柔性和可靠性，可以解决传统刚性线路板的接触问题和较差的抗振动性，从而增强了模块的稳定性和使用寿命。

通常，LCD 模块的选型主要基于其 **面板**和 **驱动 IC**。例如，我们会考虑面板的类型和分辨率以及驱动 IC 支持的接口类型和色彩格式。驱动 IC 体积很小，通常被贴在 FPC 与面板的连接部位，如图示。

## 形态

对于 LCD 的面板外形，大多数形状都是采用矩形或圆形，生活中最常见到的就是矩形屏，圆形屏多为小尺寸屏幕。

它们的特点及应用场景如下：

类型	特点	应用场景
矩形屏	面积大、效果好、信息呈现更多，应用范围更广	手机、平板电脑、控制面板
圆形屏	时尚轻便、占用空间小，有效利用设备面积	智能穿戴、电动仪器仪表、汽车显示仪表、智能家电、智能手持设备

通常使用 LCD 面板的对角线长度来衡量其 **尺寸**的大小，单位是英寸或寸，比如常说的 1.28 寸屏和 3.5 寸屏。除了屏幕的物理尺寸，开发者往往更加关心屏幕的 **分辨率**，它是指面板所能显示的像素点数量，代表了屏幕图像的精密度：可显示的像素越多，画面就越精细，同样的屏幕区域内能显示的信息越多，对主控芯片的性能要求也越高，所以分辨率是个非常重要的参数指标。

尺寸与分辨率之间不是一一对应的关系，但是总体呈正比的趋势，比如，一般情况下，2.4 寸或者 2.8 寸的屏幕常见分辨率为 320x240，3.2 寸或 3.5 寸的屏幕常见分辨率为 320x480。尺寸大的屏幕，其分辨率不

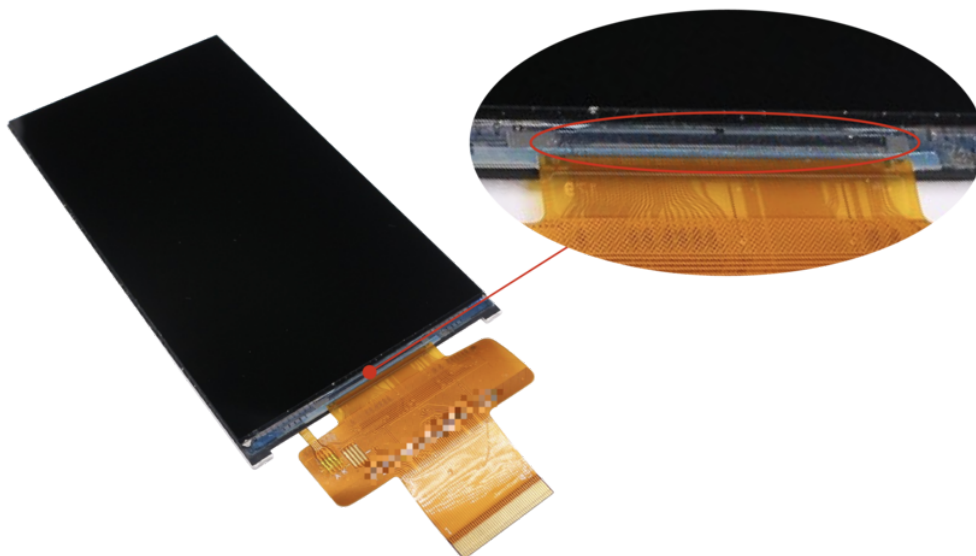


图 2: LCD 模块的驱动 IC



图 3: LCD 矩形屏



图 4: LCD 圆形屏

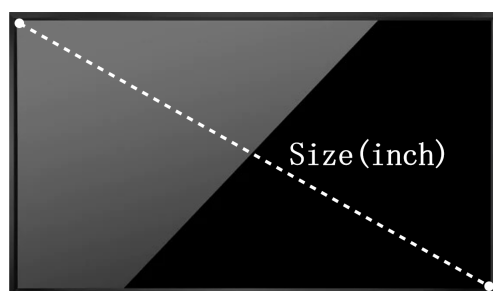


图 5: 屏幕尺寸

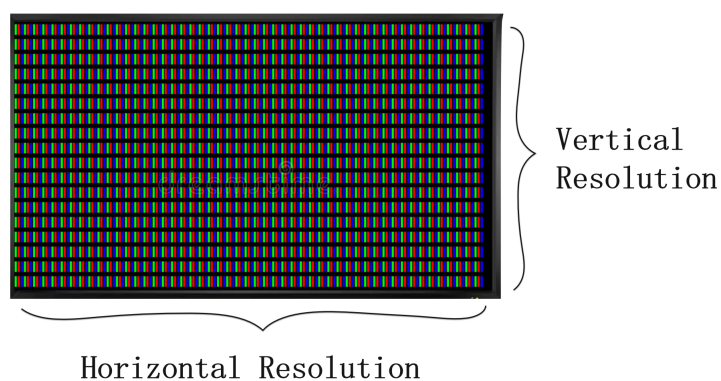


图 6: 屏幕分辨率

一定会比更小尺寸的屏幕更高，因此，在进行屏幕选型前，需要根据应用场景和需求确定好屏幕的尺寸与分辨率。

### 驱动接口

对于开发者而言，通常更加关心 LCD 的驱动接口，目前在物联网领域比较常见的接口类型有 SPI、QSPI、I80、RGB 和 MIPI-DSI，它们在占用 IO 数量、并行数据位数、数据传输带宽、GRAM 位置等方面的参数对比如下：

## 参数对比

类型	描述	占用 IO 数量	并行数据位数	数据传输带宽	GRAM 位置
SPI	串行接口，以 SPI 总线协议为基础，通常采用 4 线或 3 线模式	最少	1	最小	LCD
QSPI (Quad-SPI)	SPI 接口的一种扩展，可以使用 4 根数据线并行传输	较少	4	较小	LCD 或 主控
I80 (MCU、DBI)	并行接口，以 I80 总线协议为基础	较多	8/16	较大	LCD
RGB (DPI)	并行接口，一般需搭配 3-wire SPI 接口	最多	8/16/18/24	最大	主控
MIPI-DSI	采用差分信号传输方式的串行接口，基于 MIPI 的高速、低功率可扩展串行互联的 D-PHY 物理层规范	较多	1/2/3/4	最大	LCD 或 主控

## 备注：

- 对于 QSPI 接口，不同型号的驱动 IC 可能采用不同的驱动方式，如 *SPD2010* 内置 GRAM，其驱动方式与 SPI/I80 接口类似，而 *ST77903* 没有内置 GRAM，其驱动方式与 RGB 接口类似。
- 对于 MIPI-DSI 接口，采用 Command 模式需要 LCD 内置 GRAM，而 Video 模式则不需要。

## 总结如下：

1. SPI 接口的数据传输带宽小，比较适用于低分辨率的屏幕。
2. QSPI 和 I80 接口的数据传输带宽更大，所以能够支持较高分辨率的屏幕，但是 I80 接口要求 LCD 内置 GRAM，导致屏幕成本较高，并且难以做到大屏。
3. RGB 与 I80 接口类似，但是 RGB 接口无需 LCD 内置 GRAM，因此适用于更高分辨率的屏幕。
4. MIPI-DSI 接口适用于高分辨率、高刷新率的屏幕。

**接口详解** 驱动 LCD 的第一步是确定它的接口类型，对于大部分常见的驱动 IC，如 *ST7789*、*GC9A01*、*ILI9341* 等，它们一般都会支持多种接口，但是屏幕厂商在封装成模块的时候通常只对外留出其中一种接口（RGB LCD 通常会也会使用 SPI 接口）。以 *GC9A01* 为例，它的硬件框图如下：

很多 LCD 驱动 IC 的实际接口类型是由其 IM[3:0] 引脚的高低电平来决定的，大部分屏幕在内部已经固定了这些引脚的配置，但是也有一些屏幕会预留出这些引脚以及所有的接口引脚，这种情况下用户可以自行配置。以 *ST7789* 为例，它的接口类型配置如下：

因此，仅仅知道驱动 IC 的型号并不能确定屏幕的接口类型，在这种情况下可以咨询屏幕厂商，或者查阅屏幕的数据手册，也可以通过原理图结合经验进行判断，下面是各种接口的屏幕引脚对比：

类型	引脚
LCD 通用	RST (RESET)、Backlight (LEDA、LEDK)、TE (tear effect)、Power (VCC、GND)
SPI	CS、SCK(SCL)、SDA (MOSI)、SDO (MISO)、DC (RS)
QSPI	CS、SCK(SCL)、SDA (DATA0)、DATA1、DATA2、DATA3
I80	CS (CSX)、RD (RDX)、WR (WRX)、DC (D/CX)、D[15:0] (D[7:0])
RGB	CS、SCK(SCL)、SDA(MOSI)、HSYNC、VSYNC、PCLK、DE、D[23:0] (D[17:0]/D[7:0])

常用接口 LCD 的详细介绍如下：

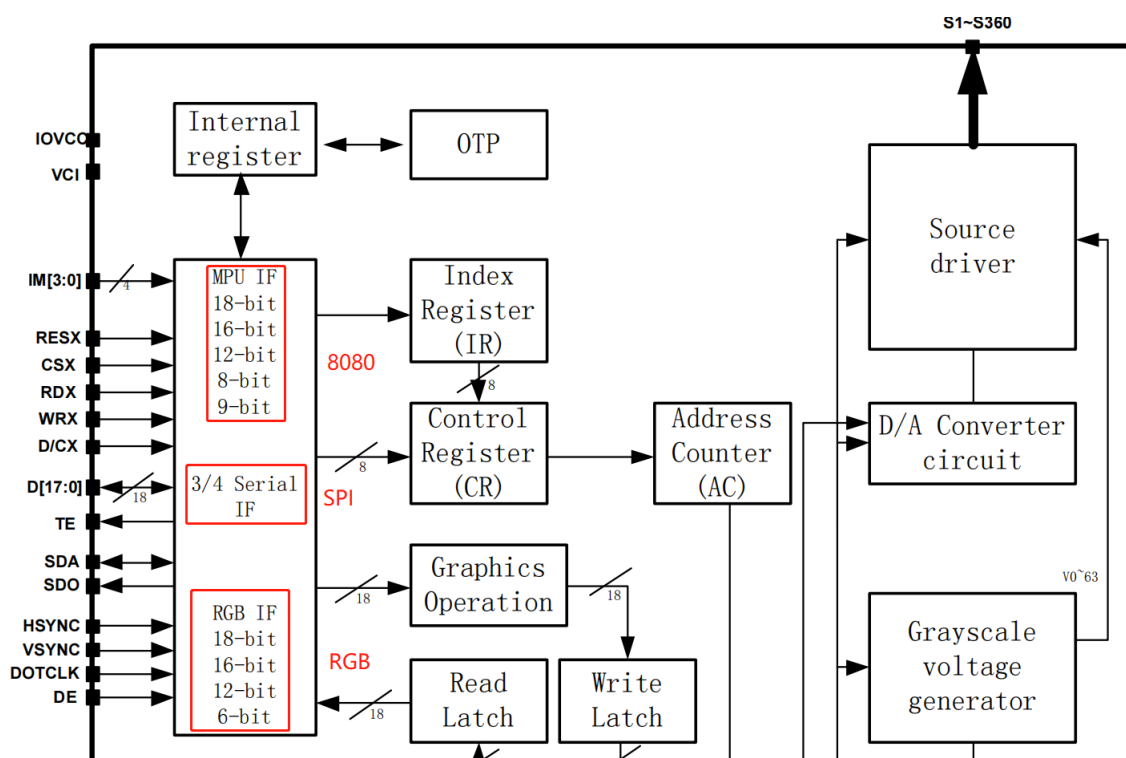


图 7: GC9A01 的硬件框图

ST7789V supports 8/16/9/18 bit parallel data bus for 8080 series CPU, RGB serial interfaces. Selection of these interfaces are set by IM[3:0] pins as shown below.

IM3	IM2	IM1	IM0	Interface	Read Back Data Bus Selection
0	0	0	0	80-8bit parallel I/F	DB[7:0]
0	0	0	1	80-16bit parallel I/F	DB[15:0]
0	0	1	0	80-9bit parallel I/F	DB[8:0]
0	0	1	1	80-18bit parallel I/F	DB[17:0],
0	1	0	1	3-line 9bit serial I/F	SDA: in/out
				2 data lane serial I/F	SDA: in/out, WRX: in
0	1	1	0	4-line 8bit serial I/F	SDA: in/out
1	0	0	0	80-16bit parallel I/F II	DB[17:10], DB[8:1]
1	0	0	1	80-8bit parallel I/F II	DB[17:10]
1	0	1	0	80-18bit parallel I/F II	DB[17:0],
1	0	1	1	80-9bit parallel I/F II	DB[17:9]
1	1	0	1	3-line 9bit serial I/F II	SDA: in/ SDO: out
1	1	1	0	4-line 8bit serial I/F II	SDA: in/ SDO: out

图 8: ST7789 的接口配置



- [SPI LCD 详解](#)
- [RGB LCD 详解](#)
- I80 LCD 详解（待更新）
- QSPI LCD 详解（待更新）

### 典型连接方式

对于通用的 LCD 引脚，通常采用如下的连接方式：

- **RST (RESET)**: 推荐连接至 GPIO，并根据 LCD 驱动 IC 的数据手册，在上电时输出复位时序。一般情况下也可以使用上拉/下拉电阻连接系统电源。
- **Backlight (LEDA、LEDK)**: 推荐 LEDA 连接至系统电源（阳极），LEDK 使用开关元器件连接至系统电源（阴极），并通过 GPIO 控制亮灭，或者通过 LEDC 外设输出 PWM 以调节背光亮度。
- **TE (tear effect)**: 推荐连接至 GPIO，通过 GPIO 中断来获取 TE 信号，以实现帧同步。
- **Power (VCC、GND)**: 推荐全部连接至对应的系统电源，而不要让一部分引脚浮空。

对于不同接口类型的引脚，主控 MCU 需要采用不同的连接方式，下面将分别介绍 SPI QSPI I80 和 RGB 四种接口的典型连接方式。

**SPI 接口** SPI 接口的 LCD 硬件设计请参考开发板 [ESP32-C3-LCDkit](#) 及其 [LCD 子板](#)，其典型连接示意图如下：

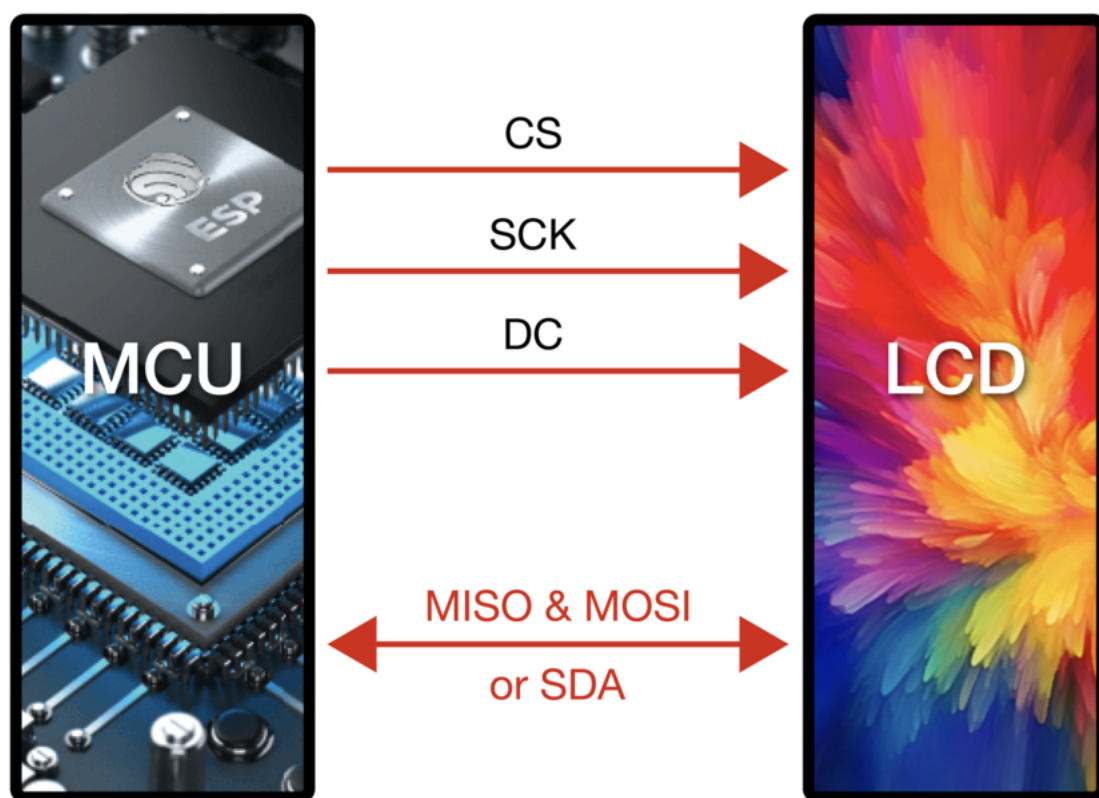


图 9: SPI 接口典型连接示意图

### 备注：

- Interface I 模式仅需使用 SDA 一根数据线，Interface II 模式需要使用 MISO & MOSI 两根数据线。



- 通常情况下不需要从 LCD 读取数据，因此可以不连接 MOSI。如果有需要的话请注意，大多数 SPI LCD 读取时的最大时钟频率要远小于写入时的频率。
- 由于 3-line 模式（无 D/C 信号线）下，每传输单位数据（通常为字节）都需要先传输 D/C 信号（1-bit），而目前 ESP 的 SPI 外设不支持直接传输 9-bit 数据，因此通常采用上图所示的 4-line 模式。

**QSPI 接口** QSPI 接口的典型连接示意图如下：

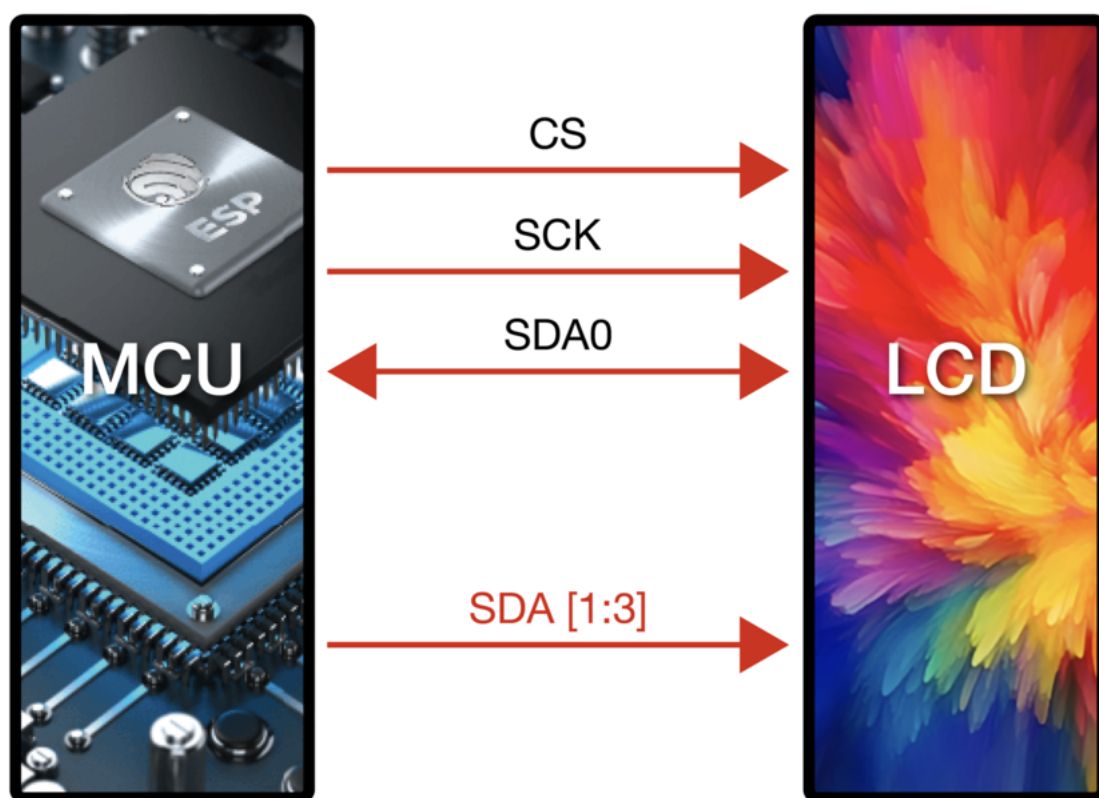


图 10: QSPI 接口典型连接示意图

**备注：**

- 不同型号驱动 IC 的 QSPI 接口连接方式可能不同，上图仅以 *ST77903* 为例。
- 写入数据时需要使用 SDA0 和 SDA[1:3] 四根数据线，读取数据时仅使用 SDA0 一根数据线。

**I80 接口** I80 接口的 LCD 硬件设计请参考开发板 [ESP32-S3-LCD-EV-Board](#) 及其 [LCD 子板 \(3.5' LCD\\_ZJY\)](#)，其典型连接示意图如下：

**备注：**

- 图中虚线表示可选引脚。
- ESP 的 I80 外设不支持使用 RD 信号进行读取操作，因此实际连接时需要将该信号拉高。

**RGB 接口** RGB 接口的 LCD 硬件设计请参考开发板 [ESP32-S3-LCD-EV-Board](#) 及其 [LCD 子板 \(3.95' LCD\\_QMZX\)](#)，其典型连接示意图如下：

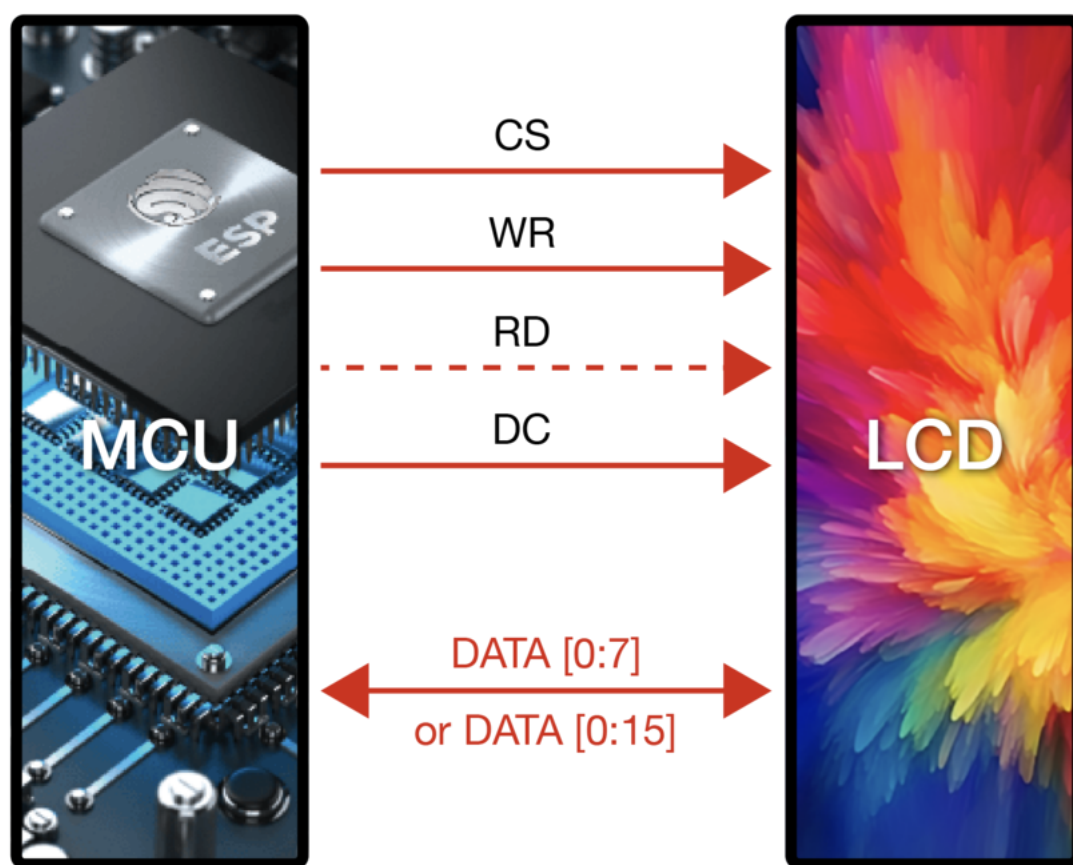


图 11: I80 接口典型连接示意图

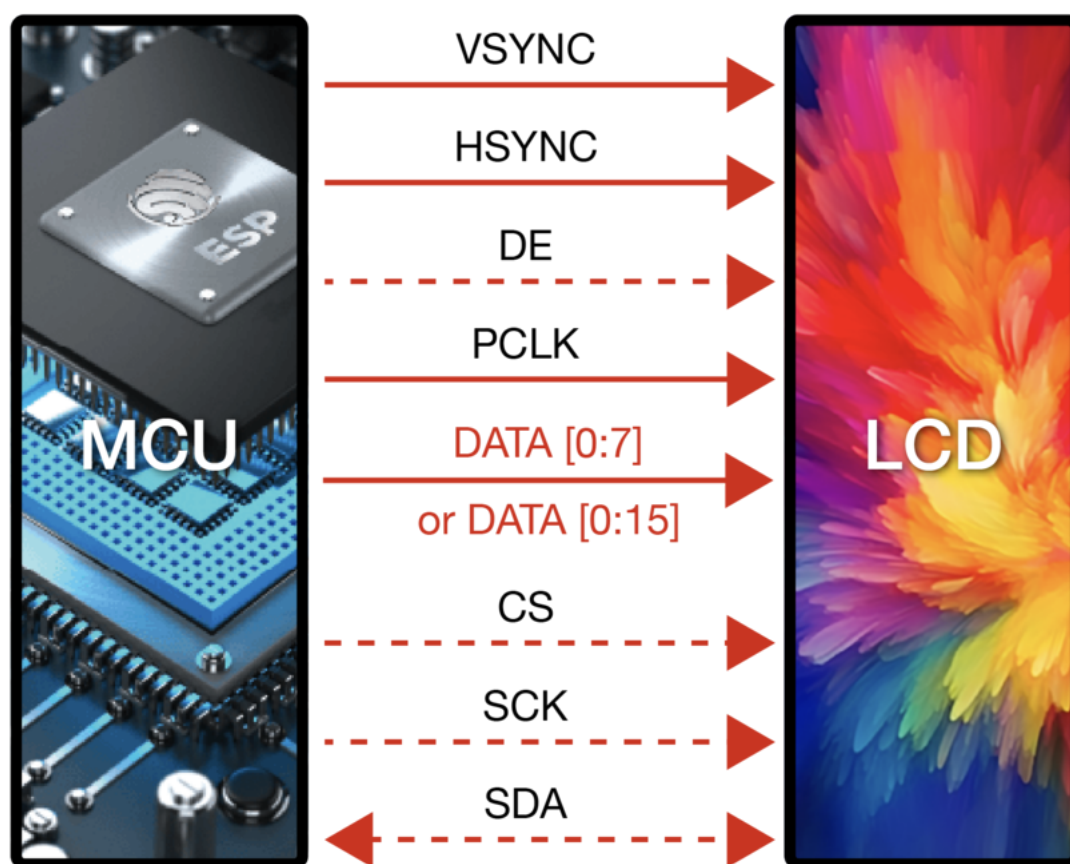


图 12: RGB 接口典型连接示意图

**备注:**

- 图中虚线表示可选引脚。
- DE 用于 DE 模式下。
- CS、SCK 和 SDA 为 3-wire (3-line) SPI 接口引脚，用于发送命令及参数对 LCD 进行配置，一些屏幕可能没有这些引脚，因此也不需要初始化配置。由于 3-wire SPI 接口可以仅用于进行 LCD 的初始化，而无需用于后续的屏幕刷新，因此，为了节省 IO 数量，可以将 SCK 和 SDA 与任意 RGB 接口引脚进行复用。

**帧率**

对于 LCD 应用来说，屏幕上的动画是通过显示多个连续的静止图像来实现的，这些图像被称为 **帧**。**帧率**就是显示新帧的速率，它通常表示为每秒变化的帧数，简称为 FPS。帧率越高，每秒显示的帧就越多，动画变化得也更平滑、更逼真。

但是一帧图像的显示并不是仅由主控一次性完成的，而是经过渲染、传输、显示等多个步骤，因此，帧率的高低不仅取决于主控的性能，还取决于 LCD 的接口类型和刷新率等因素。

**渲染** 渲染是指主控通过计算生成图像数据的过程，其快慢可以用 **渲染帧率**来衡量。

渲染帧率一方面取决于主控的性能，另一方面也受动画复杂程度的影响，比如，局部变化的动画通常比全屏变化的动画渲染帧率更高，纯色填充通常图层混叠的渲染帧率更高。因此，渲染帧率在图像变化时一般是不固定的，如 LVGL 运行时统计的 FPS。

参考 [LVGL 运行时统计的 FPS](#)。

**传输** 传输是指主控将渲染好的图像数据通过外设接口传输到 LCD 驱动 IC 的过程，其快慢可以用 **接口帧率**来衡量。

接口帧率取决于 LCD 的接口类型和主控的数据传输带宽，通常在外设接口初始化完成后就会固定，因此可以通过公式计算得出：

$$\text{帧率} = \frac{\text{传输数据量}}{\text{传输时间}}$$

对于 SPI/I80 接口：

$$\text{帧率} = \frac{\text{帧大小} \times \text{帧数}}{\text{帧大小} \times \text{帧数} \times \text{帧数}}$$

对于 RGB 接口：

$$\begin{aligned} \text{帧率} &= \frac{\text{帧大小} \times \text{帧数}}{\text{帧大小} \times \text{帧数} \times \text{帧数}} \\ \text{帧大小} &= \text{帧大小} + \text{帧大小} + \text{帧大小} + \text{帧大小} \\ \text{帧大小} &= \text{帧大小} + \text{帧大小} + \text{帧大小} + \text{帧大小} \end{aligned}$$

**显示** 显示是指 LCD 的驱动 IC 将接收到的图像数据显示到屏幕上的过程，其快慢可以用 **屏幕刷新率**来衡量。

对于 SPI/I80 接口的 LCD，屏幕刷新率是由 LCD 驱动 IC 决定的，一般可以通过发送特定的命令来设置，如 ST7789 的 FRCTRL2 (C6h) 命令；对于 RGB 接口的 LCD，屏幕刷新率是由主控决定的，其等价于接口帧率。

### 4.1.2 LCD 术语表

该部分旨在介绍 LCD 相关术语的含义。

术语	含义
<b>GRAM</b>	Graphic RAM 的缩写，用于保存图形数据的存储区域
<b>TE</b>	用于指示屏幕刷新垂直同步或水平同步的时序信号
<b>Porch</b>	在显示一行或一帧图像数据之前的空白时间间隔

### 4.1.3 LCD 开发指南

本指南主要包含如下内容：

- **支持的接口类型**：乐鑫各系列芯片对不同 LCD 接口的支持情况。
- **驱动及示例**：乐鑫提供的 LCD 驱动及示例。
- **开发框架**：开发 LCD 的软硬件框架。
- **开发步骤**：开发 LCD 应用的详细步骤。
- **常见问题**：列出了开发 LCD 应用过程中常见的问题。
- **相关文档**：列出了相关文档的链接。

#### 术语表

请参阅[LCD 术语表](#)。

#### 支持的接口类型

乐鑫芯片已经支持了[LCD 概述 - 驱动接口](#)一节介绍的全部接口类型，各系列 ESP 芯片的具体支持情况如下：

Soc	SPI (QSPI)	I80	RGB	MIPI-DSI
ESP32	Supported	Supported		
ESP32-C3	Supported			
ESP32-C6	Supported			
ESP32-S2	Supported	Supported		
ESP32-S3	Supported	Supported	Supported	
ESP32-P4	Supported	Supported	Supported	Supported

#### 驱动及示例

**LCD 外设驱动**位于 **ESP-IDF** 下的 [components/esp\\_lcd](#) 目录，目前支持 I2C、SPI (QSPI)、I80 以及 RGB 接口，详细介绍请参考 [文档](#)。下表是目前乐鑫官方基于 esp\_lcd 移植的 **LCD 驱动组件**，并且 **LCD 驱动组件**会持续更新：

接口	LCD 控制器
I2C	ssd1306, sh1107
SPI	axs15231b, st7789, nt35510, gc9b71, nv3022b, sh8601, spd2010, st77916, st77922, gc9a01, ili9341, ssd1681, st7796
QSPI	axs15231b, gc9b71, sh8601, spd2010, st77903, st77916, st77922
I80	axs15231b, st7789, nt35510, ra8875, st7796
MIPI-DSI	ek79007, jd9165, jd9365, st7701, ili9881c
3-wire SPI + RGB	st7701, st77903_rgb, st77922, gc9503

请注意：

- **st7789**、**nt35510**、**ssd1306** 组件保存在 [ESP-IDF](#) 中。其余组件可以在 [ESP 组件管理器](#) 中搜索使用。
- 即使 **LCD 驱动 IC** 的型号相同，不同的屏幕往往需要使用各自厂商提供的初始化命令配置，大部分驱动组件支持在初始化 LCD 设备时传入自定义的初始化命令，若不支持，请参考 [方法](#)。

**LCD 示例**位于 **ESP-IDF** 下的 `examples/peripherals/lcd` 目录和 **esp-iot-solution** 下的 `examples/display/lcd` 目录，可参考 LCD 驱动组件使用。

备注：

- 推荐基于 [ESP-IDF release/v5.1](#) 及以上版本分支进行开发，因为低版本不支持部分重要的新特性，尤其是对于 RGB 接口。
- 对于使用 3-wire SPI + RGB 接口的 LCD，请参考示例 [esp\\_lcd\\_st7701 - Example use](#)。

## 开发框架

**硬件框架** 对于 SPI/I80 LCD，ESP 可以通过单一的外设接口发送 **命令**来配置 LCD 以及传输 **局部的色彩数据**来刷新屏幕。LCD 的驱动 IC 会将接收到的色彩数据存储在 **全屏大小的 GRAM** 内，并按照固定的刷新速率把 **全屏的色彩数据**显示到面板上，这两个过程是异步进行的。下面是 SPI/I80 LCD 的硬件驱动框架示意图：

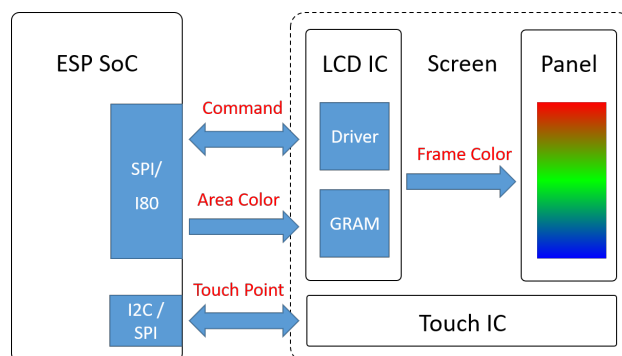


图 13: 硬件驱动框架示意图 - SPI/I80 LCD

对于大多数 RGB LCD，ESP 需要使用两种不同的接口，一方面通过 3-wire SPI 接口发送 **命令**来配置 LCD，另一方面通过 RGB 接口传输 **全屏的色彩数据**来刷新屏幕。由于 LCD 的驱动 IC 没有内置的 GRAM，它会将接收到的色彩数据直接显示到面板上，因此这两个过程是同步进行的。下面是 RGB LCD 的硬件驱动框架示意图：

通过对比这两种框架可以看出，RGB LCD 相较于 SPI/I80 LCD，不仅需要 ESP 使用两种接口来分别实现传输命令和色彩数据，还要求 ESP 提供全屏大小的 GRAM 来实现屏幕刷新（由于芯片内的 SRAM 的空间比较有限，通常将 GRAM 放在 PSRAM 上）。

对于 QSPI LCD，不同型号的驱动 IC 可能需要不同的驱动方式，比如 *SPD2010* 这款 IC 内置 GRAM，其驱动方式与 SPI/I80 LCD 类似，而 *ST77903* 这款 IC 内部没有 GRAM，其驱动方式与 RGB LCD 类似，但是它们都是通过用单一的外设接口传输命令和色彩数据，下面是这两种 QSPI LCD 的硬件驱动框架示意图：



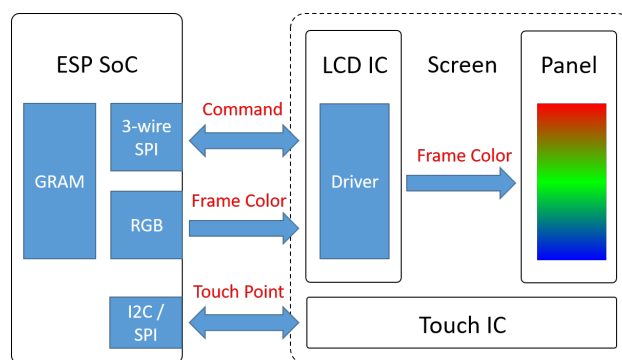


图 14: 硬件驱动框架示意图 - RGB LCD

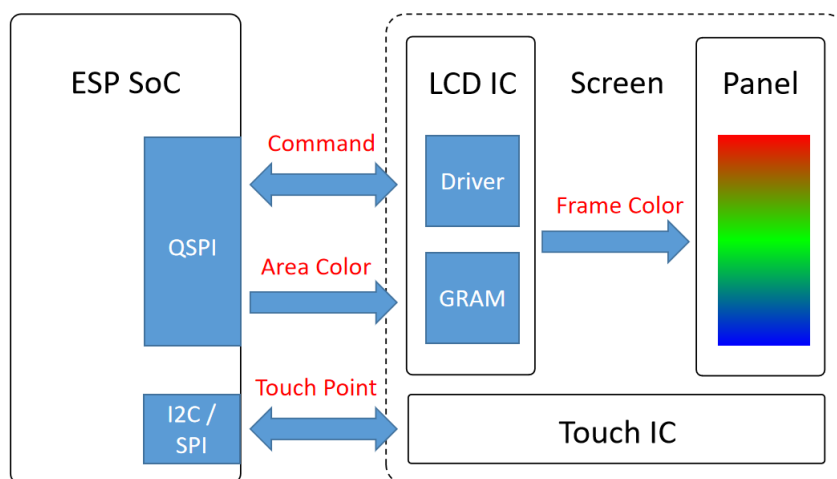


图 15: 硬件驱动框架示意图 - QSPI LCD (有 GRAM)

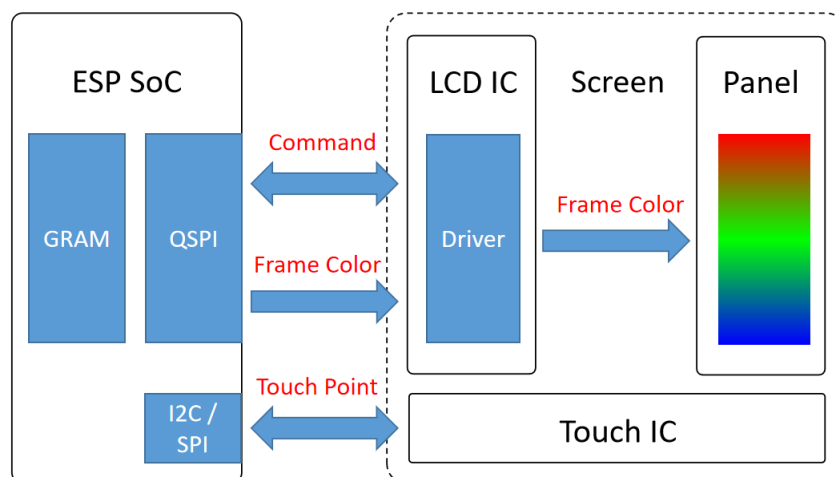


图 16: 硬件驱动框架示意图 - QSPI LCD (无 GRAM)

**软件框架** 软件开发框架主要由 SDK、Driver 和 APP 三个层次组成:

1. **SDK 层**: ESP-IDF 作为框架的基础部分, 不仅包含了驱动 LCD 所需的 I2C、SPI (QSPI)、I80 和 RGB 等多种外设, 还通过 `esp_lcd` 组件提供了统一的 APIs 来操作接口和 LCD, 如命令及参数的传输, LCD 的图像刷新、反转、镜像等功能。
2. **Driver 层**: 基于 SDK 提供的 APIs 可以实现各种设备驱动, 并通过初始化接口设备和 LCD 设备实现 LVGL (GUI 框架) 的移植。
3. **APP 层**: 使用 LVGL 提供的 APIs 实现各种 GUI 功能, 如显示图片、动画、文字等。

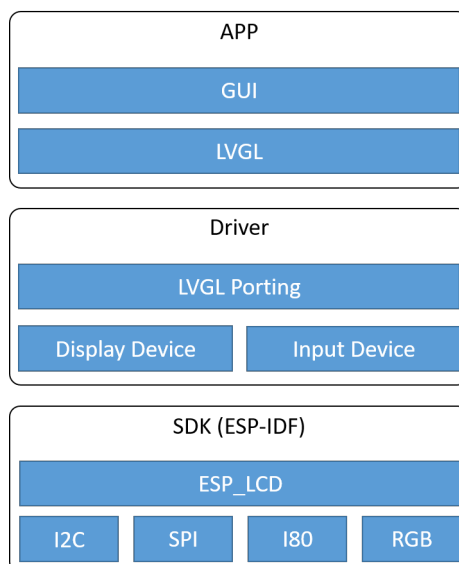


图 17: 软件开发框架示意图

## 开发步骤

**初始化接口设备** 首先, 初始化与 LCD 接口对应的外设。然后, 创建接口设备并获取其句柄, 该句柄的数据类型应为 `esp_lcd_panel_io_handle_t`。这样即可使用统一的 [接口通用 APIs](#) 进行数据传输。

---

**备注:** 对于仅采用 RGB 接口的 LCD, 不需要创建其接口设备, 请直接参考[LCD 初始化](#)。

---

不同类型的 LCD 接口需要使用不同的外设, 下面对几种常用接口的设备初始化过程进行说明:

- [SPI LCD 详解 - 初始化接口设备](#)
- [RGB LCD 详解 - 初始化接口设备](#)
- [I80 LCD 详解 - 初始化接口设备 \(待更新\)](#)
- [QSPI LCD 详解 - 初始化接口设备 \(待更新\)](#)

关于这部分更加详细的说明, 请参考 [ESP-IDF 编程指南](#)。

**初始化 LCD 设备** 由于不同型号的 LCD 驱动 IC 可能具有不同的命令 (寄存器) 和参数, 并且不同的接口类型也可能采用不同的数据格式和驱动方式, 首先需要针对特定的接口利用 [接口通用 APIs](#) 来移植目标 LCD 驱动, 然后创建 LCD 设备并获取数据类型为 `esp_lcd_panel_handle_t` 的句柄, 最终使得应用程序能够通过统一的 [LCD 通用 APIs](#) 来操作 LCD 设备。

---

**备注:** 对于仅采用 RGB 接口的 LCD, 不需要移植其驱动组件, 请直接参考[LCD 初始化](#)。

---

在移植驱动组件前, 请先尝试直接从[LCD 驱动组件](#)中获取目标 LCD 驱动 IC 的组件。若该组件不存在, 那么也可以基于已有的并且接口类型相同的组件进行移植。不同接口类型的 LCD 驱动可能具有不同的移植原理, 下面对几种常用接口的 LCD 驱动组件的移植方法进行说明:



- [SPI LCD 详解 - 移植驱动组件](#)
- [RGB LCD 详解 - 移植驱动组件](#)
- [I80 LCD 详解 - 移植驱动组件 \(待更新\)](#)
- [QSPI LCD 详解 - 移植驱动组件 \(待更新\)](#)

然后，利用驱动组件就可以实现 LCD 的初始化，下面对几种常用接口的 LCD 初始化进行说明：

- [SPI LCD 详解 - 初始化 LCD 设备](#)
- [RGB LCD 详解 - 初始化 LCD 设备](#)
- [I80 LCD 详解 - 初始化 LCD 设备 \(待更新\)](#)
- [QSPI LCD 详解 - 初始化 LCD 设备 \(待更新\)](#)

关于这部分更加详细的说明，请参考 [ESP-IDF 编程指南](#)。

**移植 LVGL** (待更新)

**设计 GUI** (待更新)

### 常见问题

下面列举了一些开发 LCD 应用过程中常见的问题，请点击问题跳转查看解决方法。

- [ESP 系列芯片如何使用 Arduino IDE 开发 GUI](#)
- [ESP 系列芯片支持 LCD 的最大分辨率及帧率](#)
- [ESP 系列芯片如何提高 LCD 的渲染帧率](#)
- [ESP32-S3 如何提高 RGB LCD 的 PCLK \(刷新帧率\)](#)
- [ESP32-S3 如何解决驱动 RGB LCD 出现屏幕偏移或闪烁的问题](#)
- [ESP32-S3R8 如何配置 PSRAM 120M Octal\(DDR\)](#)

### 相关文档

- [ESP-IDF 编程指南 - LCD](#)
- [ESP-FAQ - LCD](#)
- [LVGL 文档](#)

## 4.1.4 SPI LCD 详解

### 目录

- [术语表](#)
- [接口模式](#)
  - [Interface I/II 模式](#)
  - [3/4-line 模式](#)
- [SPI LCD 驱动流程](#)
- [初始化接口设备](#)
  - [初始化总线](#)
  - [创建接口设备](#)
- [移植驱动组件](#)
- [初始化 LCD 设备](#)
- [相关文档](#)

## 术语表

请参阅[LCD 术语表](#)。

## 接口模式

不同的接口模式需要主控采用不同的 **接线**和 **驱动**方式，下面以 *ST7789* 为例，介绍几种比较常见的接口模式。

IM3	IM2	IM1	IM0	Interface	Read back selection
0	1	0	1	3-line serial interface I	Via the read instruction (8-bit, 24-bit and 32-bit read parameter)
0	1	1	0	4-line serial interface I	
1	1	0	1	3-line serial interface II	
1	1	1	0	4-line serial interface II	

Table 13 Selection of serial interface

图 18: SPI 接口的模式选择

从上图中可以看出，*ST7789* 是通过 IM[3:0] 引脚来选择 Interface I/II 和 3/4-line 的配置，可以实现 4 种不同的接口模式。下图为 *ST7789* 的 SPI 接口的引脚描述：

注：SPI 引脚名称：CS、SCK(SCL)、SDA (MOSI)、SDO (MISO)、DC (RS)

**Interface I/II 模式** 从图中可以看出，Interface I 和 Interface II 的主要区别在于是否仅用一根数据线实现数据的读取和写入（如仅用 MOSI）。

模式	是否仅用一根数据线实现数据的读取和写入	ESP 是否支持
Interface I	是	是
Interface II	否	是

**3/4-line 模式** 从图中可以看出，3-line 和 4-line 的主要区别在于是否使用 D/C 信号线。

模式	是否使用 D/C 信号线	ESP 是否支持
3-line	否	否
4-line	是	是

### 备注：

- 3-line 模式有时也称为 3-wire 或 9-bit 模式。
- 虽然 ESP 的 SPI 外设不支持 LCD 的 3-line 模式，但是可以通过软件模拟实现，具体请参考组件 [esp\\_lcd\\_panel\\_io\\_additions](#)，它通常用于实现 RGB LCD 的初始化。

## SPI LCD 驱动流程

SPI LCD 驱动流程可大致分为三个部分：初始化接口设备、移植驱动组件和初始化 LCD 设备。

3-line serial interface I

Pin Name	Description
CSX	Chip selection signal
DCX	Clock signal
SDA	Serial input/output data

4-line serial interface I

Pin Name	Description
CSX	Chip selection signal
WRX	Data is regarded as a command when WRX is low Data is regarded as a parameter or data when WRX is high
DCX	Clock signal
SDA	Serial input/output data

3-line serial interface II

Pin Name	Description
CSX	Chip selection signal
DCX	Clock signal
SDA	Serial input data
SDO	Serial output data

4-line serial interface II

Pin Name	Description
CSX	Chip selection signal
WRX	Data is regarded as a command when WRX is low Data is regarded as a parameter or data when WRX is high

图 19: SPI 接口的引脚描述

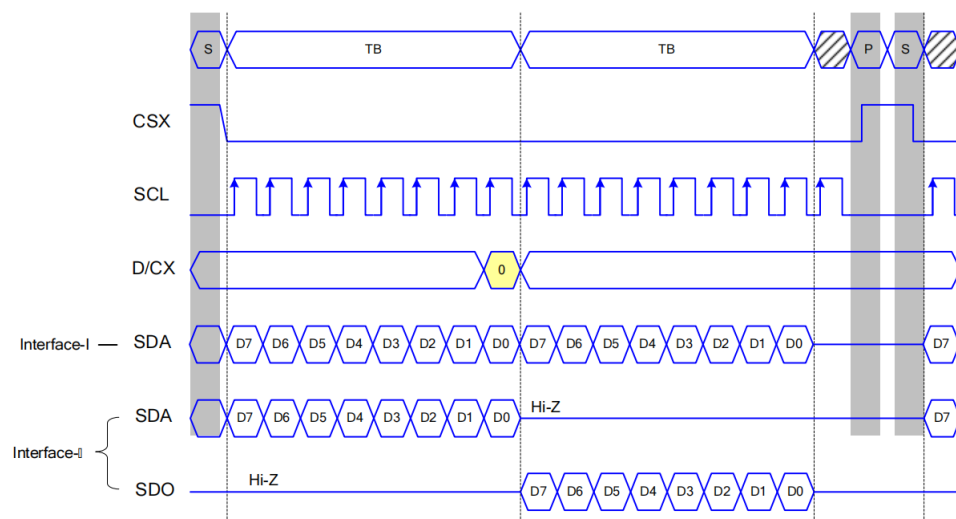


图 20: Interface I/II 模式的时序图对比 (4-line)

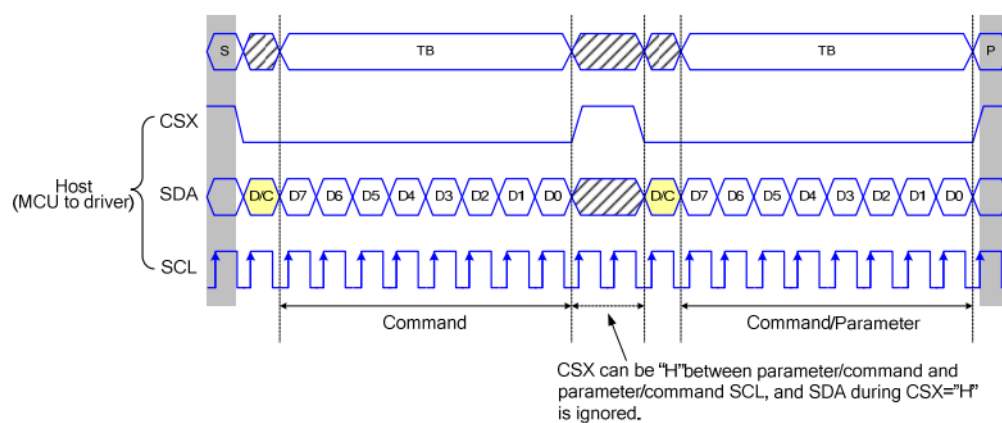


Figure 13 3-line serial interface write protocol (write to register with control bit in transmission)

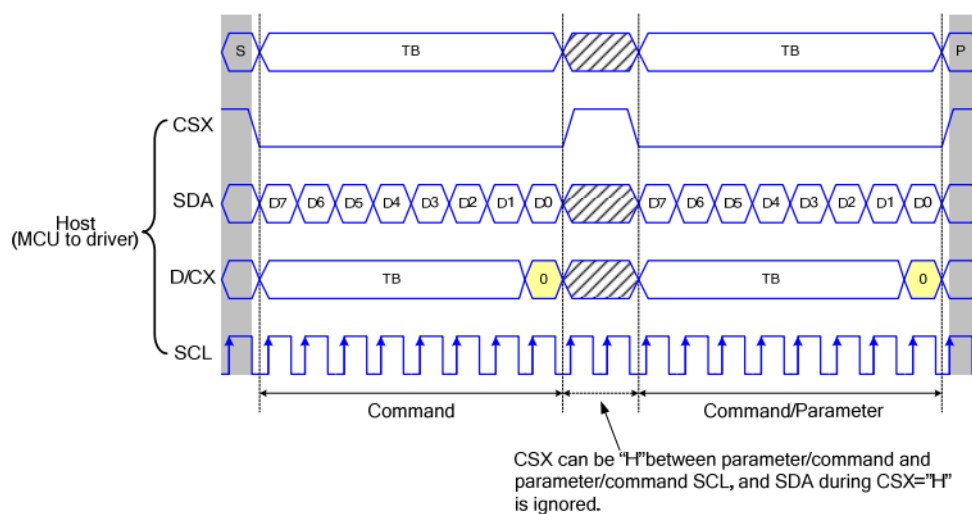


Figure 14 4-line serial interface write protocol (write to register with control bit in transmission)

图 21: 3/4-line 模式的时序图对比 (Interface I)

## 初始化接口设备

初始化接口设备需要先初始化总线，再创建接口设备。下面基于 ESP-IDF release/v5.1 中的 `spi_lcd_touch` 示例，具体介绍如何初始化 SPI 接口设备。

### 初始化总线 示例代码:

```
#include "driver/spi_master.h"           // 依赖的头文件
#include "esp_check.h"

spi_bus_config_t buscfg = {
    .sclk_io_num = EXAMPLE_PIN_NUM_SCLK, // 连接 LCD SCK (SCL) 信号的 IO 编号
    .mosi_io_num = EXAMPLE_PIN_NUM_MOSI, // 连接 LCD MOSI (SDO、SDA) 信号的 IO_
    ↪ 编号
    .miso_io_num = EXAMPLE_PIN_NUM_MISO, // 连接 LCD MISO (SDI) 信号的 IO_
    ↪ 编号, 如果不需要从 LCD 读取数据, 可以设为 `-1`
    .quadwp_io_num = -1,                 // 必须设置且为 `-1`
    .quadhd_io_num = -1,                 // 必须设置且为 `-1`
    .max_transfer_sz = EXAMPLE_LCD_H_RES * 80 * sizeof(uint16_t), // 表示 SPI_
    ↪ 单次传输允许的最大字节数上限, 通常设为全屏大小即可
};
ESP_ERROR_CHECK(spi_bus_initialize(LCD_HOST, &buscfg, SPI_DMA_CH_AUTO));
// 第 1 个参数表示使用的 SPI 主机_
    ↪ ID, 和后续创建接口设备时保持一致
// 第 3 个参数表示使用的 DMA_
    ↪ 通道号, 默认设置为 `SPI_DMA_CH_AUTO` 即可
```

如果有多个设备同时使用同一 SPI 总线，那么只需要对总线初始化一次。

下面是部分配置参数的说明：

- 若 LCD 驱动 IC 配置为 *Interface-1 接口模式*，软件仅需设置 `mosi_io_num` 为其数据线 IO，而设置 `miso_io_num` 为 -1。
- SPI 驱动** 在传输数据前会对输入数据量的大小进行判断，若单次传输的字节数超过 `max_transfer_sz` 则会报错。但是，**SPI 单次 DMA 传输允许的最大字节数** 不仅取决于 `max_transfer_sz`，而且受限于 ESP-IDF 中的 `SPI_LL_DATA_MAX_BIT_LEN`（不同系列 ESP 的值不同），即满足 `最大字节数 ≤ MIN(max_transfer_sz, (SPI_LL_DATA_MAX_BIT_LEN / 8))`。由于 `esp_lcd 驱动` 会提前判断输入的数据量是否超过限制，如果超过则进行 **分包处理**后才控制 SPI 进行多次传输，因此 `max_transfer_sz` 通常设为全屏大小即可。

### 创建接口设备 示例代码:

```
#include "esp_lcd_panel_io.h"           // 依赖的头文件

static bool example_on_color_trans_dome(esp_lcd_panel_io_handle_t panel_io, esp_
    ↪ lcd_panel_io_event_data_t *edata, void *user_ctx)
{
    /* 色彩数据传输完成时的回调函数，可以在此处进行一些操作 */

    return false;
}

esp_lcd_panel_io_handle_t io_handle = NULL;
esp_lcd_panel_io_spi_config_t io_config = {
    .dc_gpio_num = EXAMPLE_PIN_NUM_LCD_DC, // 连接 LCD DC (RS) 信号的 IO_
    ↪ 编号, 可以设为 `-1` 表示不使用
    .cs_gpio_num = EXAMPLE_PIN_NUM_LCD_CS, // 连接 LCD CS 信号的 IO_
    ↪ 编号, 可以设为 `-1` 表示不使用
    .pclk_hz = EXAMPLE_LCD_PIXEL_CLOCK_HZ, // SPI 的时钟频率 (Hz), ESP_
    ↪ 最高支持 80M (SPI_MASTER_FREQ_80M)
```

(下页继续)

(续上页)

```

// 需根据 LCD 驱动 IC
↪ 的数据手册确定其最大值
    .lcd_cmd_bits = EXAMPLE_LCD_CMD_BITS, // 单位 LCD 命令的比特数, 应为 8
↪ 的整数倍
    .lcd_param_bits = EXAMPLE_LCD_PARAM_BITS, // 单位 LCD 参数的比特数, 应为 8
↪ 的整数倍
    .spi_mode = 0, // SPI 模式 (0-3), 需根据 LCD 驱动
↪ IC 的数据手册以及硬件的配置确定 (如 IM[3:0])
    .trans_queue_depth = 10, // SPI
↪ 设备传输数据的队列深度, 一般设为 10 即可
    .on_color_trans_done = example_on_color_trans_dome, // 单次调用 `esp_lcd_
↪ panel_draw_bitmap()` 传输完成后的回调函数
    .user_ctx = &example_user_ctx, // 传给回调函数的用户参数
    .flags = { // 以下为 SPI 时序的相关参数, 需根据 LCD 驱动 IC
↪ 的数据手册以及硬件的配置确定
        .sio_mode = 0, // 通过一根数据线 (MOSI) 读写数据, 0: Interface I 型, 1:
↪ Interface II 型
    },
};
ESP_ERROR_CHECK(esp_lcd_new_panel_io_spi((esp_lcd_spi_bus_handle_t)LCD_HOST, &io_
↪ config, &io_handle));

/* 以下函数也可用于注册色彩数据传输完成事件的回调函数 */
// const esp_lcd_panel_io_callbacks_t cbs = {
//     .on_color_trans_done = example_on_color_trans_dome,
// };
// esp_lcd_panel_io_register_event_callbacks(io_handle, &cbs, &example_user_ctx);

```

基于初始化好的 SPI 总线可以创建相应的接口设备, 每个接口设备对应一个 SPI master 设备。

**注意:** 关于 SPI 接口配置参数更加详细的说明, 请参考 [ESP-IDF 编程指南](#)。

通过创建接口设备可以获取数据类型为 `esp_lcd_panel_io_handle_t` 的句柄, 然后能够使用以下 [接口通用 APIs](#) 给 LCD 的驱动 IC 发送 **命令和图像数据**:

1. `esp_lcd_panel_io_tx_param()`: 用于发送单个 LCD 的命令及配套参数, 其内部通过函数 `spi_device_polling_transmit()` 实现数据传输, 使用该函数会等待数据传输完毕后会返回。
2. `esp_lcd_panel_io_tx_color()`: 用于发送单次 LCD 刷屏命令和图像数据。在函数内部, 它通过函数 `spi_device_polling_transmit()` 发送命令和一些少量的参数, 然后通过函数 `spi_device_queue_trans()` 来分包发送大量的图像数据, 每个包的大小由 **SPI 单次 DMA 传输允许的最大字节数**来限制。这个函数将图像缓存地址等相关数据压入队列, 队列的深度由 `trans_queue_depth` 参数指定。一旦数据成功压入队列, 函数就会立刻返回。因此, 如果计划在后续操作中修改相同的图像缓存, 则需要注册一个回调函数来判断上一次的传输是否已经完成。如果不这样做, 可能会在未完成的传输上进行修改, 这会导致由于数据混乱而显示出现错误。

## 移植驱动组件

移植 SPI LCD 驱动组件的基本原理包含以下三点:

1. 基于数据类型为 `esp_lcd_panel_io_handle_t` 的接口设备句柄发送指定格式的命令及参数。
2. 实现并创建一个 LCD 设备, 然后通过注册回调函数的方式实现结构体 `esp_lcd_panel_t` 中的各项功能。
3. 实现一个函数用于提供数据类型为 `esp_lcd_panel_handle_t` 的 LCD 设备句柄, 使得应用程序能够利用 [LCD 通用 APIs](#) 来操作 LCD 设备。

下面是 `esp_lcd_panel_handle_t` 各项功能的实现说明以及和 [LCD 通用 APIs](#) 的对应关系:

功能	LCD 通用 APIs	实现说明
reset()	esp_lcd_panel_reset()	若设备连接了复位引脚，则通过该引脚进行硬件复位，否则通过命令 LCD_CMD_SWRESET (01h) 进行软件复位。
init()	esp_lcd_panel_init()	通过发送一系列的命令及参数来初始化 LCD 设备。
del()	esp_lcd_panel_del()	释放驱动占用的资源，包括申请的存储空间和使用的 IO。
draw_bitmap()	esp_lcd_panel_draw_bitmap()	首先通过命令 LCD_CMD_CASET (2Ah) 和 LCD_CMD_RASET (2Bh) 发送图像的起始和终止坐标，然后通过命令 LCD_CMD_RAMWR (2Ch) 发送图像数据。
mirror()	esp_lcd_panel_mirror()	通过命令 LCD_CMD_MADCTL (36h) 设置是否镜像屏幕的 X 轴和 Y 轴。
swap_xy()	esp_lcd_panel_swap_xy()	通过命令 LCD_CMD_MADCTL (36h) 设置是否交换屏幕的 X 轴和 Y 轴。
set_gap()	esp_lcd_panel_set_gap()	通过软件修改画图时的起始和终止坐标，从而实现画图的偏移。
invert_color()	esp_lcd_panel_invert_color()	通过命令 LCD_CMD_INVON (21h) 和 LCD_CMD_INVOFF (20h) 实现像素的颜色数据按位取反 (0xF0F0 -> 0x0F0F)。
disp_on_off()	esp_lcd_panel_disp_on_off()	通过命令 LCD_CMD_DISON (29h) 和 LCD_CMD_DISOFF (28h) 实现屏幕显示的开关。

对于大多数 SPI LCD，其驱动 IC 的命令及参数与上述实现说明中的兼容，因此可以通过以下步骤完成移植：

1. 在 **LCD 驱动组件** 中选择一个型号相似的 SPI LCD 驱动组件。
2. 通过查阅目标 LCD 驱动 IC 的数据手册，确认其与所选组件中各功能使用到的命令及参数是否一致，若不一致则需要修改相关代码。
3. 即使 LCD 驱动 IC 的型号相同，不同制造商的屏幕也通常需要使用各自提供的初始化命令配置。因此，需要修改初始化函数 init() 中发送的命令和参数。这些初始化命令通常以特定的格式存储在一个静态数组中。此外，需要注意不要在初始化命令中包含一些特殊的命令，例如 LCD\_CMD\_COLMOD (3Ah) 和 LCD\_CMD\_MADCTL (36h)，这些命令是由驱动组件进行管理和使用的。
4. 可使用编辑器的字符搜索和替换功能，将组件中的 LCD 驱动 IC 名称替换为目标名称，如将 gc9a01 替换为 st77916。

## 初始化 LCD 设备

下面以 **GC9A01** 为例的代码说明：

```
#include "esp_lcd_panel_vendor.h" // 依赖的头文件
#include "esp_lcd_panel_ops.h"
#include "esp_lcd_gc9a01.h" // 目标驱动组件的头文件

/**
 * 用于存放 LCD 驱动 IC 的初始化命令及参数
 */
// static const gc9a01_lcd_init_cmd_t lcd_init_cmds[] = {
// // {cmd, { data }, data_size, delay_ms}
// // {0xfe, (uint8_t []){0x00}, 0, 0},
// // {0xef, (uint8_t []){0x00}, 0, 0},
// // {0xeb, (uint8_t []){0x14}, 1, 0},
// // ...
// };

/* 创建 LCD 设备 */
esp_lcd_panel_handle_t panel_handle = NULL;
// const gc9a01_vendor_config_t vendor_config = { // 用于替换驱动组件中的初始化命令及参数
//     .init_cmds = lcd_init_cmds,
//     .init_cmds_size = sizeof(lcd_init_cmds) / sizeof(gc9a01_lcd_init_cmd_t),
// };
esp_lcd_panel_dev_config_t panel_config = {
    .reset_gpio_num = EXAMPLE_PIN_NUM_LCD_RST, // 连接 LCD 复位信号的 IO 编号，可以设为 '-1' 表示不使用
};
```

(下页继续)



(续上页)

```

        .rgb_ele_order = LCD_RGB_ELEMENT_ORDER_RGB, // 像素色彩的元素顺序 (RGB/
↪BGR) ,
                                                    // 一般通过命令 `LCD_CMD_
↪MADCTL (36h)` 控制
        .bits_per_pixel = EXAMPLE_LCD_BIT_PER_PIXEL, //↵
↪色彩格式的位数 (RGB565: 16, RGB666: 18) ,
                                                    // 一般通过命令 `LCD_CMD_
↪COLMOD (3Ah)` 控制
        // .vendor_config = &vendor_config, //↵
↪用于替换驱动组件中的初始化命令及参数
};
ESP_ERROR_CHECK(esp_lcd_new_panel_gc9a01(io_handle, &panel_config, &panel_handle));

/* 初始化 LCD 设备 */
ESP_ERROR_CHECK(esp_lcd_panel_reset(panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_init(panel_handle));
// ESP_ERROR_CHECK(esp_lcd_panel_invert_color(panel_handle, true)); //↵
↪这些函数可以根据需要使用
// ESP_ERROR_CHECK(esp_lcd_panel_mirror(panel_handle, true, true));
// ESP_ERROR_CHECK(esp_lcd_panel_swap_xy(panel_handle, true));
// ESP_ERROR_CHECK(esp_lcd_panel_set_gap(panel_handle, 0, 0));
ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(panel_handle, true));

```

首先通过移植好的驱动组件创建 LCD 设备并获取数据类型为 `esp_lcd_panel_handle_t` 的句柄，然后使用 [LCD 通用 APIs](#) 来初始化 LCD 设备。

下面是一些关于使用函数 `esp_lcd_panel_draw_bitmap()` 刷新 SPI LCD 图像的说明：

- 传入该函数的图像缓存的字节数可以大于 `max_transfer_sz`，此时 `esp_lcd` 驱动内部会根据 SPI 单次 DMA 传输允许的最大字节数进行分包处理。
- 由于该函数是采用 DMA 的方式来传输图像数据，也就是说该函数调用完成后数据仍在通过 DMA 进行传输，此时不能修改正在使用的缓存区域（如进行 LVGL 的渲染）。因此，需要通过总线初始化或者调用 `esp_lcd_panel_io_register_event_callbacks()` 注册的回调函数来判断上一次传输是否完成。
- 由于 SPI 驱动目前不支持直接通过 DMA 传输 PSRAM 上的数据，其内部会判断数据是否存放在 PSRAM 上，若是则会将其拷贝到 SRAM 中再进行传输。因此，推荐使用 SRAM 作为图像的缓存进行传输（如用于 LVGL 渲染的缓存），否则直接传输 PSRAM 上较大的图像数据，很可能会出现 SRAM 不足的情况。

## 相关文档

- [ST7789 数据手册](#)

## 4.1.5 RGB LCD 详解

### 目录

- 术语表
- 接口模式
  - 模式选择
  - DE 模式
  - SYNC 模式
  - 模式对比
- 色彩格式
- RGB LCD 驱动流程



- 初始化接口设备
- 移植驱动组件
- 初始化 *LCD* 设备
- 相关文档

### 术语表

请参阅 *LCD* 术语表。

### 接口模式

大多数 RGB LCD 采用 SPI + RGB 接口，它们需要通过 SPI 接口发送命令对 LCD 进行初始化，也可以在初始化后根据需要动态修改相关配置，如垂直/水平镜像，更具灵活性。一些 RGB LCD 仅采用 RGB 接口，它们无需发送命令对 LCD 进行初始化，但也无法修改任何配置，驱动方法更加简单。下图为 *ST7701S* 的接口类型选择：

IM3	IM2	IM1	IM0	Interface	Data pins
0	0	0	1	RGB+8b_SPI(fall)	D[0~23]
	0	1	0	RGB+9b_SPI(fall)	D[0~23]
	0	1	1	RGB+16b_SPI(rise)	D[0~23]
	1	0	1	MIPI	HSSI_D1_P/N,HSSI_D0_P/N
	1	1	0	MIPI+16b_SPI(rise)	HSSI_D1_P/N,HSSI_D0_P/N
1	0	0	1	RGB+8b_SPI(rise)	D[0~23]
	0	1	0	RGB+9b_SPI(rise)	D[0~23]
	0	1	1	RGB+16b_SPI(fall)	D[0~23]
	1	0	1	MIPI	HSSI_D1_P/N,HSSI_D0_P/N
	1	1	0	MIPI+16b_SPI(fall)	HSSI_D1_P/N,HSSI_D0_P/N

Table 10 Interface Type Selection

图 22: *ST7701S* 的接口类型选择

从上图中可以看出，*ST7701S* 是通过 IM[3:0] 引脚来选择 SPI + RGB 接口的配置。通常来说，这类 LCD 会选择 3-wire SPI + RGB 的接口类型，对应于上图中的 RGB+9b\_SPI(rise/fall)，其中，9b\_SPI 表示 SPI 接口的 *3-line 模式* (一般称为 3-wire)，rise/fall 表示 SCL 信号的有效边沿，rise 表示上升沿有效 (SPI 模式 0/3)，fall 表示下降沿 (SPI 模式 1/2)。

下图为 *ST7701S* 的 SPI 和 RGB 接口的引脚描述：

注：RGB 引脚名称：CS、SCK(SCL)、SDA(MOSI)、HSYNC、VSYNC、PCLK、DE、D[23:0](D[17:0]/D[7:0])

对于采用 SPI + RGB 接口的 LCD，一般可以通过命令配置 RGB 接口为 DE 模式或者 SYNC 模式，下面以 *ST7701S* 为例介绍这两种模式。

**模式选择** 从图中可以看出，*ST7701S* 可以通过命令 C3h 配置 RGB 的模式。需注意，不同型号的 LCD 驱动 IC 可能使用不同的命令，如 *GC9503* 是通过命令 B0h 进行配置的。

### DE 模式

### SYNC 模式

3-line serial interface (9 bits)

Pin Name	Description
CSX	Chip selection signal
SCL	Serial input CLK
SDA	Serial input data
SDO	Serial output data

4-line serial interface (8 bits)

Pin Name	Description
CSX	Chip selection signal
DCX	Data is regarded as a command when SCL is low Data is regarded as a parameter or data when SCL is high
SCL	Clock signal
SDA	Serial input data
SDO	Serial output data

图 23: ST7701S SPI 接口的引脚描述

Symbol	Name	Description
PCLK	Pixel clock	Pixel clock for capturing pixels at display interface
HS	Horizontal sync	Horizontal synchronization timing signal
VS	Vertical sync	Vertical synchronization timing signal
DE	Data enable	Data enable signal (assertion indicates valid pixels)
DB[23:0]	Pixel data	Pixel data in 16-bit, 18-bit and 24-bit format

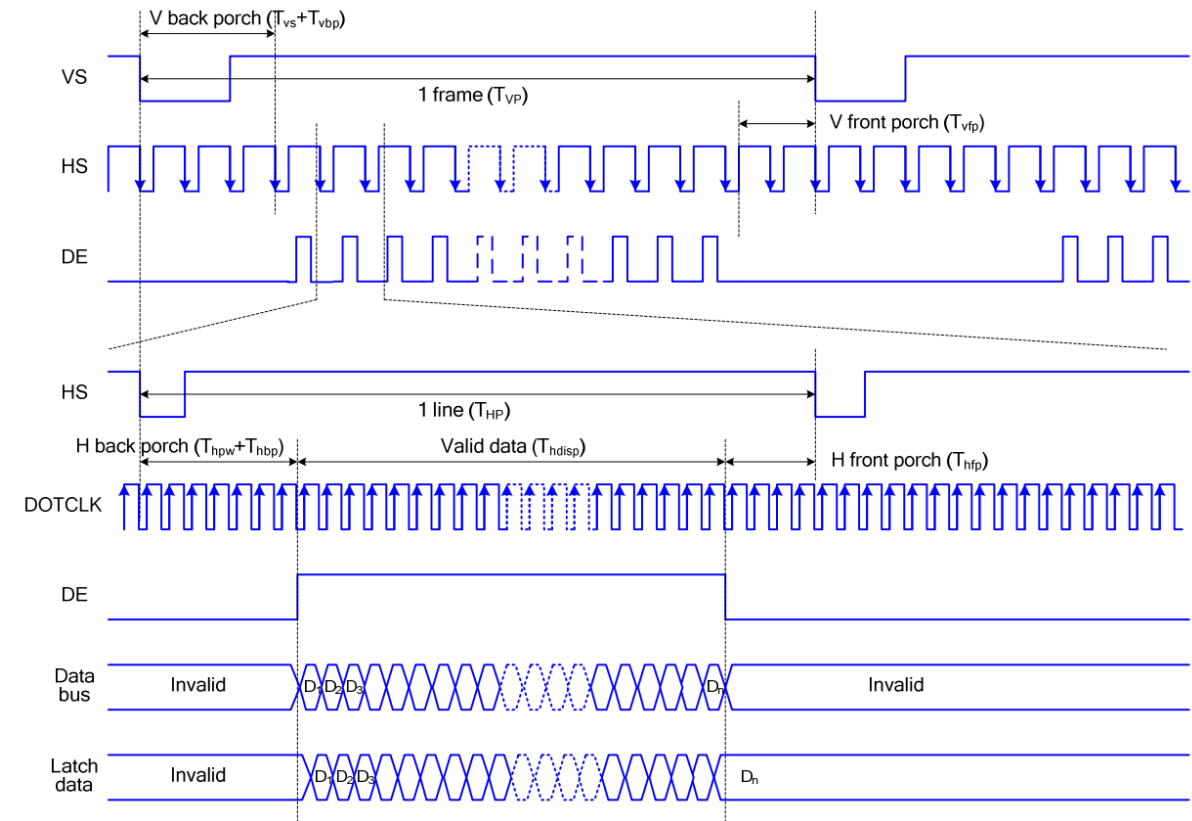
Table 11 The interface signals of RGB interface

图 24: ST7701S RGB 接口的引脚描述

ST7701S supports two kinds of RGB interface, DE mode and HV mode. The table shown below uses command C3h to select RGB interface mode.

DE/Sync	RGB Mode
0	DE mode
1	HV mode

图 25: ST7701S RGB 接口的模式选择



Note: The setting of front porch and back porch in host must match that in IC as this mode.

Figure 23 Timing Chart of Signals in RGB Interface DE Mode

图 26: ST7701S DE 模式的时序图

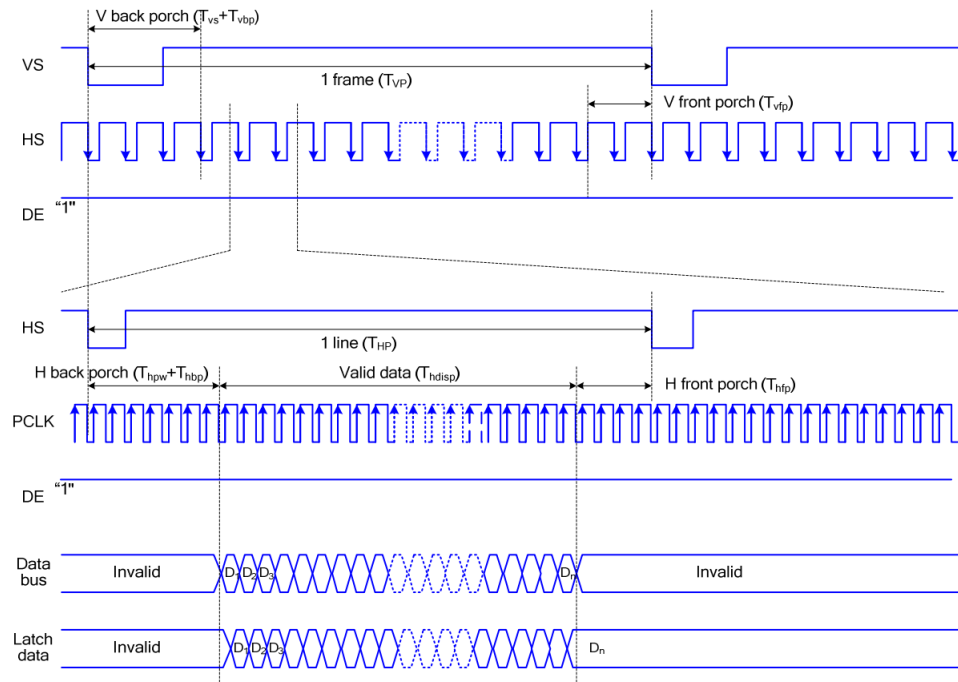


Figure 24 Timing chart of RGB interface HV mod

图 27: ST7701S SYNC 模式的时序图

**模式对比** 通过对比 DE 模式和 SYNC 模式的时序图，可以看出它们的主要区别在于是否使用 DE 信号线以及对于消隐区域（Blanking Porch）的配置要求，总结为下表：

模式	是否使用 DE 信号线	是否配置消隐区域寄存器	ESP 是否支持
DE 模式	是	否	是
SYNC 模式	否	是	是

### 色彩格式

大多数 RGB LCD 支持多种色彩（输入数据）格式，包括 RGB565、RGB666、RGB888 等，通常可以使用 COLMOD (3Ah) 命令来配置。下图为 ST7701S 的色彩格式配置：

Pad name	24 bits configuration VIPF[3:0]=0111	18 bits configuration VIPF[3:0]=0110		16 bits configuration VIPF[3:0]=0101
		MDT=0	MDT=1	
DB[23]	R7	Not used	Not used	Not used
DB[22]	R6	Not used	Not used	Not used
DB[21]	R5	R5	Not used	Not used
DB[20]	R4	R4	Not used	R4
DB[19]	R3	R3	Not used	R3
DB[18]	R2	R2	Not used	R2
DB[17]	R1	R1	R5	R1
DB[16]	R0	R0	R4	R0
DB[15]	G7	Not used	R3	Not used
DB[14]	G6	Not used	R2	Not used
DB[13]	G5	G5	R1	G5
DB[12]	G4	G4	R0	G4
DB[11]	G3	G3	G5	G3
DB[10]	G2	G2	G4	G2
DB[09]	G1	G1	G3	G1
DB[08]	G0	G0	G2	G0
DB[07]	B7	Not used	G1	Not used
DB[06]	B6	Not used	G0	Not used
DB[05]	B5	B5	B5	Not used
DB[04]	B4	B4	B4	B4
DB[03]	B3	B3	B3	B3
DB[02]	B2	B2	B2	B2
DB[01]	B1	B1	B1	B1
DB[00]	B0	B0	B0	B0

Table 12 The interface color mapping of RGB interface

图 28: ST7701S 的色彩格式配置

从上图可以看出，ST7701S 支持 16-bit RGB565、18-bit RGB666、24-bit RGB888 三种色彩格式，其中 N-bit 表示接口的数据线位数，并且是通过 COLMOD (3Ah) : VIPF[2:0] 和 COLCTRL (CDh) : MDT 命令来进行选择。需注意，命令配置需要与硬件接口保持一致，例如 LCD 模块仅提供了 18-bit 的数据线，那么软件一定不能配置色彩格式为 24-bit RGB888，并且在此情况下只有在数据线为 D[21:16]，D[13:8]，D[5:0] 时才能配置为 16-bit RGB565。

除此之外，色彩格式的位数并不等于接口的有效数据线位数，下图为 *ST77903* 的接口类型选择和色彩格式配置：

IM	3Ah	RGB Interface Mode	Data pins
1,0	101	3-SPI with RGB565	DB[7:2]
1,0	110	3-SPI with RGB666	DB[7:2]
1,0	111	3-SPI with RGB888	DB[7:0]
1,1	101	4-SPI with RGB565	DB[7:2]
1,1	110	4-SPI with RGB666	DB[7:2]
1,1	111	4-SPI with RGB888	DB[7:0]

图 29: ST77903 RGB 接口的类型选择

从上图可以看出，*ST77903* 支持 6-bit RGB565、6-bit RGB666 和 8-bit RGB888 三种色彩格式，而它们的位数分别为 16-bit、18-bit 和 24-bit。多数 LCD 的 RGB 接口仅需一个时钟周期即可并行传输单个像素的色彩数据，而像 *ST77903* 这类 LCD 接口则需要多个时钟周期传输单个像素的色彩数据，所以这类接口也被称为 **串行 RGB 接口 (SRGB)**。

**备注：**虽然 ESP32-S3 仅支持 16-bit RGB565 和 8-bit RGB888 两种色彩格式，但是通过特殊的硬件连接方式可以使其驱动支持 18-bit RGB666 或 24-bit RGB888 色彩格式的 LCD，连接方式请参考开发板 [ESP32-S3-LCD-EV-Board](#) 的 [LCD 子板 2 \(3.95' LCD\\_QMZX\)](#) 和 [LCD 子板 3](#) 原理图。

## RGB LCD 驱动流程

RGB LCD 驱动流程可大致分为三个部分：初始化接口设备、移植驱动组件和初始化 LCD 设备。

### 初始化接口设备

下面是使用 `esp_lcd_panel_io_additions` 组件来创建 3-wire SPI 接口设备的代码说明：

```
#include "esp_check.h"           // 依赖的头文件
#include "esp_lcd_panel_io.h"
#include "esp_lcd_panel_io_additions.h"

esp_lcd_panel_io_3wire_spi_config_t io_config = {
    .line_config = {
        .cs_io_type = IO_TYPE_GPIO,           // 设置为 `IO_TYPE_EXPANDER`
        ↪表示使用 IO 扩展芯片的引脚，否则使用 GPIO
        .cs_gpio_num = EXAMPLE_LCD_IO_SPI_CS, // 连接 LCD CS 信号的 GPIO 编号
        // .cs_expander_pin = EXAMPLE_LCD_IO_SPI_CS, // 连接 LCD CS 信号的扩展
        ↪IO 芯片引脚编号
        .scl_io_type = IO_TYPE_GPIO,           // 设置为 `IO_TYPE_EXPANDER`
        ↪表示使用 IO 扩展芯片的引脚，否则使用 GPIO
        .scl_gpio_num = EXAMPLE_LCD_IO_SPI_SCK, // 连接 LCD SCK (SCL) 信号的
        ↪GPIO 编号
        // .scl_expander_pin = EXAMPLE_LCD_IO_SPI_SCK, // 连接 LCD
        ↪SCK (SCL) 信号的扩展 IO 芯片引脚编号
        .sda_io_type = IO_TYPE_GPIO,           // 设置为 `IO_TYPE_EXPANDER`
        ↪表示使用 IO 扩展芯片的引脚，否则使用 GPIO
        .sda_gpio_num = EXAMPLE_LCD_IO_SPI_SDO, // 连接 LCD MOSI (SDO、SDA)
        ↪信号的 GPIO 编号
    }
};
```

(下页继续)

serial RGB 565					
Pin	1 <sup>st</sup> Data	2 <sup>nd</sup> Data	3 <sup>rd</sup> Data	...	(3N+1) <sup>th</sup> Data
DAP[0]	x	x	x	...	x
DAP[1]	x	x	x	...	x
DAP[2]	x	1'G0	x	...	x
DAP[3]	1'R0	1'G1	1'B0	...	N'R0
DAP[4]	1'R1	1'G2	1'B1	...	N'R1
DAP[5]	1'R2	1'G3	1'B2	...	N'R2
DAP[6]	1'R3	1'G4	1'B3	...	N'R3
DAP[7]	1'R4	1'G5	1'B4	...	N'R4

serial RGB 666					
Pin	1 <sup>st</sup> Data	2 <sup>nd</sup> Data	3 <sup>rd</sup> Data	...	(3N+1) <sup>th</sup> Data
DAP[0]	x	x	x	...	x
DAP[1]	x	x	x	...	x
DAP[2]	1'R0	1'G0	1'B0	...	N'R0
DAP[3]	1'R1	1'G1	1'B1	...	N'R1
DAP[4]	1'R2	1'G2	1'B2	...	N'R2
DAP[5]	1'R3	1'G3	1'B3	...	N'R3
DAP[6]	1'R4	1'G4	1'B4	...	N'R4
DAP[7]	1'R5	1'G5	1'B5	...	N'R5

serial RGB 888					
Pin	1 <sup>st</sup> Data	2 <sup>nd</sup> Data	3 <sup>rd</sup> Data	...	(3N+1) <sup>th</sup> Data
DAP[0]	1'R0	1'G0	1'B0	...	N'R0
DAP[1]	1'R1	1'G1	1'B1	...	N'R1
DAP[2]	1'R2	1'G2	1'B2	...	N'R2
DAP[3]	1'R3	1'G3	1'B3	...	N'R3
DAP[4]	1'R4	1'G4	1'B4	...	N'R4
DAP[5]	1'R5	1'G5	1'B5	...	N'R5
DAP[6]	1'R6	1'G6	1'B6	...	N'R6
DAP[7]	1'R7	1'G7	1'B7	...	N'R7

图 30: ST77903 的色彩格式配置

(续上页)

```

        // .sda_expander_pin = EXAMPLE_LCD_IO_SPI_SDO, // 连接 LCD_
        ↪ MOSI (SDO、SDA) 信号的扩展 IO 芯片引脚编号
        .io_expander = NULL, // 若使用 IO_
        ↪ 扩展芯片的引脚，则需要传入已经初始化好的设备句柄
    },
    .expect_clk_speed = PANEL_IO_3WIRE_SPI_CLK_MAX, // 期望的 SPI_
    ↪ 时钟频率，由于采用软件模拟的方式，实际可能有较大误差，
        // 默认设为 `PANEL_IO_3WIRE_
    ↪ SPI_CLK_MAX` 即可
    .spi_mode = 0, // SPI 模式 (0-3)，需根据 LCD 驱动 IC_
    ↪ 的数据手册以及硬件的配置确定 (如 IM[3:0])
    .lcd_cmd_bytes = 1, // 单位 LCD 命令的字节数 (1-4)，通常设为 `1`_
    ↪ 即可
    .lcd_param_bytes = 1, // 单位 LCD 参数的字节数 (1-4)，通常设为 `1`_
    ↪ 即可
    .flags = {
        .use_dc_bit = 1, // 默认设为 `1` 即可
        .del_keep_cs_inactive = 1, // 默认设为 `1` 即可
    },
}
esp_lcd_panel_io_handle_t io_handle = NULL;
ESP_ERROR_CHECK(esp_lcd_new_panel_io_3wire_spi(&io_config, &io_handle));

```

对于仅采用 RGB 接口的 LCD，因为它们不支持传输命令及参数，所以这里不需要初始化接口设备，请直接参考[初始化 LCD 设备](#)。

对于采用 3-wire SPI 和 RGB 接口的 LCD，这里仅需创建 3-wire SPI 接口设备。由于 ESP 的 SPI 外设不支持直接传输 9-bit 数据，并且该接口仅用于传输数据量较小的命令及参数，而且对于数据传输的带宽及时序要求不高，因此可以使用 GPIO 或者 IO 扩展芯片引脚（如 TCA9554）通过软件模拟 SPI 协议的方式来实现。

通过创建接口设备可以获取数据类型为 `esp_lcd_panel_io_handle_t` 的句柄，然后能够使用 `esp_lcd_panel_io_tx_param()` 给 LCD 的驱动 IC 发送命令。

### 移植驱动组件

对于仅采用 RGB 接口的 LCD，由于 RGB 接口驱动中已经通过注册回调函数的方式实现了结构体 `esp_lcd_panel_t` 中的各项功能，并且提供了函数 `esp_lcd_new_rgb_panel()` 用于创建数据类型为 `esp_lcd_panel_handle_t` 的 LCD 设备，使得应用程序能够使用 LCD 通用 APIs 来操作 LCD 设备。因此，这种 LCD 不需要移植驱动组件，请直接参考[初始化 LCD 设备](#)。

对于采用 3-wire SPI 和 RGB 接口的 LCD，在上述 RGB 接口驱动的基础上，还需要通过 3-wire SPI 接口发送命令及参数。因此，实现这种 LCD 驱动组件的基本原理包含以下三点：

1. 基于数据类型为 `esp_lcd_panel_io_handle_t` 的接口设备发送指定格式的命令及参数。
2. 使用函数 `esp_lcd_new_rgb_panel()` 创建一个 LCD 设备，然后通过注册回调函数的方式保存和覆盖该设备中的部分功能。
3. 实现一个函数用于提供数据类型为 `esp_lcd_panel_handle_t` 的 LCD 设备句柄，使得应用程序能够利用 LCD 通用 APIs 来操作 LCD 设备。

下面是 `esp_lcd_panel_handle_t` 各项功能的实现说明以及和 RGB 接口驱动还有 LCD 通用 APIs 的对应关系：



功能	RGB 接口驱动	LCD 通用 APIs	实现说明
reset()	rgb_panel_reset()	esp_lcd_panel_reset()	若设备连接了复位引脚，则通过该引脚进行硬件复位，否则通过命令 LCD_CMD_SWRESET (01h) 进行软件复位，最后使用 rgb_panel_reset() 复位 RGB 接口。
init()	rgb_panel_init()	esp_lcd_panel_init()	若 3-wire SPI 接口没有与 RGB 接口复用引脚，则通过发送一系列的命令及参数来初始化 LCD 设备，否则需要提前在 LCD 创建时进行初始化，最后使用 rgb_panel_init() 初始化 RGB 接口。
del()	rgb_panel_del()	esp_lcd_panel_del()	释放驱动占用的资源，包括申请的存储空间和使用的 IO，还要使用 rgb_panel_del() 删除 RGB 接口。
draw_bitmap()	rgb_panel_draw_bitmap()	esp_lcd_panel_draw_bitmap()	需保存和覆盖，使用 rgb_panel_draw_bitmap() 发送图像数据。
mirror()	rgb_panel_mirror()	esp_lcd_panel_mirror()	根据用户配置，既可以通过命令，也可以使用 rgb_panel_mirror() 通过软件实现镜像 X 轴和 Y 轴。
swap_xy()	rgb_panel_swap_xy()	esp_lcd_panel_swap_xy()	无需保存和覆盖，使用 rgb_panel_swap_xy() 通过软件实现交换 X 轴和 Y 轴。
set_gap()	rgb_panel_set_gap()	esp_lcd_panel_set_gap()	无需保存和覆盖，使用 rgb_panel_set_gap() 通过软件修改画图时的起始和终止坐标，从而实现画图的偏移。
invert_color()	rgb_panel_invert_color()	esp_lcd_panel_invert_color()	需保存和覆盖，使用 rgb_panel_invert_color() 通过硬件实现像素的色彩数据按位取反 (0xF0F0 -> 0x0F0F)。
disp_on_off()	rgb_panel_disp_on_off()	esp_lcd_panel_disp_on_off()	根据用户配置来实现 LCD 显示的开关。如果没有配置 disp_gpio_num，则可以通过 LCD 命令 LCD_CMD_DISON (29h) 和 LCD_CMD_DISOFF (28h) 来进行控制。另外，如果配置了 disp_gpio_num，则可以通过调用函数 rgb_panel_disp_on_off() 来实现控制。

对于大多数 RGB LCD，其驱动 IC 的命令及参数与上述实现说明中的兼容，因此可以通过以下步骤完成移植：

1. 在 [LCD 驱动组件](#) 中选择一个型号相似的 RGB LCD 驱动组件。
2. 通过查阅目标 LCD 驱动 IC 的数据手册，确认其与所选组件中各功能使用到的命令及参数是否一致，若不一致则需要修改相关代码。
3. 即使 LCD 驱动 IC 的型号相同，不同制造商的屏幕也通常需要使用各自提供的初始化命令配置。因此，需要修改初始化函数 init() 中发送的命令和参数。这些初始化命令通常以特定的格式存储在一个静态数组中。此外，需要注意不要在初始化命令中包含由驱动 IC 控制的命令，例如 LCD\_CMD\_COLMOD (3Ah)，以确保成功初始化 LCD 设备。
4. 可使用编辑器的字符搜索和替换功能，将组件中的 LCD 驱动 IC 名称替换为目标名称，如将 gc9503 替换为 st7701。

## 初始化 LCD 设备

下面是以 ESP-IDF release/v5.1 中 [rgb\\_panel](#) 为例的代码说明：

```
#include "esp_check.h"           // 依赖的头文件
#include "esp_lcd_panel_ops.h"
#include "esp_lcd_panel_rgb.h"

esp_lcd_panel_handle_t panel_handle = NULL;
```

(下页继续)



(续上页)

```

esp_lcd_rgb_panel_config_t panel_config = {    // RGB 接口的配置参数
    .data_width = EXAMPLE_LCD_DATA_WIDTH,      // RGB
    ↪接口的数据线位数, 如 `16-bit RGB565`: 16, `8-bit RGB888`: 8
    .bits_per_pixel = EXAMPLE_LCD_BIT_PER_PIXEL, // 色彩格式的位数, 可能与
    ↪RGB 接口的数据线位数不相等,
                                                    // 如 `16-bit RGB565`:
    ↪16, `8-bit RGB888`: 24
    .psram_trans_align = 64,                    // 默认设为 `64` 即可
    .num_fbs = EXAMPLE_LCD_NUM_FB,             // RGB
    ↪接口的帧缓存数, 默认设为 `1`, 大于 `1` 时用于实现多缓冲防撕裂
    .bounce_buffer_size_px = 10 * EXAMPLE_LCD_H_RES, // 用于提升 RGB
    ↪接口的数据传输带宽, 通常设为 `10 * EXAMPLE_LCD_H_RES`
    .clk_src = LCD_CLK_SRC_DEFAULT,            // 默认设为 `LCD_CLK_SRC_
    ↪DEFAULT` 即可
    .disp_gpio_num = EXAMPLE_PIN_NUM_DISP_EN,  // 连接 LCD DISP
    ↪信号的引脚编号, 可以设置为 `-1` 表示不使用
    .pclk_gpio_num = EXAMPLE_PIN_NUM_PCLK,     // 连接 LCD PCLK
    ↪信号的引脚编号
    .vsync_gpio_num = EXAMPLE_PIN_NUM_VSYNC,   // 连接 LCD VSYNC
    ↪信号的引脚编号
    .hsync_gpio_num = EXAMPLE_PIN_NUM_HSYNC,   // 连接 LCD HSYNC
    ↪信号的引脚编号
    .de_gpio_num = EXAMPLE_PIN_NUM_DE,         // 连接 LCD DE
    ↪信号的引脚编号, 可以设置为 `-1` 表示不使用
    .data_gpio_nums = {                       // 连接 LCD D[15:0]
    ↪信号的引脚编号, 有效数量由 `data_width` 指定,
                                                    // 8-bit 时设置 D[7:0] 即可

        EXAMPLE_PIN_NUM_DATA0,
        EXAMPLE_PIN_NUM_DATA1,
        EXAMPLE_PIN_NUM_DATA2,
        EXAMPLE_PIN_NUM_DATA3,
        EXAMPLE_PIN_NUM_DATA4,
        EXAMPLE_PIN_NUM_DATA5,
        EXAMPLE_PIN_NUM_DATA6,
        EXAMPLE_PIN_NUM_DATA7,
        EXAMPLE_PIN_NUM_DATA8,
        EXAMPLE_PIN_NUM_DATA9,
        EXAMPLE_PIN_NUM_DATA10,
        EXAMPLE_PIN_NUM_DATA11,
        EXAMPLE_PIN_NUM_DATA12,
        EXAMPLE_PIN_NUM_DATA13,
        EXAMPLE_PIN_NUM_DATA14,
        EXAMPLE_PIN_NUM_DATA15,
    },
    .timings = {                                // 以下为 RGB 时序的相关参数, 需根据 LCD 驱动 IC
    ↪的数据手册以及硬件的配置确定
        .pclk_hz = EXAMPLE_LCD_PIXEL_CLOCK_HZ,
        .h_res = EXAMPLE_LCD_H_RES,
        .v_res = EXAMPLE_LCD_V_RES,
        .hsync_back_porch = 40,                // 在 DE 模式下, HSYNC 和 VSYNC
    ↪的相关参数可以根据期望的刷新率进行调整
        .hsync_front_porch = 20,              // 在 SYNC 模式下, HSYNC 和 VSYNC
    ↪的相关参数需要和软件初始化命令中的配置保持一致
        .hsync_pulse_width = 1,
        .vsync_back_porch = 8,
        .vsync_front_porch = 4,
        .vsync_pulse_width = 1,
        .flags = {                             // 由于一些 LCD
    ↪可以通过硬件引脚配置这些参数, 需要确保它们与配置保持一致, 但通常情况下均为 `0`
        .hsync_idle_low = 0,                  // HSYNC 信号空闲时的电平, 0: 高电平, 1: 低电平
        .vsync_idle_low = 0,                  // VSYNC 信号空闲时的电平, 0
    ↪表示高电平, 1: 低电平

```

(下页继续)

(续上页)

```

        .de_idle_high = 0,          // DE 信号空闲时的电平, 0: 高电平, 1: 低电平
        .pclk_active_neg = 0,      // _
    ↪ 时钟信号的有效边沿, 0: 上升沿有效, 1: 下降沿有效
        .pclk_idle_high = 0,       // PCLK 信号空闲时的电平, 0: 高电平, 1: 低电平
    },
},
    .flags.fb_in_psram = 1,         // 默认设置为 `1` 即可
};
ESP_ERROR_CHECK(esp_lcd_new_rgb_panel(&panel_config, &panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_reset(panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_init(panel_handle));

/* 以下函数可以根据需要调用 */
// ESP_ERROR_CHECK(esp_lcd_panel_invert_color(panel_handle, true)); // _
    ↪ 通过硬件实现像素的色彩数据按位取反 (0xF0F0 -> 0x0F0F)
// ESP_ERROR_CHECK(esp_lcd_panel_mirror(panel_handle, true, true)); // _
    ↪ 通过软件实现镜像 X 轴和 Y 轴
// ESP_ERROR_CHECK(esp_lcd_panel_swap_xy(panel_handle, true)); // _
    ↪ 通过软件实现交换 X 轴和 Y 轴
// ESP_ERROR_CHECK(esp_lcd_panel_set_gap(panel_handle, 0, 0)); // _
    ↪ 通过软件修改画图时的起始和终止坐标, 从而实现画图的偏移
// ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(panel_handle, true)); // 通过 _
    ↪ `disp_gpio_num` 引脚控制 LCD 显示的开关,
                                                                    // _
    ↪ 仅当该引脚设置且不为 `-1` 时可用, 否则会报错

```

对于采用 **3-wire SPI** 和 **RGB 接口** 的 **LCD**, 首先通过 **RGB 接口驱动** 中的 `esp_lcd_new_rgb_panel()` 函数创建 LCD 设备并获取数据类型为 `esp_lcd_panel_handle_t` 的句柄, 然后使用 **LCD 通用 APIs** 来初始化 LCD 设备。

关于 RGB 接口的参数配置和一些功能函数的说明, 请参考 **RGB 参数配置及功能函数**

下面是以 **ST7701S** 为例的代码说明:

```

#include "esp_check.h"          // 依赖的头文件
#include "esp_lcd_panel_ops.h"
#include "esp_lcd_panel_rgb.h"
#include "esp_lcd_panel_vendor.h"
#include "esp_lcd_st7701.h"     // 目标驱动组件的头文件

/**
 * 用于存放 LCD 驱动 IC 的初始化命令及参数
 */
// static const st7701_lcd_init_cmd_t lcd_init_cmds[] = {
// //   cmd   data           data_size  delay_ms
// //   {0xFF, (uint8_t []){0x77, 0x01, 0x00, 0x00, 0x13}, 5, 0},
// //   {0xEF, (uint8_t []){0x08}, 1, 0},
// //   {0xFF, (uint8_t []){0x77, 0x01, 0x00, 0x00, 0x10}, 5, 0},
// //   {0xC0, (uint8_t []){0x3B, 0x00}, 2, 0},
// //   ...
// // };

/* 创建 LCD 设备 */
esp_lcd_rgb_panel_config_t rgb_config = {    // RGB 接口的配置参数
    .data_width = EXAMPLE_LCD_DATA_WIDTH,    // RGB _
    ↪ 接口的数据线位数, 如 `16-bit RGB565`: 16, `8-bit RGB888`: 8
    .bits_per_pixel = EXAMPLE_LCD_BIT_PER_PIXEL, // 色彩格式的位数, 可能与 _
    ↪ RGB 接口的数据线位数不相等,
                                                    // 如 `16-bit RGB565`: _
    ↪ 16, `8-bit RGB888`: 24
    .psram_trans_align = 64,                    // 默认设为 `64` 即可
    .num_fbs = EXAMPLE_LCD_NUM_FB,             // RGB _
    ↪ 接口的帧缓存数量, 默认设为 `1`, 大于 `1` 时用于实现多缓冲防撕裂
}

```

(下页继续)

(续上页)

```

        .bounce_buffer_size_px = 10 * EXAMPLE_LCD_H_RES, // 用于提升 RGB
    ↪接口的数据传输带宽, 通常设为 `10 * EXAMPLE_LCD_H_RES`
        .clk_src = LCD_CLK_SRC_DEFAULT, // 默认设为 `LCD_CLK_SRC_
    ↪DEFAULT` 即可
        .disp_gpio_num = EXAMPLE_PIN_NUM_DISP_EN, // 连接 LCD DISP
    ↪信号的引脚编号, 可以设置为 -1 表示不使用
        .pclk_gpio_num = EXAMPLE_PIN_NUM_PCLK, // 连接 LCD PCLK
    ↪信号的引脚编号
        .vsync_gpio_num = EXAMPLE_PIN_NUM_VSYNC, // 连接 LCD VSYNC
    ↪信号的引脚编号
        .hsync_gpio_num = EXAMPLE_PIN_NUM_HSYNC, // 连接 LCD HSYNC
    ↪信号的引脚编号
        .de_gpio_num = EXAMPLE_PIN_NUM_DE, // 连接 LCD DE
    ↪信号的引脚编号, 可以设置为 -1 表示不使用
        .data_gpio_nums = { // 连接 LCD D[15:0]
    ↪信号的引脚编号, 有效数量由 `data_width` 指定,
        EXAMPLE_PIN_NUM_DATA0,
        EXAMPLE_PIN_NUM_DATA1,
        EXAMPLE_PIN_NUM_DATA2,
        EXAMPLE_PIN_NUM_DATA3,
        EXAMPLE_PIN_NUM_DATA4,
        EXAMPLE_PIN_NUM_DATA5,
        EXAMPLE_PIN_NUM_DATA6,
        EXAMPLE_PIN_NUM_DATA7,
        EXAMPLE_PIN_NUM_DATA8,
        EXAMPLE_PIN_NUM_DATA9,
        EXAMPLE_PIN_NUM_DATA10,
        EXAMPLE_PIN_NUM_DATA11,
        EXAMPLE_PIN_NUM_DATA12,
        EXAMPLE_PIN_NUM_DATA13,
        EXAMPLE_PIN_NUM_DATA14,
        EXAMPLE_PIN_NUM_DATA15,
    },
        .timings = { // 以下为 RGB 时序的相关参数, 需根据 LCD 驱动 IC
    ↪的数据手册以及软硬件的配置确定
        .pclk_hz = EXAMPLE_LCD_PIXEL_CLOCK_HZ,
        .h_res = EXAMPLE_LCD_H_RES,
        .v_res = EXAMPLE_LCD_V_RES,
        .hsync_back_porch = 40, // 在 DE 模式下, HSYNC 和 VSYNC
    ↪的相关参数可以根据期望的刷新率进行调整
        .hsync_front_porch = 20, // 在 SYNC 模式下, HSYNC 和 VSYNC
    ↪的相关参数需要和软件初始化命令中的配置保持一致
        .hsync_pulse_width = 1,
        .vsync_back_porch = 8,
        .vsync_front_porch = 4,
        .vsync_pulse_width = 1,
        .flags = { // 由于一些 LCD
    ↪可以通过硬件引脚或者软件命令配置这些参数, 需要确保它们与配置保持一致, 但通常情况下均为
    ↪`0`
        .hsync_idle_low = 0, // HSYNC 信号空闲时的电平, 0: 高电平, 1: 低电平
        .vsync_idle_low = 0, // VSYNC 信号空闲时的电平, 0
    ↪表示高电平, 1: 低电平
        .de_idle_high = 0, // DE 信号空闲时的电平, 0: 高电平, 1: 低电平
        .pclk_active_neg = 0, //
    ↪时钟信号的有效边沿, 0: 上升沿有效, 1: 下降沿有效
        .pclk_idle_high = 0, // PCLK 信号空闲时的电平, 0: 高电平, 1: 低电平
    },
    },
        .flags.fb_in_psram = 1, // 默认设置为 `1` 即可
    };

```

(下页继续)

(续上页)

```

st7701_vendor_config_t vendor_config = {
    .rgb_config = &rgb_config,      // RGB 接口的配置参数
    // .init_cmds = lcd_init_cmds,    // 用于替换驱动组件中的初始化命令及参数
    // .init_cmds_size = sizeof(lcd_init_cmds) / sizeof(st7701_lcd_init_cmd_t),
    .flags = {                      // LCD 驱动 IC 的配置参数
        .mirror_by_cmd = 1,          // 若为 `1` 则使用 LCD 命令实现镜像功能 (esp_lcd_
        ↪ panel_mirror()), 若为 `0` 则通过软件实现
        .enable_io_multiplex = 0,    // 若为 `1` 则在删除 LCD_
        ↪ 设备时自动删除接口设备, 此时应设置所有名称为 `*_by_cmd` 的参数为 `0`,
        // 若为 `0` 则不删除。如果 3-wire SPI_
        ↪ 接口的引脚与 RGB 接口的复用, 那么需要设置此参数为 `1`
    },
};

const esp_lcd_panel_dev_config_t panel_config = {
    .reset_gpio_num = EXAMPLE_LCD_IO_RST,      // 连接 LCD 复位信号的 IO_
    ↪ 编号, 可以设为 `-1` 表示不使用
    .rgb_ele_order = LCD_RGB_ELEMENT_ORDER_RGB, // 像素色彩的元素顺序 (RGB/
    ↪ BGR) ,
    // 一般通过命令 `LCD_CMD_
    ↪ MADCTL (36h)` 控制
    .bits_per_pixel = EXAMPLE_LCD_BIT_PER_PIXEL, // _
    ↪ 色彩格式的位数 (RGB565: 16, RGB666: 18, RGB888: 24) ,
    // 一般通过命令 `LCD_CMD_
    ↪ COLMOD (3Ah)` 控制
    .vendor_config = &vendor_config,           // RGB 接口及 LCD 驱动 IC_
    ↪ 的配置参数
};

esp_lcd_panel_handle_t panel_handle = NULL;
ESP_ERROR_CHECK(esp_lcd_new_panel_st7701(io_handle, &panel_config, &panel_handle));

/* 初始化 LCD 设备 */
ESP_ERROR_CHECK(esp_lcd_new_rgb_panel(&panel_config, &panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_reset(panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_init(panel_handle));
// 以下函数可以根据需要使用
// ESP_ERROR_CHECK(esp_lcd_panel_invert_color(panel_handle, true));
// ESP_ERROR_CHECK(esp_lcd_panel_mirror(panel_handle, true, true));
// ESP_ERROR_CHECK(esp_lcd_panel_swap_xy(panel_handle, true));
// ESP_ERROR_CHECK(esp_lcd_panel_set_gap(panel_handle, 0, 0));
// ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(panel_handle, true));

```

对于采用 3-wire SPI 和 RGB 接口的 LCD，首先通过移植好的驱动组件创建 LCD 设备并获取数据类型为 `esp_lcd_panel_handle_t` 的句柄，然后使用 **LCD 通用 APIs** 来初始化 LCD 设备。

关于 RGB 接口配置参数更加详细的说明，请参考 **ESP-IDF 编程指南**。下面是一些关于使用函数 `esp_lcd_panel_draw_bitmap()` 刷新 RGB LCD 图像的说明：

- 该函数是通过内存拷贝的方式刷新帧缓存里的图像数据，也就是说该函数调用完成后帧缓存内的图像数据也已经更新完成，而 RGB 接口本身是通过 DMA 从帧缓存中获取图像数据来刷新 LCD，这两个过程是异步进行的。
- 该函数会判断传入参数 `color_data` 的值是否为 RGB 接口内部的帧缓存地址，若是，则不会进行上述的内存拷贝操作，而是直接将 RGB 接口的 DMA 传输地址设置为该缓存地址，从而在具有多个帧缓存的情况下实现切换的功能。

除了 **LCD 通用 APIs** 之外，**RGB 接口驱动** 中还提供了一些特殊功能的函数，下面是一些常用函数的使用说明：

- `esp_lcd_rgb_panel_set_pclk()`：动态修改时钟频率，可以在 LCD 初始化后使用。
- `esp_lcd_rgb_panel_restart()`：复位数据传输，用于在屏幕发生偏移时调用可以使其恢复正常。
- `esp_lcd_rgb_panel_get_frame_buffer()`：获取帧缓存的地址，可用数量由配置参数 `num_fbs` 决定，用于多缓冲防撕裂。

- `esp_lcd_rgb_panel_register_event_callbacks()`：注册多种事件的回调函数，示例代码及说明如下：

```
static bool example_on_vsync_event(esp_lcd_panel_handle_t panel, const esp_lcd_
→rgb_panel_event_data_t *edata, void *user_ctx)
{
    /* 可以在此处进行一些操作 */

    return false;
}

static bool example_on_bounce_event(esp_lcd_panel_handle_t panel, const esp_
→lcd_rgb_panel_event_data_t *edata, void *user_ctx)
{
    /* 可以在此处进行一些操作 */

    return false;
}

esp_lcd_rgb_panel_event_callbacks_t cbs = {
    .on_vsync = example_on_vsync_event,           //↵
    →刷新完一帧图像时的回调函数
    .on_bounce_frame_finish = example_on_bounce_event, // 通过 Bounce Buffer↵
    →机制搬运完一帧图像时的回调函数
                                                    // 需注意，此时 RGB↵
    →接口还未传输完该帧图像
};
ESP_ERROR_CHECK(esp_lcd_rgb_panel_register_event_callbacks(panel_handle, &cbs,↵
→&example_user_ctx));
```

## 相关文档

- [ST7701S 数据手册](#)
- [ST77903 数据手册](#)
- [GC9503 数据手册](#)

## 4.1.6 LCD 屏幕撕裂详解

### 目录

- 术语表
- 屏幕撕裂的原理
  - 现象
  - 原因
- 防止屏幕撕裂的方法
  - 基于多 *GRAM* 的方法
  - 基于 *TE* 信号的方法

该部分旨在介绍 LCD 出现屏幕撕裂的原理以及相应的处理方法。

### 术语表

请参阅[LCD 术语表](#)。

## 屏幕撕裂的原理

**现象** 屏幕撕裂，通常也称为撕裂效应，是 LCD 应用中常见的问题，通常在 GUI 发生全屏或者较大面积的区域变化时出现，现象是在 LCD 上同时显示了几帧图像中的不同部分，使得人眼能够观察到图像出现明显的断层，大大降低了 GUI 的视觉体验。下面为运行 LVGL Musci 示例时出现和没有出现屏幕撕裂的效果图。

点击链接可参考 [出现屏幕撕裂的效果图](#)。

点击链接可参考 [没有出现屏幕撕裂的效果图](#)。

**原因** 接下来会通过设置一系列假设条件并结合示意图来详细说明产生屏幕撕裂的原因。为了方便起见，这里将屏幕的刷新过程简化为 **写入**和 **读取**两个步骤，写入是指主控将渲染得到的色彩数据写入 GRAM 的过程，读取是指屏幕持续从 GRAM 中读取色彩数据并显示到面板上的过程。下面为简化的屏幕刷新示意图。

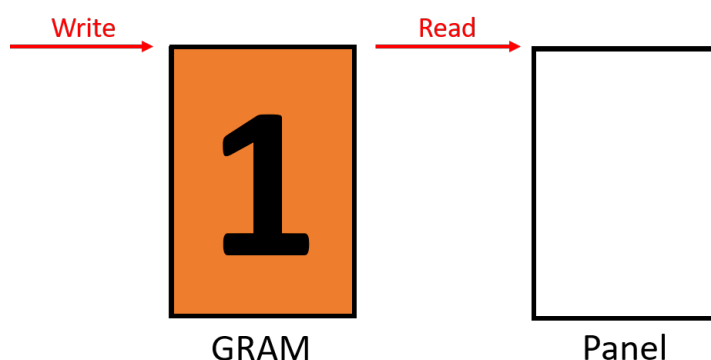


图 31: 简化的屏幕刷新示意图

**备注：**一些 LCD 可以通过命令控制主控写入和屏幕读取的方向，如 **ST7789** 的 36h 命令，当方向不一致时，也有可能出现屏幕撕裂，后续仅针对读取和写入方向一致的情况进行说明。

当主控在写入时没有进行任何同步操作，即写入的初始位置以及初始时刻都是未知的，如果写入和读取的速度不相等，那么就有可能出现屏幕撕裂。

1. 假设写入的速度比读取慢（以速度比值为 1: 2 举例），在写入第二帧图像的过程中，读取的位置会超过写入位置，使得屏幕仅读取到第二帧图像的前半部分，从而导致显示图像出现撕裂，下面为演示过程的示意图。

点击链接可参考 [写入与读取不同步且速度比值为 1: 2 时的示意图](#)。

2. 假设写入的速度比读取快（以速度比值为 2: 1 举例），在屏幕读取第一帧图像的过程中，写入的位置会超过读取的位置，使得屏幕读取到第二帧图像的后半部分，从而导致显示图像出现撕裂，下面为演示过程的示意图。

点击链接可参考 [写入与读取不同步且速度比值为 2: 1 时的示意图](#)。

当主控在写入时采取了同步操作，即写入的初始位置以及初始时刻和读取是同步的，如果写入和读取的速度不匹配，那么也有可能出现屏幕撕裂。

1. 假设写入的速度小于读取的二分之一（以速度比值为 1: 3 举例），在写入第二帧图像的过程中，读取的位置会超过写入的位置，使得屏幕仅读取到第二帧图像的前半部分，从而导致显示图像出现撕裂，下面为演示过程的示意图。

点击链接可参考 [写入与读取同步且速度比值为 1: 3 时的示意图](#)。



2. 假设写入的速度大于或等于读取的二分之一（以速度比值为 1: 2 举例），在写入第二帧图像的过程中，读取的位置不会与写入的位置重叠，使得屏幕能够读取完整的第二帧图像，于是显示图像没有出现撕裂，下面为演示过程的示意图。

点击链接可参考 [写入与读取同步且速度比值为 1: 2 时的示意图](#)。

基于上述假设进行总结，出现屏幕撕裂的主要原因包含以下两点：

1. 写入和读取同时操作同一个 GRAM
2. 写入和读取的初始状态不同步或者速度不匹配

### 防止屏幕撕裂的方法

在了解屏幕撕裂出现的原因之后，可以分别从 **GRAM** 和 **读写状态与速度** 两个角度来实现屏幕的防撕裂方法。由于不同接口类型的 LCD 可能具有不同的 **刷新机制** 和 **GRAM 位置**，需要根据具体的接口类型来选择推荐的防撕裂方法，下表为不同接口类型下 GRAM 的位置以及相应的防撕裂方法。

接口类型	GRAM 位置	防撕裂方法
RGB, MIPI-DSI (video mode), QSPI (without internal GRAM)	主控	<a href="#">基于多 GRAM 的方法</a>
SPI, I80, QSPI (with internal GRAM)	LCD	<a href="#">基于 TE 信号的方法</a>

**基于多 GRAM 的方法** 这种方法适用于 GRAM 在写入主控内的情况，并且要求主控可以自由调整屏幕读取的目标 GRAM，工作原理是：通过增加额外的 GRAM 来避免写入和读取同时操作同一个 GRAM。下面介绍了基于双 GRAM 的防撕裂方法，演示过程的示意图如下。

点击链接可参考 [基于双 GRAM 实现防撕裂的示意图](#)。

从图中可以看出，初始时写入主控准备将第二帧图像写入 GRAM2，而屏幕准备读取 GRAM1 中的第一帧图像。当写入完成后，首先需要设置屏幕的下一帧从 GRAM2 读取，然后等待屏幕读取完当前帧图像。当屏幕读取完成后，接着就开始读取 GRAM2 中的第二帧图像，同时写入主控也开始将第三帧图像写入 GRAM1。因此，写入和读取不会同时操作同一个 GRAM，从而避免了屏幕出现撕裂。

下面是基于 LVGL 实现的相关示例代码：

1. [rgb\\_avoid\\_tearing](#)
2. [qspi\\_without\\_ram](#)

**备注：**为了优化显示性能，还可以在使用两个 GRAM 的基础上再新增一个 GRAM，此时，写入主控在完成一帧的写入后无需等待屏幕读取完一帧，而是直接开始写入下一帧。关于如何实现三个 GRAM 的防撕裂方法，请参阅[示例代码](#)。

**基于 TE 信号的方法** 这种方法适用于 GRAM 在 LCD 内的情况，并且要求 LCD 提供对外的 TE 信号引脚，工作原理是：通过 TE 信号控制写入的初始状态，使其与读取保持同步，同时控制写入的速度不小于读取的二分之一，从而避免写入和读取在 GRAM 的中间位置发生重合。下面介绍了基于 TE 信号的防撕裂方法，演示过程的示意图如下。

点击链接可参考 [基于 TE 信号实现防撕裂的示意图](#)。

从图中可以看出，初始时写入主控正在等待 TE 信号，而屏幕准备进入消影区域（Porch）。当屏幕开始读取 GRAM 中的第一帧图像时，会向写入主控发送 TE 信号。当写入主控接收到 TE 信号后，就开始向 GRAM 写入第二帧图像，并且保证写入和读取的速度比值为 2: 3。因此，写入和读取不会在 GRAM 的中间位置发生重合，从而避免了屏幕出现撕裂。

下面是基于 LVGL 实现的相关示例代码：

1. [lcd\\_with\\_te](#)

**备注:**

1. 一些 LCD 可以通过命令控制 TE 信号的开关以及触发时机等参数，如 **ST7789** 的 35h 和 44h 命令，为了保证上述方法的有效性，用户需要根据具体 LCD 驱动 IC 的数据手册来设置相应的参数，使得 TE 信号开启并在合适的位置触发。
2. 一些 LCD 可以通过命令控制主控写入和屏幕读取的方向，如 **ST7789** 的 36h 命令，当方向不一致时，上述防止屏幕撕裂的方法会失效，用户需要根据具体 LCD 驱动 IC 的数据手册来设置相应的参数，使得写入和读取的方向一致。

## 4.1.7 LCD Application Solution

### LCD 方案介绍

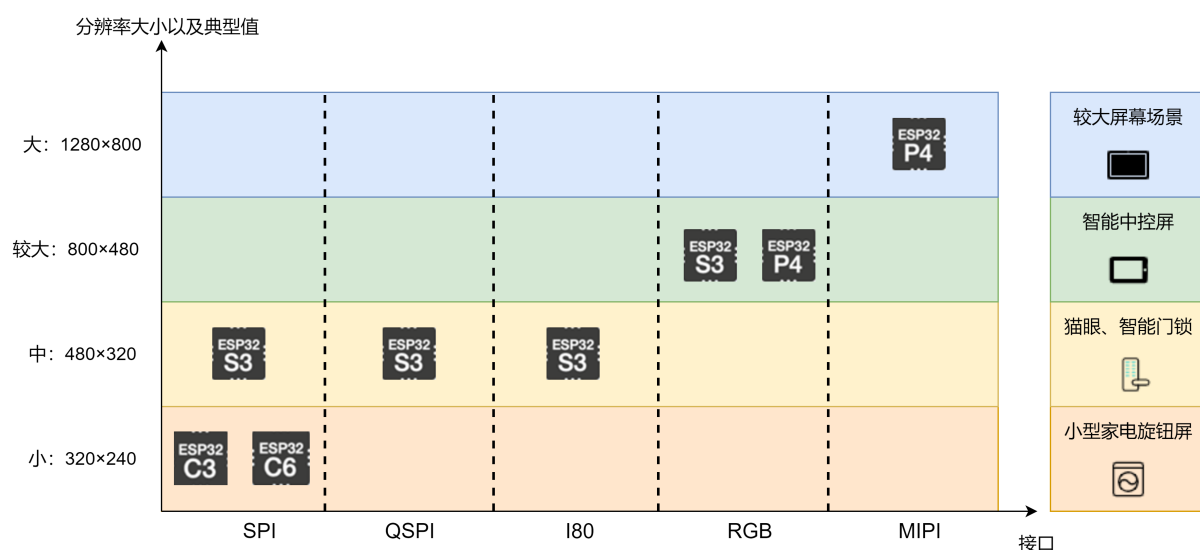
乐鑫 HMI 智能屏 (LCD) 方案具有卓越性能和可扩展性，可与不同 ESP 主控芯片搭配。该方案在智能家居控制、家电屏幕、医疗设备、工业控制和儿童教育等多个应用场景下表现出色。优势包括高性能图形可视化，低内存占用等。此外，屏幕适配方案完善，并支持高性能 JPEG 解码和帧率优化。以下是 LCD 方案具体介绍：

- **出色的图形可视化：**利用 ESP 芯片的高性能图形处理能力，并深度合作于 LVGL 官方，使其在 LVGL 兼容性方面表现优异。该方案提供精美的视觉效果，低内存占用，并可轻松移植到产品设计中。
- **简单的 UI 设计：**UI 编辑器 Squareline Studio 支持快速轻松地设计和开发嵌入式设备的 UI。移植简单，无需代码即可实现 UI，最大限度地减少开发时间。
- **丰富的软硬件参考：**提供全面的 LCD 软硬件开发资料，包括详尽的指导文档和示例。此外，特定为各种 HMI 应用场景设计的 HMI 开发板可帮助开发者快速上手。
- **完善的屏幕适配：**支持多种操作方式，包括触摸、旋钮等。支持多种外设接口，如 RGB、SPI 等。已适配了多款 LCD 驱动 IC 和 Touch 驱动 IC 并已经整理成组件，满足不同用户群体的需求。

此外，乐鑫还支持以下功能来进一步保证在 LCD 应用场景下有更丝滑的交互体验：

- **帧率优化和防撕裂技术：**通过精心优化的帧率控制和防撕裂技术，确保图像显示的流畅性和一致性。
- **高性能 JPEG 解码：**支持高效的图像处理，确保流畅的多媒体体验。
- **语音唤醒和识别：**可集成先进的语音识别技术，为用户提供更便捷的交互方式。

以下是一图流方案综述：





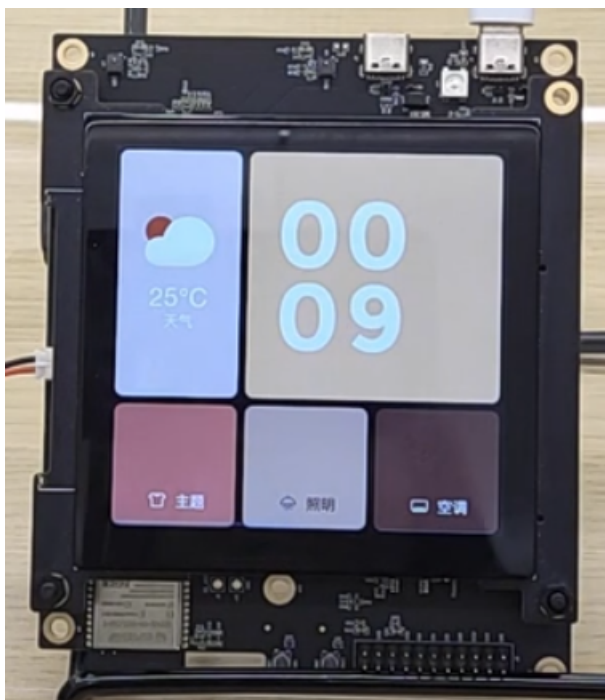
## LCD 常见应用场景

乐鑫 LCD 方案广泛应用于各个领域，包括但不限于：

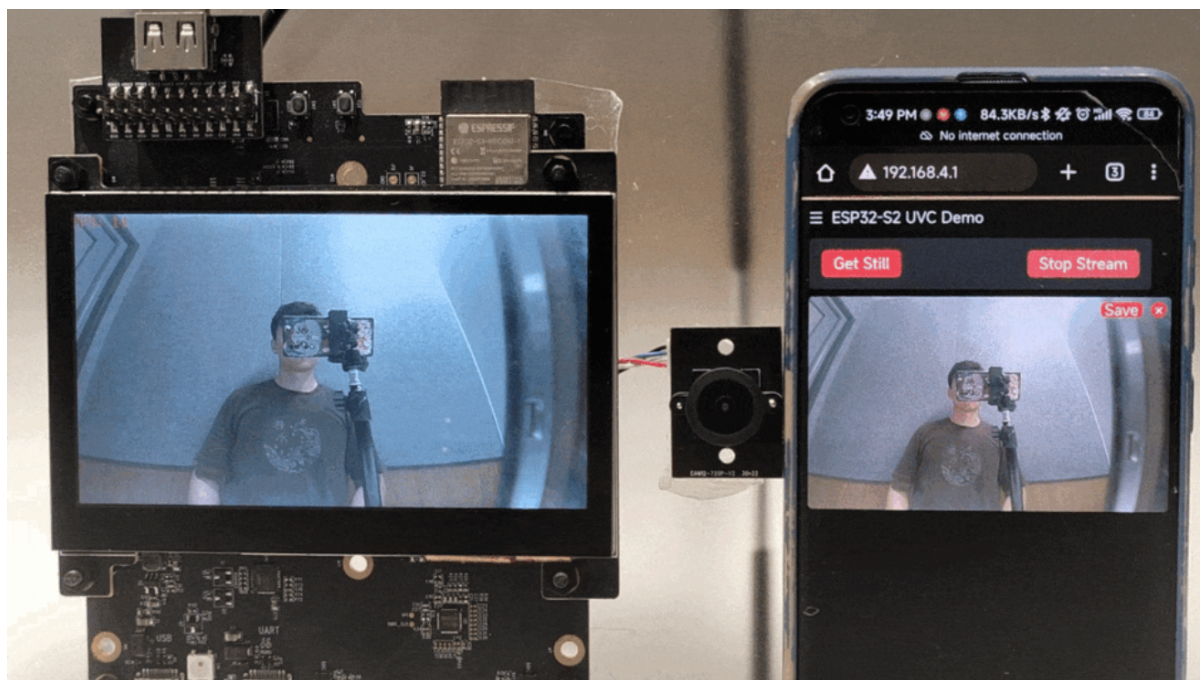
- **旋钮屏方案：**针对智能家电产品，传统段码屏和黑白屏升级首选。支持 Wi-Fi、蓝牙，扩展接口可实现串口通讯等功能。典型应用场景为小型家电应用中的旋钮屏。



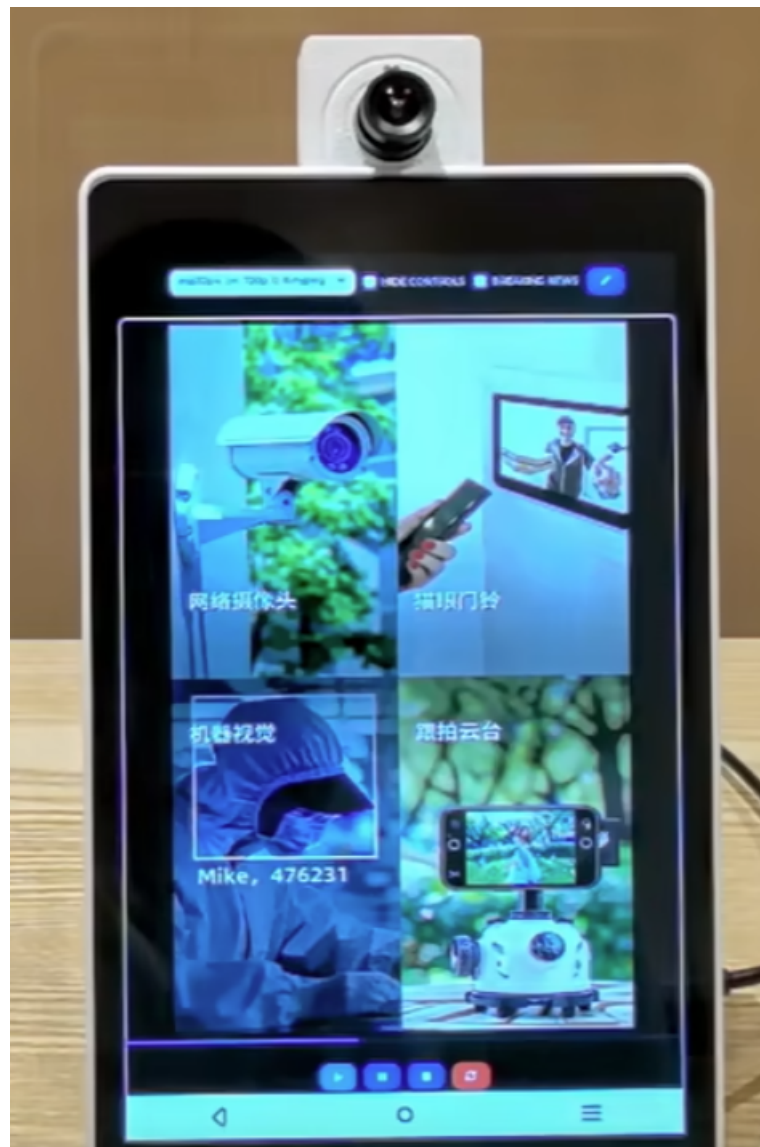
- **中控屏方案：**集成 Wi-Fi、BLE、离线语音、RGB LCD 显示，支持离线命令词和连续语音识别。适用于传统 86 面板的升级迭代，构建了集设备控制、开关面板、温控面板、智能遥控器为一体的智能家居控制中枢。



- **可视语音方案：**使用原生 USB 对接通用 USB 摄像头，在单颗 SoC 上同时实现摄像头数据流读取、JPEG 解码和 RGB 接口屏实时显示，无需增加额外的 USB 芯片。适用于猫眼、智能门铃门锁、电子内窥镜等场景。



- **高性能多媒体方案：**该方案采用 ESP32-P4 芯片，支持 MIPI-CSI 和 MIPI-DSI 接口，适用于各种需要高分辨率摄像头和显示的场景。该芯片还集成了多种媒体编码和压缩协议的硬件加速器、硬件像素处理加速器（PPA）和 2D-DMA，适合各种多媒体场景。



总结如下:

表 1: LCD 方案概览

方案类别	方案名称	主控	屏幕	功能	应用场景
旋钮屏	2.1 英寸旋钮屏方案 (启明)	ESP32-S3	2.1 英寸、480 x 480 RGB 接口屏	支持 Wi-Fi、蓝牙, 扩展接口可实现串口通讯、按键、USB 摄像头等功能	智能家电产品, 传统段码屏和黑白屏升级首选
旋钮屏	1.28 英寸旋钮屏方案	ESP32-C3	1.28 英寸、240 x 240 SPI 接口屏	带按压开关的旋转编码器, 小封装紧凑设计	小型家电应用中的旋钮屏和小尺寸显示屏的应用场景
中控屏	智能语音触控面板 (86 盒) 方案	ESP32-S3	3.95 英寸、480 x 480 RGB 接口屏	集成 Wi-Fi、BLE、离线语音、RGB LCD 显示, 支持离线命令词和连续语音识别	传统 86 面板的升级迭代, 构建了集设备控制、开关面板、温控面板、智能遥控器等为一体的智能家居控制中枢
中控屏	智能中控开关方案	ESP32-S3	7 英寸、800 x 480 RGB 接口屏	多点触摸屏实现手势动作识别, 支持 Wi-Fi CSI 人体接近感应, 用于家庭可视面板的搭建	智能家庭中的快捷开关控制, 如场景模式切换和灯开关
可视语音方案	可视语音方案	ESP32-S3	4.3 英寸、800 x 480 RGB 接口屏	使用原生 USB 对接通用 USB 摄像头, 在单颗 SoC 上同时实现摄像头数据流读取、JPEG 解码和 RGB 接口屏实时显示, 无需增加额外的 USB 芯片, 本地视频解码及屏幕刷新分辨率可达 800 x 480 @15 FPS	猫眼, 智能门铃门锁, 电子内窥镜等使用场景
高性能多媒体方案	高性能多媒体方案	ESP32-P4	8 英寸、800 x 1280 MIPI-DSI 接口屏	支持 MIPI-CSI 和 MIPI-DSI 接口, 适用于各种需要高分辨率摄像头和显示的场景, 应用多种媒体编码和压缩协议的硬件加速器、硬件像素处理加速器 (PPA) 和 2D-DMA, 适合各种多媒体场景	高性能多媒体场景

## LCD 参考方案

### ESP-BOX

#### 描述:

语音助手、触摸屏控制器、传感器、红外控制器和智能 Wi-Fi 网关开发的家电控制平台。

#### 硬件:

- 开发板: [ESP32-S3-BOX-3](#)

#### 相关链接:

- 代码仓库: [esp-box](#)
- 相关视频: [ESP32-S3-BOX-3 惊喜开箱!](#)

**特性:**

- 基于 LVGL GUI 框架
- 双 mic 远场语音交互, 中英文 AI 离线语识别, 可支持 200 多条语音命令
- 集成端到端 AIoT 开发框架 ESP-RainMaker
- Pmod™ 兼容接头支持外围模块, 可拓展传感器、红外控制器等
- PSRAM 要求 8 线 (8M)

**ESP32-C3 旋钮屏****描述:**

圆形旋钮屏方案, 集成洗衣机、调光器、温控器等常用场景

**硬件:**

- 开发板: [ESP32-C3-LCDkit](#)

**相关链接:**

- 代码仓库: [esp32-c3-lcdkit](#)
- 相关视频:
  - [ESP32-C3 旋钮屏 Demo](#)
  - [ESP32-C3-LCDKit 旋钮屏开发板](#)

**特性:**

- 基于 LVGL GUI 框架
- 圆屏 UI 显示 (非触摸), 旋转编码器控制

**智能语音触控面板 (86 盒)****描述:**

可用于传统 86 面板的升级迭代, 构建了集设备控制、开关面板、温控面板、智能遥控器等为一体的智能家居控制中枢

**硬件:**

- 开发板: [ESP32-S3-LCD-EV-Board](#)
- 屏幕: LCD 子板 2 (480x480)

**相关链接:**

- 代码仓库: [esp32-s3-lcd-ev-board/86-box Smart Panel Example](#)
- 相关视频: [ESP32-S3 智能语音触控面板](#)

**特性:**

- 基于 LVGL GUI 框架
- 双 mic 远场语音交互，中英文 AI 离线语音识别，可支持 200 多条语音命令
- PSRAM 要求 8 线 (R8)，并开启 120M

### 电子可视门铃

#### 描述:

使用原生 USB 对接通用 USB 摄像头，在单颗 SoC 上同时实现摄像头数据流读取、JPEG 解码和 RGB 接口屏实时显示，无需增加额外的 USB 芯片，本地视频解码及屏幕刷新分辨率可达 800x480@15 FPS

#### 硬件:

- 开发板: [ESP32-S3-LCD-EV-Board](#)
- 屏幕: LCD 子板 3 (800x480)

#### 相关链接:

- 代码仓库: [esp32-s3-lcd-ev-board/USB Camera LCD Example](#)
- 相关视频: [ESP32-S3 驱动 RGB 接口屏 + USB CDC 摄像头 Demo](#)

#### 特性:

- USB 摄像头数据流读取，需要支持 Bulk 模式
- JPEG 解码
- 800x480 RGB LCD 显示
- PSRAM 要求 8 线 (R8)，并开启 120M

### 智能中控开关

#### 描述:

通过多点触摸屏实现双指叩击、拍一拍等手势动作识别，可用于智能家庭中的快捷开关控制，如场景模式切换和灯开关。结合 Wi-Fi CSI 人体接近感应功能，还可以实现屏幕接近亮屏和远离息屏的自动开关控制

#### 硬件:

- 开发板: [ESP32-S3-LCD-EV-Board](#)
- 屏幕: 7 英寸、RGB 接口、800x480 分辨率

#### 相关链接:

- 相关视频: [ESP32-S3 驱动超大 RGB 接口屏](#)

#### 特性:

- 7 英寸超大 LCD 屏幕，支持多点触摸
- Wi-Fi CSI 人体接近感应
- PSRAM 要求 8 线 (R8)，并开启 120M



## 高性能多媒体方案

### 描述:

支持 MIPI-CSI 和 MIPI-DSI 接口, 适用于各种需要高分辨率摄像头和显示的场景, 应用多种媒体编码和压缩协议的硬件加速器、硬件像素处理加速器 (PPA) 和 2D-DMA, 适合各种多媒体场景

### 硬件:

- 开发板: ESP32-P4\_Function\_EV\_Board
- 屏幕: 8 英寸 800 x 1280 液晶屏 (IC: ILI9881C )

### 相关链接:

- 相关视频: [挑战用 ESP32-P4 做一部智能手机](#)

### 特性:

- 支持 MIPI-DSI 和 MIPI-CSI 接口
- 多种媒体编码和压缩协议的硬件加速器
- 硬件像素处理加速器 (PPA) 和 2D-DMA

## LCD 参考资料

- LCD 软件参考
  - [ESP LCD 驱动库](#)
  - [Arduino LCD 驱动库](#)
  - [ESP LCD 驱动文档](#)
  - [ESP LCD 例程](#)
  - [ESP-BOX AIoT 开发框架](#)
- LCD 方案 & 开发指南
  - [ESP-HMI 智能屏方案](#)
  - [快速入门 GUI \(上\)](#)
  - [快速入门 GUI \(下\)](#)
  - [ESP LCD 开发指南](#)
- LCD 相关开发板购买
  - [ESP32-S3-LCD-EV-Board](#): 目前支持 800 x 480 4.3 寸 (RGB) 和 480 x 480 3.95 寸 (RGB) 两种子板, 支持电容触屏。[购买链接](#)
  - [ESP32-S3-BOX](#): 240 x 320 2.4 寸 (SPI) ILI9342, 支持电容触屏。[购买链接](#)
  - [ESP32-S3-BOX-Lite](#): 240 x 320 2.4 寸 (SPI) ST7789V, 不支持触屏, 但是屏上有 3 个按键。[购买链接](#)
  - [ESP32-C3-LCDkit](#): 240 x 240 1.28 寸 (SPI) GC9A01, 不支持触屏。[购买链接](#)
- 模组/开发板资料及选项参考
  - [ESP32-S3 技术规格书](#)
  - [ESP32-S3-WROOM-1](#)
  - [ESP32-C3 技术规格书](#)
  - [ESP32-C3-MINI-1](#)
  - [乐鑫产品选型工具](#)

## 4.2 数码管驱动

数码管/LED 点阵是嵌入式系统中常见的显示方案，该方案比 LCD 显示屏占用更少的引脚和内存资源，实现也更加简单，比较适合计时、计数、状态显示等具有单一显示需求的应用场景。

ESP-IoT-Solution 已经适配的数码管/LED 显示驱动器如下：

名称	功能	接口	驱动	数据手册
CH450	数码管显示驱动芯片，支持 6 位数码管	I2C	ch450	CH450
HT16C21	20x4/16x8 LCD 控制器，支持 RAM 映射	I2C	ht16c21	HT16C21
IS31FL3XXX	LED 点阵控制器	I2C	is31fl3xxx	IS31FL3XXX

### 4.2.1 CH450 驱动

CH450 是一款数码管显示驱动芯片，可以用于驱动 6 位数码管或 48 点 LED 矩阵，可通过 I2C 接口与 ESP32 进行通信。

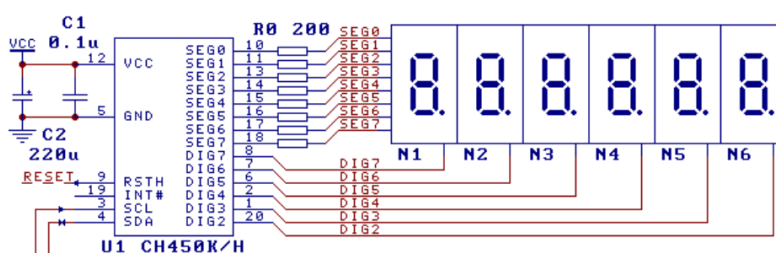


图 32: CH450 典型应用电路图

该驱动对 CH450 的基本操作进行了封装，用户可以直接调用 `ch450_write()` 或 `ch450_write_num()` 接口在数码管上进行数字显示。

#### 示例

```
i2c_bus_handle_t i2c_bus = NULL;
ch450_handle_t seg = NULL;
i2c_config_t conf = {
    .mode = I2C_MODE_MASTER,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .master.clk_speed = I2C_MASTER_FREQ_HZ,
};
i2c_bus = i2c_bus_create(I2C_MASTER_NUM, &conf);
seg = ch450_create(i2c_bus);

for (size_t i = 0; i < 10; i++) {
    for (size_t index = 0; index < 6; index++) {
        ch450_write_num(seg, index, i);
    }
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}

ch450_delete(seg);
i2c_bus_delete(&i2c_bus);
```



## 4.2.2 HT16C21 驱动

HT16C21 是一款支持 RAM 映射的 LCD 控制/驱动芯片，可用于驱动  $20 \times 4$  或  $16 \times 8$  段码式液晶屏，该芯片通过 I2C 接口与 ESP32 进行通信。

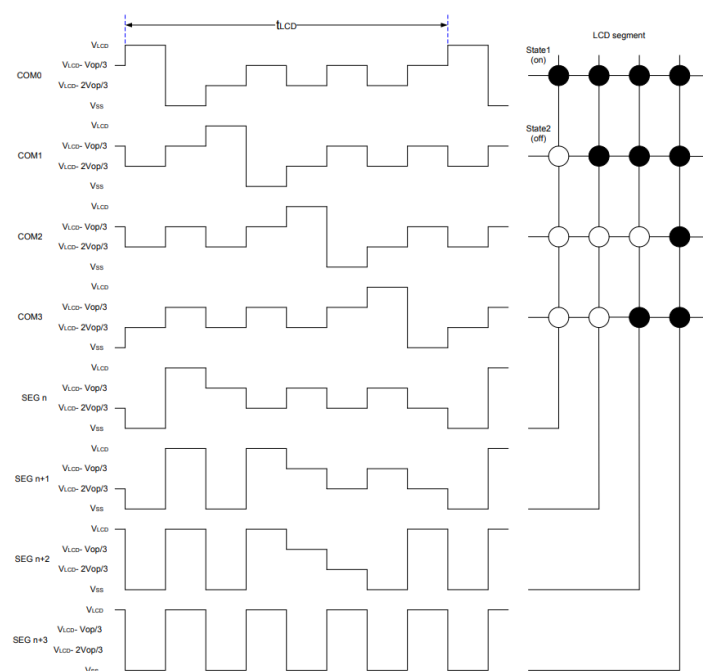


图 33: HT16C21 典型驱动模型

该驱动对 HT16C21 的基本操作进行了封装，用户使用 `ht16c21_create` 创建实例之后，通过 `ht16c21_param_config` 对驱动器参数进行配置，之后即可直接调用 `ht16c21_ram_write` 进行写入操作。

### 示例

```
i2c_bus_handle_t i2c_bus = NULL;
ht16c21_handle_t seg = NULL;
uint8_t lcd_data[8] = { 0x10, 0x20, 0x30, 0x50, 0x60, 0x70, 0x80 };

i2c_config_t conf = {
    .mode = I2C_MODE_MASTER,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .master.clk_speed = I2C_MASTER_FREQ_HZ,
};
i2c_bus = i2c_bus_create(I2C_MASTER_NUM, &conf);
seg = ht16c21_create(i2c_bus, HT16C21_I2C_ADDRESS_DEFAULT);

ht16c21_config_t ht16c21_conf = {
    .duty_bias = HT16C21_4DUTY_3BIAS;
    .oscillator_display = HT16C21_OSCILLATOR_ON_DISPLAY_ON;
    .frame_frequency = HT16C21_FRAME_160HZ;
    .blinking_frequency = HT16C21_BLINKING_OFF;
    .pin_and_voltage = HT16C21_VLCD_PIN_VOL_ADJ_ON;
    .adjustment_voltage = 0;
};
```

(下页继续)

(续上页)

```

ht16c21_param_config(seg, &ht16c21_conf);
ht16c21_ram_write(seg, 0x00, lcd_data, 8);

ht16c21_delete(seg);
i2c_bus_delete(&i2c_bus);

```

### 4.2.3 IS31FL3XXX 驱动

IS31FL3XXX 系列芯片可用于驱动不同规模的 LED 点阵屏幕。其中 IS31FL3218 支持 18 个恒流通道，每个通道由独立的 PWM 控制，最大输出电流 38 mA，可直接驱动 LED 进行显示。IS31FL3736 支持更多的通道，最大可组成 12 × 8 LED 矩阵，每个通道由一个 8 位 PWM 驱动，最大支持 256 级渐变。

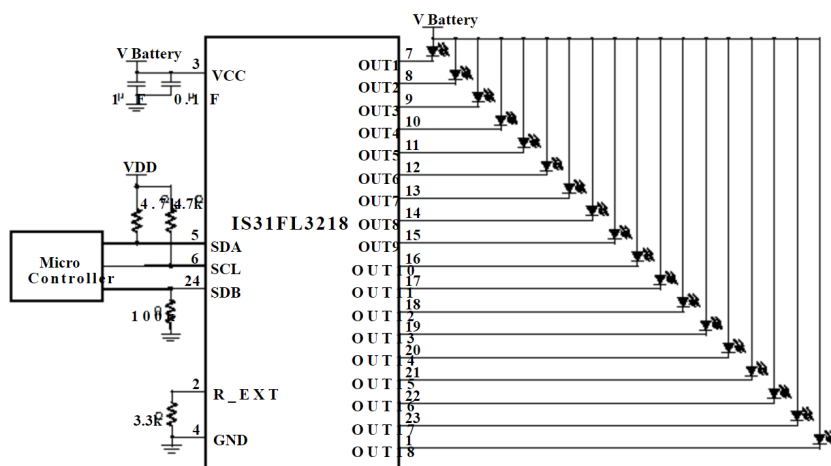


图 34: IS31FL3218 典型应用电路图

该驱动对 IS31FL3XXX 的基本操作进行了封装，示例如下节所示。

#### IS31FL3218 示例

```

i2c_bus_handle_t i2c_bus = NULL;
is31fl3218_handle_t fxled = NULL;
i2c_config_t conf = {
    .mode = I2C_MODE_MASTER,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .master.clk_speed = I2C_MASTER_FREQ_HZ,
};
i2c_bus = i2c_bus_create(I2C_MASTER_NUM, &conf);
fxled = is31fl3218_create(i2c_bus);
is31fl3218_channel_set(fxled, 0x00ff, 128); // set PWM 1 ~ PWM 8 duty cycle 50%
is31fl3218_delete(fxled);
i2c_bus_delete(&i2c_bus);

```

## 4.3 LED 指示灯

本指南包含以下内容：

## 目录

- 支持的指示灯类型
- 定义闪烁类型
  - 控制亮灭
  - 控制亮度
  - 控制颜色
  - 控制索引
- 预定义闪烁优先级
- 控制指示灯闪烁
- 自定义指示灯闪烁
- *gamma* 曲线调光
- 驱动电平设置
- API 参考
  - *Header File*
  - *Functions*
  - *Structures*
  - *Type Definitions*
  - *Enumerations*

LED 指示灯是最简单的输出外设之一，可以通过不同形式的闪烁指示系统当前的工作状态。ESP-IoT-Solution 提供的 LED 指示灯组件具有以下功能：

- 支持定义多组闪烁类型
- 支持定义闪烁类型优先级
- 支持创建多个指示灯
- LEDC 等驱动支持调节亮度，渐变，颜色等

### 4.3.1 支持的指示灯类型

驱动类型	说明	亮灭	亮度	呼吸	颜色	颜色渐变	索引
GPIO	通过 GPIO 控制指示灯	√	×	×	×	×	×
LEDC	通过一路 PWM 方式控制指示灯	√	√	√	×	×	×
RGB LED	通过三路 PWM 方式控制指示灯	√	√	√	√	√	×
LED Strips	通过 RMT/SPI 控制的灯条，如 WS2812	√	√	√	√	√	√

### 4.3.2 定义闪烁类型

#### 控制亮灭

闪烁步骤结构体 *blink\_step\_t* 定义了该步骤的类型、指示灯状态和状态持续时间。多个步骤组合成一个闪烁类型，不同的闪烁类型可用于标识不同的系统状态。闪烁类型的定义方法如下：

例 1. 定义一个循环闪烁：亮 0.05 s，灭 0.1 s，开始之后一直循环。

```
const blink_step_t test_blink_loop[] = {
    {LED_BLINK_HOLD, LED_STATE_ON, 50},           // step1: turn on LED 50 ms
    {LED_BLINK_HOLD, LED_STATE_OFF, 100},         // step2: turn off LED 100 ms
    {LED_BLINK_LOOP, 0, 0},                       // step3: loop from step1
};
```

例 2. 定义一个循环闪烁：亮 0.05 s，灭 0.1 s，亮 0.15 s，灭 0.1 s，执行完毕灯熄灭。

```
const blink_step_t test_blink_one_time[] = {
    {LED_BLINK_HOLD, LED_STATE_ON, 50},           // step1: turn on LED 50 ms
    {LED_BLINK_HOLD, LED_STATE_OFF, 100},         // step2: turn off LED 100 ms
    {LED_BLINK_HOLD, LED_STATE_ON, 150},          // step3: turn on LED 150 ms
    {LED_BLINK_HOLD, LED_STATE_OFF, 100},         // step4: turn off LED 100 ms
    {LED_BLINK_STOP, 0, 0},                       // step5: stop blink (off)
};
```

定义闪烁类型之后，需要在 `led_indicator_blink_type_t` 添加该类型对应的枚举成员，然后将其添加到闪烁类型列表 `led_indicator_blink_lists`，示例如下：

```
typedef enum {
    BLINK_TEST_BLINK_ONE_TIME, /**< test_blink_one_time */
    BLINK_TEST_BLINK_LOOP,     /**< test_blink_loop */
    BLINK_MAX,                 /**< INVALID type */
} led_indicator_blink_type_t;

blink_step_t const * led_indicator_blink_lists[] = {
    [BLINK_TEST_BLINK_ONE_TIME] = test_blink_one_time,
    [BLINK_TEST_BLINK_LOOP] = test_blink_loop,
    [BLINK_MAX] = NULL,
};
```

## 控制亮度

对于支持控制亮度的驱动，可以通过以下方式控制指示灯的亮度：

例 1. 定义一个亮度设置：设置指示灯亮度为 50%，持续 0.5 s。

```
const blink_step_t test_blink_50_brightness[] = {
    {LED_BLINK_BRIGHTNESS, LED_STATE_50_PERCENT, 500}, // step1: set to half_
    ↪brightness 500 ms
    {LED_BLINK_STOP, 0, 0},                             // step4: stop blink (50%_
    ↪brightness)
};
```

例 2. 定义一个循环闪烁：渐亮 0.5s，逐灭 0.5s，重复执行。

```
const blink_step_t test_blink_breathe[] = {
    {LED_BLINK_HOLD, LED_STATE_OFF, 0},           // step1: set LED off
    {LED_BLINK_BREATHE, LED_STATE_ON, 500},       // step2: fade from off_
    ↪to on 500ms
    {LED_BLINK_BREATHE, LED_STATE_OFF, 500},      // step3: fade from on to_
    ↪off 500ms
    {LED_BLINK_LOOP, 0, 0},                       // step4: loop from step1
};
```

例 3. 定义一个闪烁：从 50% 亮度渐亮到 100% 亮度，持续 0.5s。

```
const blink_step_t test_blink_breathe_2[] = {
    {LED_BLINK_BRIGHTNESS, LED_STATE_50_PERCENT, 0}, // step1: set to half_
    ↪brightness 0 ms
    {LED_BLINK_BREATHE, LED_STATE_ON, 500},       // step2: fade from off_
    ↪to on 500ms
    {LED_BLINK_STOP, 0, 0},                       // step3: stop blink (100
    ↪% brightness)
};
```

## 控制颜色

对于支持控制颜色的驱动，我们可以通过 `LED_BLINK_RGB`, `LED_BLINK_RGB_RING`, `LED_BLINK_HSV`, `LED_BLINK_HSV_RING` 来控制颜色。

- `LED_BLINK_RGB`: 通过 RGB 控制颜色，其中 R 占 8 bites (0-255), G 占 8 bites (0-255), B 占 8 bites (0-255)。
- `LED_BLINK_RGB_RING`: 通过 RGB 控制颜色渐变，会从上一次的颜色按照色环渐变到当前的设置颜色。采用 RGB 值插值法。
- `LED_BLINK_HSV`: 通过 HSV 控制颜色，其中 H 占 9 bites (0-360), S 占 8 bites (0-255), V 占 8 bites (0-255)。
- `LED_BLINK_HSV_RING`: 通过 HSV 控制颜色渐变，会从上一次的颜色按照色环渐变到当前的设置颜色。采用 HSV 值插值法。

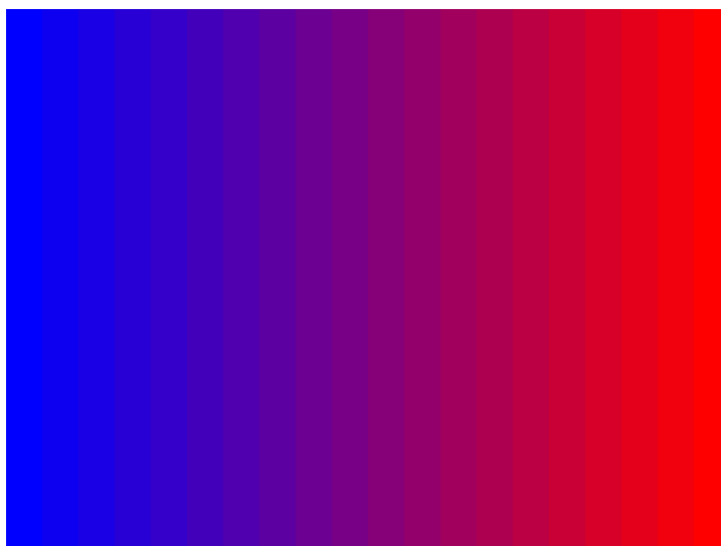
例 1. 定义一个颜色设置，让指示灯显示红色。

```
const blink_step_t test_blink_rgb_red[] = {
    {LED_BLINK_RGB, SET_RGB(255,0,0), 0},           // step1: set to half_
    ↪brightness 500 ms
    {LED_BLINK_STOP, 0, 0},                         // step2: stop blink (red_
    ↪color)
};
```

例 2. 定义一个颜色渐变，让指示灯从红色渐变到蓝色，并循环执行。

```
const blink_step_t test_blink_rgb_red_blue[] = {
    {LED_BLINK_RGB, SET_RGB(0xFF, 0, 0), 0},       // step1: set to red_
    ↪color 0 ms
    {LED_BLINK_RGB_RING, SET_RGB(0, 0, 0xFF), 4000}, // step2: fade from red_
    ↪to blue 4000ms
    {LED_BLINK_RGB_RING, SET_RGB(0xFF, 0, 0), 4000}, // step3: fade from blue_
    ↪to red 4000ms
    {LED_BLINK_LOOP, 0, 0},                        // step4: loop from step1
};
```

采用 RGB 插值法显示颜色渐变，效果如下。



同时，驱动还支持通过 HSV 颜色来设置，使用方法与 RGB 类似。

例 3. 定义一个颜色设置，让指示灯显示红色 0.5s, 绿色 0.5s, 蓝色 0.5s, 最后停止

```
const blink_step_t test_blink_rgb_red[] = {
    {LED_BLINK_HSV, SET_HSV(0,255,255), 500},     // step1: set color_
    ↪to red 500 ms
```

(下页继续)

(续上页)

```

    {LED_BLINK_HSV, SET_HSV(120,255,255), 500},           // step2: set color
    ↪to green 500 ms
    {LED_BLINK_HSV, SET_HSV(240,255,255), 500},           // step3: set color
    ↪to blue 500 ms
    {LED_BLINK_STOP, 0, 0},                                // step4: stop blink
    ↪(blue color)
};

```

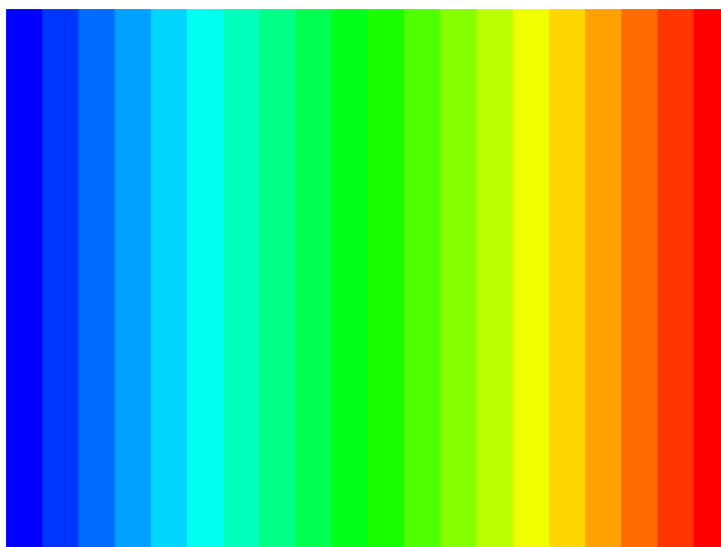
例 4. 定义一个颜色渐变，让指示灯从红色渐变到蓝色，并循环执行。

```

const blink_step_t test_blink_hsv_red_blue[] = {
    {LED_BLINK_HSV, SET_HSV(0,255,255), 0},               // step1: set to red
    ↪color 0 ms
    {LED_BLINK_HSV_RING, SET_HSV(240,255,255), 4000},      // step2: fade from red
    ↪to blue 4000ms
    {LED_BLINK_HSV_RING, SET_HSV(0,255,255), 4000},        // step3: fade from blue
    ↪to red 4000ms
    {LED_BLINK_LOOP, 0, 0},                                // step4: loop from step1
};

```

采用 HSV 插值法显示颜色渐变，效果如下。这种方式渐变色彩更加丰富。



## 控制索引

对于支持索引的驱动，我们还可以通过索引来控制灯条上的每个灯的状态。索引的值是通过宏 *IN-SERT\_INDEX*, *SET\_IHSV*, *SET\_IRGB* 来设置。当设置为 *MAX\_INDEX:127* 时，表示设置所有的灯。

例 1. 定义一个颜色类型，让位于 0 号位的灯显示红色，位于 1 号位的灯显示绿色，位于 2 号位的灯显示蓝色，最后退出。

```

const blink_step_t test_blink_index_setting1[] = {
    {LED_BLINK_RGB, SET_IRGB(0,255,0,0), 0},               // step1: set
    ↪index 0 to red color 0 ms
    {LED_BLINK_RGB, SET_IRGB(1,0,255,0), 0},               // step2: set
    ↪index 1 to green color 0 ms
    {LED_BLINK_RGB, SET_IRGB(2,0,0,255), 0},               // step3: set
    ↪index 2 to blue color 0 ms
    {LED_BLINK_LOOP, 0, 0},                                // step4: loop
    ↪from step1
};

```

例 2. 定义一个颜色类型，让所有的灯呼吸，并一直循环。

```
const blink_step_t test_blink_all_breath[] = {
    {LED_BLINK_BRIGHTNESS, INSERT_INDEX(MAX_INDEX, LED_STATE_OFF), 0},
    ↪ // step1: set all leds to off 0 ms
    {LED_BLINK_BREATHE, INSERT_INDEX(MAX_INDEX, LED_STATE_ON), 1000},
    ↪ // step2: set all leds fade to on 1000 ms
    {LED_BLINK_BREATHE, INSERT_INDEX(MAX_INDEX, LED_STATE_OFF), 1000},
    ↪ // step3: set all leds fade to off 1000 ms
    {LED_BLINK_LOOP, 0, 0},
    ↪ // step4: loop from step1
};
```

### 4.3.3 预定义闪烁优先级

对于同一个指示灯，高优先级闪烁可以打断正在进行的低优先级闪烁，当高优先级闪烁结束，低优先级闪烁恢复执行。可以通过调整闪烁类型 `led_indicator_blink_type_t` 枚举成员的顺序调整闪烁的优先级，数值越小的成员执行优先级越高。

例如，在以下示例中闪烁 `test_blink_one_time` 比 `test_blink_loop` 优先级高，可优先闪烁：

```
typedef enum {
    BLINK_TEST_BLINK_ONE_TIME, /**< test_blink_one_time */
    BLINK_TEST_BLINK_LOOP,    /**< test_blink_loop */
    BLINK_MAX,                /**< INVALID type */
} led_indicator_blink_type_t;
```

### 4.3.4 控制指示灯闪烁

创建一个指示灯：指定一个 IO 和一组配置信息创建一个指示灯

```
led_indicator_config_t config = {
    .mode = LED_GPIO_MODE,
    .led_gpio_config = {
        .active_level = 1,
        .gpio_num = 1,
    },
    .blink_lists = led_indicator_get_sample_lists(),
    .blink_list_num = led_indicator_get_sample_lists_num(),
};
led_indicator_handle_t led_handle = led_indicator_create(8, &config); // attach to ↪
↪ gpio 8
```

开始/停止闪烁：控制指示灯开启/停止指定闪烁类型，函数调用后立刻返回，内部由定时器控制闪烁流程。同一个指示灯可以开启多种闪烁类型，将根据闪烁类型优先级依次执行。

```
led_indicator_start(led_handle, BLINK_TEST_BLINK_LOOP); // call to start, the ↪
↪ function not block

/*
 *.....
 */

led_indicator_stop(led_handle, BLINK_TEST_BLINK_LOOP); // call stop
```

删除指示灯：您也可以在不需要进一步操作时，删除指示灯以释放资源

```
led_indicator_delete(&led_handle);
```

抢占操作：您可以在任何时候直接闪烁指定的类型。

```
led_indicator_preempt_start(led_handle, BLINK_TEST_BLINK_LOOP);
```

停止抢占：您可以使用停止抢占函数，来取消正在抢占的闪烁模式。

```
led_indicator_preempt_stop(led_handle, BLINK_TEST_BLINK_LOOP);
```

**备注：** 该组件支持线程安全操作，您可使用全局变量共享 LED 指示灯的操作句柄 `led_indicator_handle_t`，也可以使用 `led_indicator_get_handle` 在其它线程通过 LED 的 IO 号获取句柄以进行操作。

### 4.3.5 自定义指示灯闪烁

```
static blink_step_t const *led_blink_lst[] = {
    [BLINK_DOUBLE] = double_blink,
    [BLINK_TRIPLE] = triple_blink,
    [BLINK_NUM] = NULL,
};

led_indicator_config_t config = {
    .mode = LED_GPIO_MODE,
    .led_gpio_config = {
        .active_level = 1,
        .gpio_num = 1,
    },
    .blink_lists = led_blink_lst,
    .blink_list_num = BLINK_MAX,
};
```

通过定义 `led_blink_lst[]` 实现自定义指示灯。

### 4.3.6 gamma 曲线调光

人眼感知亮度的方式并非线性，而是具有一定的非线性特征。在标准情况下，人眼对较暗的区域更敏感，对较亮的区域不太敏感。然而，在数字显示设备（如显示器）上，图像的亮度值通常以线性方式编码。这就导致在将线性编码的亮度值转换为人眼感知的亮度时，图像会出现明暗失真或细节丢失的情况。为了解决这个问题，需要对图像进行 Gamma 校正。Gamma 校正是通过对亮度值进行非线性调整来纠正图像的显示。通过施加一个 Gamma 值（通常介于 2.2 至 2.4 之间），可以将线性编码的亮度值映射到更符合人眼感知的非线性亮度曲线上。这样可以提高暗部细节的可见性，并使图像在视觉上更加准确和平衡。

```
float gamma = 2.3;
led_indicator_new_gamma_table(gamma);
```

默认的 gamma 表是 2.3，可以通过 `led_indicator_new_gamma_table()` 生成新的 gamma 表。

### 4.3.7 驱动电平设置

对于不同的硬件，可能分为共阳接法和共阴接法。可以将设置中的 `is_active_level_high` 设置为 `true` 或 `false` 来设置驱动电平。

### 4.3.8 API 参考

#### Header File

- [components/led/led\\_indicator/include/led\\_indicator.h](#)



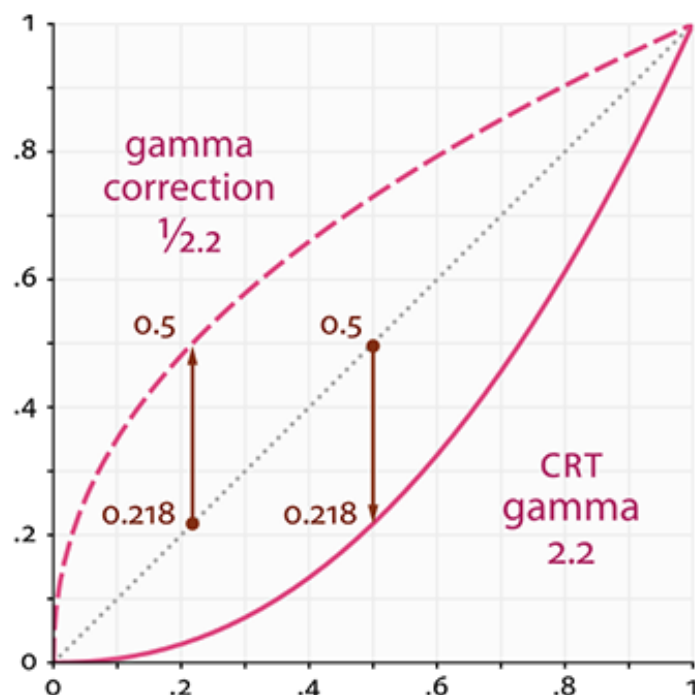


图 35: Gamma 曲线

## Functions

`led_indicator_handle_t led_indicator_create` (const `led_indicator_config_t` \*config)

create a LED indicator instance with GPIO number and configuration

参数 **config** –configuration of the LED, eg. GPIO level when LED off

返回 `led_indicator_handle_t` handle of the LED indicator, NULL if create failed.

`esp_err_t led_indicator_delete` (`led_indicator_handle_t` handle)

delete the LED indicator and release resource

参数 **handle** –pointer to LED indicator handle

返回 `esp_err_t`

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_OK` Success
- `ESP_FAIL` Delete fail

`esp_err_t led_indicator_start` (`led_indicator_handle_t` handle, int blink\_type)

start a new blink\_type on the LED indicator. if multiple blink\_type started simultaneously, it will be executed according to priority.

参数

- **handle** –LED indicator handle
- **blink\_type** –predefined blink type

返回 `esp_err_t`

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_ERR_NOT_FOUND` no predefined blink\_type found
- `ESP_OK` Success

`esp_err_t led_indicator_stop` (`led_indicator_handle_t` handle, int blink\_type)

stop a blink\_type. you can stop a blink\_type at any time, no matter it is executing or waiting to be executed.

参数

- **handle** –LED indicator handle

- **blink\_type** –predefined blink type
- 返回 esp\_err\_t
- ESP\_ERR\_INVALID\_ARG if parameter is invalid
  - ESP\_ERR\_NOT\_FOUND no predefined blink\_type found
  - ESP\_OK Success

esp\_err\_t **led\_indicator\_preempt\_start** (*led\_indicator\_handle\_t* handle, int blink\_type)

Immediately execute an action of any priority. Until the action is executed, or call led\_indicator\_preempt\_stop().

#### 参数

- **handle** –LED indicator handle
- **blink\_type** –predefined blink type

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail
- ESP\_ERR\_INVALID\_ARG if parameter is invalid

esp\_err\_t **led\_indicator\_preempt\_stop** (*led\_indicator\_handle\_t* handle, int blink\_type)

Stop the current preemptive action.

#### 参数

- **handle** –LED indicator handle
- **blink\_type** –predefined blink type

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail
- ESP\_ERR\_INVALID\_ARG if parameter is invalid

uint8\_t **led\_indicator\_get\_brightness** (*led\_indicator\_handle\_t* handle)

Get the current brightness value of the LED indicator.

参数 **handle** –LED indicator handle

返回 uint8\_t Current brightness value: 0-255 if handle is null return 0

esp\_err\_t **led\_indicator\_set\_on\_off** (*led\_indicator\_handle\_t* handle, bool on\_off)

Set the LED indicator on or off.

---

**备注:** If you have an RGB/Strips type of light, this API will control the last LED index you set, and the color will be displayed based on the last color you set.

---

#### 参数

- **handle** –LED indicator handle.
- **on\_off** –true: on, false: off

返回 esp\_err\_t

- ESP\_OK: Success
- ESP\_FAIL: Failure
- ESP\_ERR\_INVALID\_ARG: Invalid parameter

esp\_err\_t **led\_indicator\_set\_brightness** (*led\_indicator\_handle\_t* handle, uint32\_t brightness)

Set the brightness for the LED indicator.

#### 参数

- **handle** –LED indicator handle.
- **brightness** –Brightness value to set (0 to 255). You can control a specific LED by specifying the index using SET\_IB, and set it to MAX\_INDEX 127 to control all LEDs. This feature is only supported for LEDs of type LED\_RGB\_MODE. Index: (0-126), set (127) to control all.

返回 esp\_err\_t

- ESP\_OK: Success
- ESP\_FAIL: Failure
- ESP\_ERR\_INVALID\_ARG: Invalid parameter

uint32\_t **led\_indicator\_get\_hsv** (*led\_indicator\_handle\_t* handle)

Get the HSV color of the LED indicator.

---

**备注:** Index settings are only supported for LED\_RGB\_MODE.

---

**参数** **handle** –LED indicator handle.

**返回** HSV color value H: 0-360, S: 0-255, V: 0-255

esp\_err\_t **led\_indicator\_set\_hsv** (*led\_indicator\_handle\_t* handle, uint32\_t hsv\_value)

Set the HSV color for the LED indicator.

---

**备注:** Index settings are only supported for LED\_RGB\_MODE.

---

**参数**

- **handle** –LED indicator handle.
- **hsv\_value** –HSV color value to set. I: 0-126, set 127 to control all H: 0-360, S: 0-255, V: 0-255

**返回** esp\_err\_t

- ESP\_OK: Success
- ESP\_FAIL: Failure
- ESP\_ERR\_INVALID\_ARG: Invalid parameter

uint32\_t **led\_indicator\_get\_rgb** (*led\_indicator\_handle\_t* handle)

Get the RGB color of the LED indicator.

---

**备注:** Index settings are only supported for LED\_RGB\_MODE.

---

**参数** **handle** –LED indicator handle.

**返回** RGB color value (0xRRGGBB) R: 0-255, G: 0-255, B: 0-255

esp\_err\_t **led\_indicator\_set\_rgb** (*led\_indicator\_handle\_t* handle, uint32\_t rgb\_value)

Set the RGB color for the LED indicator.

---

**备注:** Index settings are only supported for LED\_RGB\_MODE.

---

**参数**

- **handle** –LED indicator handle.
- **rgb\_value** –RGB color value to set (0xRRGGBB). I: 0-126, set 127 to control all R: 0-255, G: 0-255, B: 0-255

**返回** esp\_err\_t

- ESP\_OK: Success
- ESP\_FAIL: Failure
- ESP\_ERR\_INVALID\_ARG: Invalid parameter

esp\_err\_t **led\_indicator\_set\_color\_temperature** (*led\_indicator\_handle\_t* handle, const uint32\_t temperature)

Set the color temperature for the LED indicator.

---

**备注:** Index settings are only supported for LED\_RGB\_MODE.

---

#### 参数

- **handle** –LED indicator handle.
- **temperature** –Color temperature of LED (0xIIITTTTTT) I: 0-126, set 127 to control all, TTTTTT: 0-1000000

返回 `esp_err_t`

- ESP\_OK: Success
- ESP\_FAIL: Failure
- ESP\_ERR\_INVALID\_ARG: Invalid parameter

## Structures

struct **blink\_step\_t**

one blink step, a meaningful signal consists of a group of steps

### Public Members

*blink\_step\_type\_t* **type**

action type in this step

uint32\_t **value**

hold on or off, set 0 if LED\_BLINK\_STOP() or LED\_BLINK\_LOOP

uint32\_t **hold\_time\_ms**

hold time(ms), set 0 if not LED\_BLINK\_HOLD

struct **led\_indicator\_config\_t**

LED indicator specified configurations, as a arg when create a new indicator.

### Public Members

*led\_indicator\_mode\_t* **mode**

LED work mode, eg. GPIO or pwm mode

`led_indicator_gpio_config_t` \***led\_indicator\_gpio\_config**

LED GPIO configuration

`led_indicator_ledc_config_t` \***led\_indicator\_ledc\_config**

LED LEDC configuration

`led_indicator_rgb_config_t` \***led\_indicator\_rgb\_config**

LED RGB configuration

`led_indicator_strips_config_t *led_indicator_strips_config`  
LED LEDC rgb configuration

`led_indicator_custom_config_t *led_indicator_custom_config`  
LED custom configuration

`union led_indicator_config_t::[anonymous] [anonymous]`  
LED configuration

`const blink_step_t **blink_lists`  
user defined LED blink lists

`uint16_t blink_list_num`  
number of blink lists

## Type Definitions

`typedef void *led_indicator_handle_t`  
LED indicator operation handle

## Enumerations

`enum [anonymous]`  
LED state: 0-100, only hardware that supports to set brightness can adjust brightness.  
*Values:*

enumerator `LED_STATE_OFF`  
turn off the LED

enumerator `LED_STATE_25_PERCENT`  
25% brightness, must support to set brightness

enumerator `LED_STATE_50_PERCENT`  
50% brightness, must support to set brightness

enumerator `LED_STATE_75_PERCENT`  
75% brightness, must support to set brightness

enumerator `LED_STATE_ON`  
turn on the LED

`enum blink_step_type_t`  
actions in this type  
*Values:*

enumerator `LED_BLINK_STOP`  
stop the blink

enumerator **LED\_BLINK\_HOLD**

hold the on-off state

enumerator **LED\_BLINK\_BREATHE**

breathe state

enumerator **LED\_BLINK\_BRIGHTNESS**

set the brightness, it will transition from the old brightness to the new brightness

enumerator **LED\_BLINK\_RGB**

color change with R(0-255) G(0-255) B(0-255)

enumerator **LED\_BLINK\_RGB\_RING**

Gradual color transition from old color to new color in a color ring

enumerator **LED\_BLINK\_HSV**

color change with H(0-360) S(0-255) V(0-255)

enumerator **LED\_BLINK\_HSV\_RING**

Gradual color transition from old color to new color in a color ring

enumerator **LED\_BLINK\_LOOP**

loop from first step

enum **led\_indicator\_mode\_t**

LED indicator blink mode, as a member of [led\\_indicator\\_config\\_t](#).

*Values:*

enumerator **LED\_GPIO\_MODE**

blink with max brightness

enumerator **LED\_LEDC\_MODE**

blink with LEDC driver

enumerator **LED\_RGB\_MODE**

blink with RGB driver

enumerator **LED\_STRIPS\_MODE**

blink with LEDC strips driver

enumerator **LED\_CUSTOM\_MODE**

blink with custom driver

## 4.4 LCD 工具

### 4.4.1 ESP LV SPNG

允许在 LVGL 中使用 PNG 图像。此外，还支持一种自定义格式，称为分片 PNG (SPNG)，这种格式在嵌入式系统上可以更优化地解码。参考 [SJPG](#) 的实现。

#### 功能

- 支持标准 PNG 和自定义 SPNG 格式。
- 解码标准 PNG 需要的 RAM 大小是整个未压缩图像的大小（建议在内存较大的设备上使用）。
- SPNG 是基于标准 PNG 的自定义格式，专门为 LVGL 设计。
- SPNG 是一种分片 PNG 格式，由多个小 PNG 片段和一个 SPNG 头部组成。
- SPNG 图像是分段部分解码，因此不支持缩放或旋转。

#### 将 PNG 转换为 SPNG

依赖组件 [esp\\_mmap\\_assets](#)。它将在编译过程中自动打包并转换 PNG 图像为 SPNG 格式。

```
[12/1448] Move and Pack assets...
--support_format: .jpg, .png
--support_spng: ON
--support_sjpg: ON
--split_height: 16
Input: temp_icon.png      RES: 90 x 90      splits: 6
Completed, saved as: temp_icon.spng
```

#### 应用示例

**注册解码器** 在 LVGL 启动后注册解码器函数。

```
esp_lv_split_png_init();
```

#### API 参考

##### Header File

- [components/display/tools/esp\\_lv\\_spng/include/esp\\_lv\\_spng.h](#)

##### Functions

`esp_err_t esp_lv_split_png_init(esp\_lv\_spng\_decoder\_handle\_t *ret_handle)`

Register the PNG decoder functions in LVGL.

**参数** `ret_handle` –Pointer to the handle where the decoder handle will be stored

**返回**

- ESP\_OK on success
- ESP\_ERR\_\* error codes on failure

`esp_err_t esp_lv_split_png_deinit(esp\_lv\_spng\_decoder\_handle\_t handle)`

Deinitialize the PNG decoder handle.

**参数** `handle` –The handle to be deinitialized

**返回**

- ESP\_OK on success
- ESP\_ERR\_\* error codes on failure

## Type Definitions

```
typedef void *esp_lv_spng_decoder_handle_t
```

Type of handle for the split PNG decoder.

## 4.4.2 ESP MMAP ASSETS

该模块主要用于打包资源（如图像、字体等），并将其直接映射以供用户访问。

### 功能

#### 添加导入文件类型

- 支持多种文件格式，如 .bin、.jpg、.ttf 等。

#### 启用分片 JPG

- 需要使用 SJPG 来解析。参见 LVGL [SJPG](#)。

#### 启用分片 PNG

- 需要使用 esp\_lv\_spng 来解析。参见组件 [esp\\_lv\\_spng](#)。

#### 设置分片高度

- 设置分片高度，依赖于 MMAP\_SUPPORT\_SJPG 或 MMAP\_SUPPORT\_SPNG。

### CMake

用户可以选择将图像与应用程序二进制文件、分区表等一起自动刷写到设备上，通过在 idf.py flash 时指定 FLASH\_IN\_PROJECT。例如：

```
/* partitions.csv
 * -----
 * | Name                | Type | SubType | Offset | Size | Flags      |
 * -----
 * | my_spiffs_partition | data | spiffs  |        | 6000K |           |
 * -----
 */
spiffs_create_partition_assets(my_spiffs_partition my_folder FLASH_IN_PROJECT)
```

### 应用示例

**生成头文件 (assets\_generate.h)** 该头文件自动生成，包含内存映射资源的基本定义。

```
#include "esp_mmap_assets.h"

#define TOTAL_MMAP_FILES      2
#define MMAP_CHECKSUM         0xB043

enum MMAP_FILES {
    MMAP_JPG_JPG = 0,    /*!< jpg.jpg */
    MMAP_PNG_PNG = 1,    /*!< png.png */
};
```

**创建资源句柄** 资源初始化配置确保与 assets\_generate.h 一致。它设置了 max\_files 和 checksum，用来验证头文件和内存映射的二进制文件是否匹配。



```

mmap_assets_handle_t asset_handle;

const mmap_assets_config_t config = {
    .partition_label = "my_spiffs_partition",
    .max_files = TOTAL_MMAP_FILES,
    .checksum = MMAP_CHECKSUM,
};

ESP_ERROR_CHECK(mmap_assets_new(&config, &asset_handle));

```

**资源使用** 可以使用 `assets_generate.h` 中定义的枚举来获取资源信息。

```

const char *name = mmap_assets_get_name(asset_handle, MMAP_JPG_JPG);
const void *mem = mmap_assets_get_mem(asset_handle, MMAP_JPG_JPG);
int size = mmap_assets_get_size(asset_handle, MMAP_JPG_JPG);
int width = mmap_assets_get_width(asset_handle, MMAP_JPG_JPG);
int height = mmap_assets_get_height(asset_handle, MMAP_JPG_JPG);

ESP_LOGI(TAG, "Name:[%s], Mem:[%p], Size:[%d bytes], Width:[%d px], Height:[%d px]
↪", name, mem, size, width, height);

```

## API 参考

### Header File

- [components/display/tools/esp\\_mmap\\_assets/include/esp\\_mmap\\_assets.h](#)

### Functions

`esp_err_t mmap_assets_new (const mmap\_assets\_config\_t *config, mmap\_assets\_handle\_t *ret_item)`

Create a new asset instance.

#### 参数

- **config** –[in] Pointer to the asset configuration structure.
- **ret\_item** –[out] Pointer to the handle of the newly created asset instance.

#### 返回

- ESP\_OK: Success
- ESP\_ERR\_NO\_MEM: Insufficient memory
- ESP\_ERR\_NOT\_FOUND: Can't find partition
- ESP\_ERR\_INVALID\_SIZE: File num mismatch
- ESP\_ERR\_INVALID\_CRC: Checksum mismatch

`esp_err_t mmap_assets_del (mmap\_assets\_handle\_t handle)`

Delete an asset instance.

**参数** **handle** –[in] Asset instance handle.

#### 返回

- ESP\_OK: Success
- ESP\_ERR\_INVALID\_ARG: Invalid argument

`const uint8_t *mmap_assets_get_mem (mmap\_assets\_handle\_t handle, int index)`

Get the memory of the asset at the specified index.

#### 参数

- **handle** –[in] Asset instance handle.
- **index** –[in] Index of the asset.

**返回** Pointer to the asset memory, or NULL if index is invalid.

```
const char *mmap_assets_get_name (mmap_assets_handle_t handle, int index)
```

Get the name of the asset at the specified index.

**参数**

- **handle** –[in] Asset instance handle.
- **index** –[in] Index of the asset.

**返回** Pointer to the asset name, or NULL if index is invalid.

```
int mmap_assets_get_size (mmap_assets_handle_t handle, int index)
```

Get the size of the asset at the specified index.

**参数**

- **handle** –[in] Asset instance handle.
- **index** –[in] Index of the asset.

**返回** Size of the asset, or -1 if index is invalid.

```
int mmap_assets_get_width (mmap_assets_handle_t handle, int index)
```

Get the width of the asset at the specified index.

**参数**

- **handle** –[in] Asset instance handle.
- **index** –[in] Index of the asset.

**返回** Width of the asset, or -1 if index is invalid.

```
int mmap_assets_get_height (mmap_assets_handle_t handle, int index)
```

Get the height of the asset at the specified index.

**参数**

- **handle** –[in] Asset instance handle.
- **index** –[in] Index of the asset.

**返回** Height of the asset, or -1 if index is invalid.

## Structures

```
struct mmap_assets_config_t
```

Asset configuration structure, contains the asset table and other configuration information.

## Public Members

```
const char *partition_label
```

Configuration partition\_label

```
int max_files
```

Number of assets

```
uint32_t checksum
```

Checksum of table

## Type Definitions

```
typedef struct mmap_assets_t *mmap_assets_handle_t
```

Asset handle type, points to the asset.

Type of asset handle



# Chapter 5

## USB 主机 & 设备

### 5.1 USB 外设综述

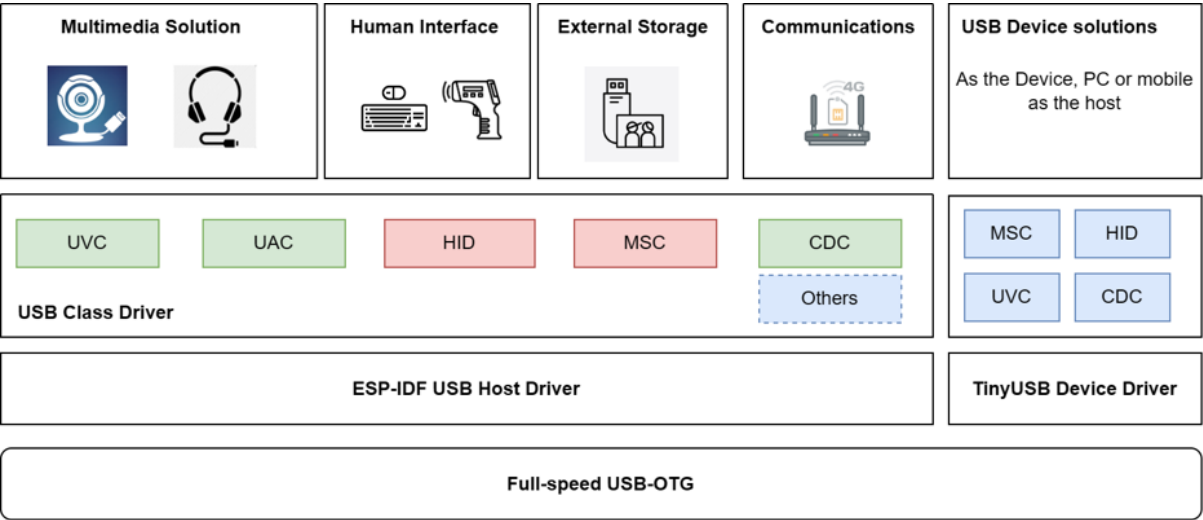
#### 5.1.1 ESP USB 外设介绍

##### USB 简介

USB (Universal Serial Bus) 是一种通用的总线标准，用于连接主机和外设设备。USB 主机可以通过 USB 接口与 USB 设备连接，实现数据传输、电源供给等功能。

USB IF (USB Implementers Forum) 是 USB 标准的制定者，它制定了 USB 标准，包括 USB 1.1、USB 2.0、USB 3.0 等，定义了 USB 接口的物理层、数据链路层、传输层、会话层、表示层等协议，以及 USB 设备类 (Device Class) 标准，常见的设备类包括 HID (Human Interface Device，人机接口设备)、MSC (Mass Storage Class，大容量存储设备)、CDC (Communication Device Class，通信设备)、Audio、Video 等。

乐鑫 ESP32-S2/S3/C3 等芯片均已内置 USB-OTG 或 USB-Serial-JTAG 外设，支持各种各样的 USB 应用，包括 USB 多媒体类应用，USB 通信类应用，USB 存储类应用，USB 人机交互类应用等。



**USB 电气属性** Type-A 接口的 USB 电气属性如下：

Pin	Name	Cable color	Description
1	VBUS	Red	+5V
2	D-	White	Data- (0 或 3.3V)
3	D+	Green	Data+ (0 或 3.3V)
4	GND	Black	Ground

- 对于自供电设备，需要使用 1 个额外 IO 检测 VBUS 电压，用于检测设备是否拔出
- D- D+ 接反不会损坏硬件，但是主机将无法识别

**USB-OTG Full-speed 控制器简介** USB OTG Full-speed 控制器是指同时具有 USB-OTG，USB Host 和 USB Device 模式的控制器，支持模式的协商和切换。支持 Full-speed (12Mbps) 和 Low-speed (1.5Mbps) 两种速率，支持 USB 1.1 和 USB 2.0 协议。

ESP-IDF 从 v4.4 开始已经包含 USB Host 和 USB Device 协议栈和各种设备类驱动，支持用户二次开发。

更多介绍，请参考[USB-OTG 控制器介绍](#)。

**USB-Serial-JTAG 控制器简介** USB-Serial-JTAG Controller：同时具有 USB Serial 和 USB JTAG 功能的专用 USB 控制器，支持通过 USB 接口下载固件、打印 log、CDC 传输和 JTAG 调试，不支持修改 USB 功能、修改描述符等二次开发。

更多介绍，请参考[USB-Serial-JTAG 控制器介绍](#)。

**USB Full-speed PHY 简介** USB Full-speed PHY：也称 USB Full-speed Transceiver，用于 USB Controller 数字信号到 USB 总线信号电平转换，提供总线驱动能力等。内部 USB Full-speed PHY 连接到外部固定 IO 引脚。

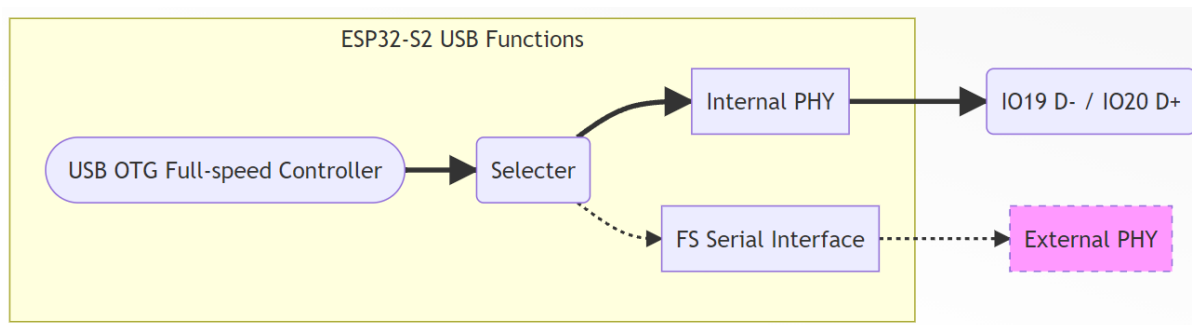
更多介绍，请参考[USB-PHY 介绍](#)。

#### ESP32-S/C 系列 USB 外设支持情况

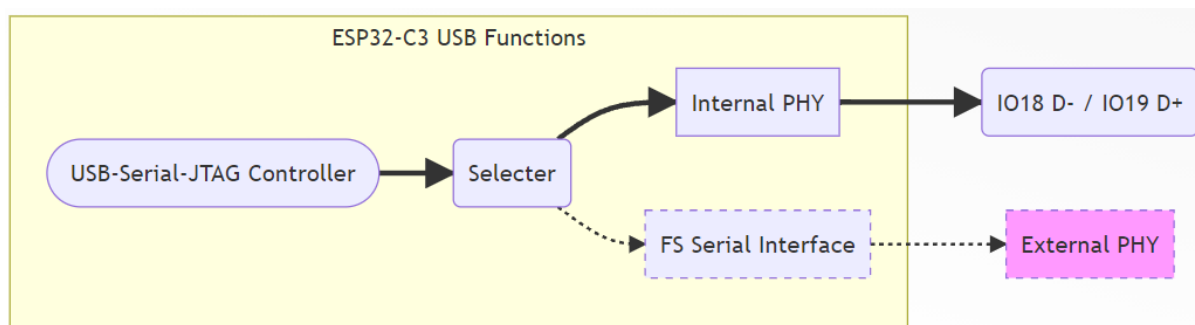
	USB OTG High-speed	USB OTG Full-speed	USB-Serial-JTAG	Fulls-speed PHY	High-speed PHY
ESP32-P4	√	√	√	√	√
ESP32-S3	X	√	√	√	X
ESP32-S2	X	√	X	√	X
ESP32-C6	X	X	√	√	X
ESP32-C3	X	X	√	√	X
ESP32-C2	X	X	X	X	X
ESP32	X	X	X	X	X
ESP8266	X	X	X	X	X

- √ : Supported
- X : Not Supported

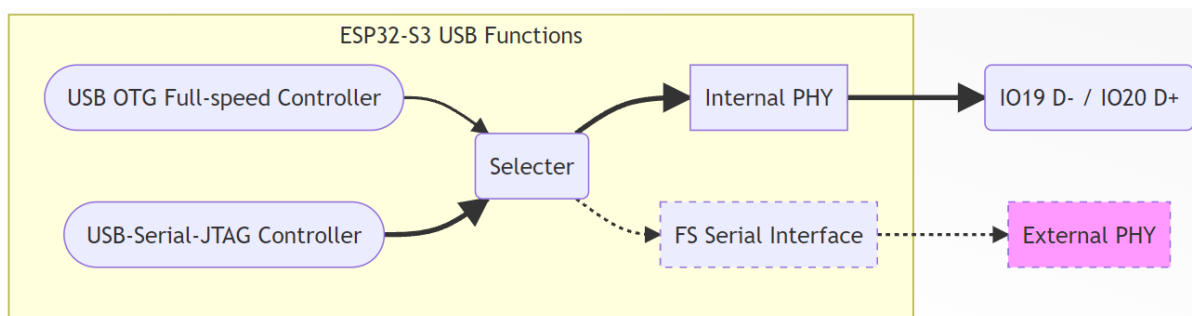
**ESP32-S2 USB 功能简介** ESP32-S2 内置 USB OTG Full-speed Controller 和 USB Full-speed PHY，内部结构如下：



**ESP32-C3 USB 功能简介** ESP32-C3 内置 **USB-Serial-JTAG Controller** 和 **USB Full-speed PHY**，内部结构如下：



**ESP32-S3 USB 功能简介** ESP32-S3 内置两个 USB 控制器，分别是 **USB OTG Full-speed Controller** 和 **USB-Serial-JTAG Controller**，内置一个 **USB Full-speed PHY**。内部 USB PHY 默认连接到 **USB-Serial-JTAG** 控制器，可通过烧写 eFuse 修改默认，或配置寄存器动态切换，也可通过增加外部 PHY，同时启用两个控制器。内部 USB PHY 的切换详情，参考 [USB PHY 切换](#)。



### 5.1.2 USB-OTG 外设介绍

ESP32-S2/S3 等芯片内置 USB-OTG 外设，它包含了 USB 控制器和 USB PHY，支持通过 USB 线连接到 PC，实现 USB Host 和 USB Device 功能。

#### USB-OTG 传输速率

ESP32-S2/S3 USB-OTG Full Speed 总线传输速率为 12 Mbps，但由于 USB 传输存在一些校验和同步机制，实际的有效传输速率将低于 12 Mbps。具体数值和传输类型相关，如下表所示：

传输类型	控制	中断	批量	同步
适用场合	设备初始化和管 理	鼠标和键盘	打印机和批量存 储	流式音频和视频
支持低速	有	有	无	无
校验重传	有	有	有	无
保证传输速度	无	无	无	有
使用固定带宽	有 (10%)	有 (90%)	无	有 (90%)
减少延迟时间	无	有	无	有
传输的最大尺寸	64 字节	64 字节	64 字节	~512 字节
每毫秒传输包数 量	•	1	19	1
理论有效速率	•	64000 Bytes/s	1216000 Bytes/s	512000 Bytes/s

- 传输速率的计算公式为：传输速率 (Bytes/s) = 传输的最大尺寸 \* 每毫秒传输包数量 \* 1000
- 控制传输用于传输设备控制信息，包含多个阶段，有效传输速率需要按照协议栈的实现来计算。

### USB-OTG 外设内置功能

**使用 USB OTG Console 下载固件和打印 LOG** ESP32-S2/S3 等内置 USB-OTG 外设的芯片，ROM Code 中内置了 USB 通信设备类 (CDC) 的功能，该功能可用于替代 UART 接口，实现 Log、Console 和固件下载功能。

1. 由于 USB OTG Console 默认为关闭状态，如需使用它下载固件，需要通过以下方法完成初次下载：
  1. 在 menuconfig 中先使能 USB OTG Console 功能，然后编译固件
  2. 手动芯片的 Boot 控制引脚拉低，然后将芯片通过 USB 线连接到 PC，进入下载模式。PC 会发现新的串口设备，Windows 为 COM\*，Linux 为 /dev/ttyACM\*，MacOS 为 /dev/cu\*。
  3. 使用 esptool 工具（或直接使用 idf.py flash）配置设备对应的串口号下载固件。
2. 初次下载完成以后，USB OTG Console 功能将自动使能，即可通过 USB 线连接到 PC，PC 会发现新的串口设备，Windows 为 COM\*，Linux 为 /dev/ttyACM\*，MacOS 为 /dev/cu\*，LOG 数据将从该虚拟串口打印。
3. 用户无需再手动拉低 Boot 控制引脚，使用 esptool 工具（或直接使用 idf.py flash）配置设备对应的串口号即可下载固件，下载期间，esptool 通过 USB 控制协议自动将设备 Reset 并切换到下载模式。

更多详细信息，请参考：[USB OTG Console](#)

**使用 USB OTG DFU 下载固件** ESP32-S2/S3 等内置 USB-OTG 外设的芯片，ROM Code 中内置了 USB DFU (Device Firmware Upgrade) 功能，可用于实现标准的 DFU 下载模式。

1. 使用 DFU 下载固件，用户每次都需要手动进入下载模式，将芯片的 Boot 控制引脚拉低，然后通过 USB 线连接到 PC。
2. 在工程目录下运行指令 `idf.py dfu` 生成 DFU 固件，然后使用 `idf.py dfu-flash` 下载固件。
3. 如果存在多个 DFU 设备，用户可以使用 `idf.py dfu-list` 查看 DFU 设备列表，然后使用 `idf.py dfu-flash --path <path>` 指定下载端口。

更多详细信息，请参考：[Device Firmware Upgrade via USB](#)

### 使用 USB-OTG 外设进行 USB Host 开发

USB-OTG 外设支持 USB Host 功能，用户可以通过 USB 接口直接连接到外部 USB 设备。ESP-IDF 从 v4.4 版本开始，已经支持 USB Host Driver，用户可以参考 [ESP-IDF USB Host](#)，开发 USB Class Driver。

此外乐鑫也已经官方支持 USB Host HID，USB Host MSC，USB Host CDC，USB Host UVC 等设备类驱动，用户可以直接使用这些驱动进行应用开发。

USB Host 方案详情，请参考 [USB Host Solution](#)。

## 使用 USB-OTG 外设进行 USB Device 开发

USB-OTG 外设支持 USB Device 功能，乐鑫已经官方适配了 TinyUSB 协议栈，用户可以直接使用基于 TinyUSB 开源协议栈开发的 USB 标准设备或自定义设备，例如 HID，MSC，CDC，ECM，UAC 等。

USB Device 方案详情，请参考 [USB Device Solution](#)。

### 5.1.3 USB-Serial-JTAG 外设介绍

ESP32-S3/C3 等芯片内置 USB-Serial-JTAG 外设，它包含了 USB-to-serial 转换器和 USB-to-JTAG 转换器，支持通过 USB 线连接到 PC，实现固件下载、调试和打印系统 LOG 等功能。USB-Serial-JTAG 外设的内部结构可参考 [ESP32-C3 技术参考手册-USB Serial/JTAG Controller](#)。

#### USB-Serial-JTAG 外设驱动

- Linux 和 MacOS 系统下，无需手动安装驱动
- Windows 10 及以上系统，联网将自动安装驱动
- Windows 7/8 系统，需要手动安装驱动，驱动下载地址：[esp32-usb-jtag-2021-07-15](#)。用户也可以使用 [ESP-IDF Windows Installer](#)，勾选 USB-Serial-JTAG 驱动进行安装，

#### USB-Serial-JTAG 外设内置功能

USB-Serial-JTAG 外设接入 PC 后，设备管理器将新增两个设备：

Windows 如下图所示：



Linux 如下图所示：



random	tty0	tty2	tty30	tty41	tty52	tty63
rftkill	tty1	tty20	tty31	tty42	tty53	tty7
rtc	tty10	tty21	tty32	tty43	tty54	tty8
rtc0	tty11	tty22	tty33	tty44	tty55	tty9
serial	tty12	tty23	tty34	tty45	tty56	ttyACM0
shm	tty13	tty24	tty35	tty46	tty57	ttyprintk
snapshot	tty14	tty25	tty36	tty47	tty58	ttyS0
snd	tty15	tty26	tty37	tty48	tty59	ttyS1
stderr	tty16	tty27	tty38	tty49	tty6	ttyS10
stdin	tty17	tty28	tty39	tty5	tty60	ttyS11
stdout	tty18	tty29	tty4	tty50	tty61	ttyS12
tty	tty19	tty3	tty40	tty51	tty62	ttyS13

### 使用 USB-Serial-JTAG 下载固件

- 默认情况下，USB-Serial-JTAG 下载功能处于使能状态，可以直接使用 USB 线连接到 PC，然后使用 esptool 工具（或直接使用 idf.py flash）配置 USB-Serial-JTAG 设备对应的串口号（Windows 为 COM\*，Linux 为 /dev/ttyACM\*，MacOS 为 /dev/cu\*）下载固件。下载期间，esptool 通过 USB 控制协议自动将设备 Reset 并切换到下载模式。
- 如果在应用程序中将 USB-Serial-JTAG 对应的 USB 引脚用作了其它功能，例如用作普通 GPIO 或其它外设 IO，USB-Serial-JTAG 将无法与 USB 主机建立连接，因此无法通过 USB 将设备切换到下载模式，用户必须通过 Boot 控制引脚，将设备手动切换到下载模式，然后再使用 esptool 下载固件。
- 为了避免在应用程序中将 USB-Serial-JTAG 对应的 USB 引脚用作其它功能，导致无法通过 USB 自动进入下载模式，用户需要在硬件设计时，引出 Boot 控制引脚。
- 默认情况下，通过 USB 接口下载不同的芯片，COM 号将递增，可能对量产造成不便，用户可参考 [阻止 Windows 依据 USB 设备序列号递增 COM 编号](#)。

### 使用 USB-Serial-JTAG 调试代码

USB-Serial-JTAG 支持通过 JTAG 接口调试代码，用户仅需使用 USB 线连接到 PC，然后使用 OpenOCD 工具即可调试代码。配置方法请参考 [配置 ESP32-C3 内置 JTAG 接口](#)。

### 使用 USB-Serial-JTAG 打印系统 LOG

- 用户可通过 menuconfig-> Component config -> ESP System Settings -> Channel for console secondary output 配置 USB-Serial-JTAG LOG 功能的使能状态。
- LOG 功能使能以后，可以直接使用 USB 线连接到 PC，然后使用 idf.py monitor 或其它串口工具打开 USB-Serial-JTAG 设备对应的串口号（Windows 为 COM\*，Linux 为 /dev/ttyACM\*，MacOS 为 /dev/cu\*），即可打印系统 LOG。
- USB-Serial-JTAG 仅在主机接入后才会打印 LOG，如果主机未接入，USB-Serial-JTAG 不会被初始化，也不会打印 LOG。
- USB-Serial-JTAG LOG 功能无法在睡眠模式下使用（包括 deep sleep 和 light sleep 模式），如果需要在睡眠模式下打印 LOG，可以使用 UART 接口。

### 使用 USB-Serial-JTAG 引脚作为普通 GPIO

如果用户需要在应用程序中将 USB-Serial-JTAG 对应的 USB 引脚用作其它功能，例如用作普通 GPIO。需要注意 USB D+ 接口默认上拉 USB 电阻，该电阻会导致 USB D+ 引脚常为高电平，因此需要在用作 GPIO 时将其禁用。

- ESP-IDF v4.4 及以后版本 GPIO 驱动默认会禁用 USB D+ 上拉电阻，用户使用 GPIO 驱动时无需额外配置。
- 用户也可以修改寄存器值 `USB_SERIAL_JTAG.conf0.dp_pullup = 0`；将 USB D+ 上拉电阻禁用。

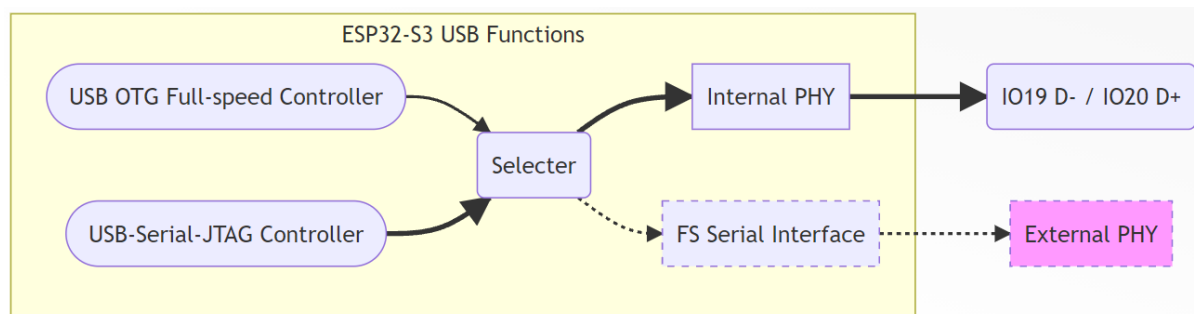
需要特别注意的是 USB D+ 引脚的上拉电阻在上电时刻即存在，用户在调用软件禁用上拉电阻之前，USB D+ 引脚已经被拉高，导致 D+ 引脚做为 GPIO 时，初始阶段为高电平，如果用户需要在上电后 USB D+ 引脚立即为低电平，需要在硬件设计时，将 USB D+ 引脚通过外部电路拉低。

### 5.1.4 USB PHY/Transceiver 介绍

USB Full-speed PHY/Transceiver 的功能是将 USB 控制器的数字信号转换为 USB 总线信号电平，提供总线驱动能力，检测接收错误等。ESP32-S2/S3 等芯片已内置一个 USB Full-speed PHY，用户可直接使用芯片指定的 USB D+ D- 与外部 USB 系统通信。此外，ESP32-S2/S3 还保留了外部 PHY 的扩展接口，用户可在需要时连接外部 PHY。

#### 使用内部 PHY

ESP32-S2/S3/C3 内部集成了 USB PHY，因此无需外接 PHY 芯片，可以直接与外部 USB 主机或设备通过 USB D+/D- 连接。但对于集成两个 USB 控制器的芯片，例如 ESP32-S3 内置 USB-OTG 和 USB-Serial-JTAG，两者共用一个内部 PHY，同一时间只能有一个工作。



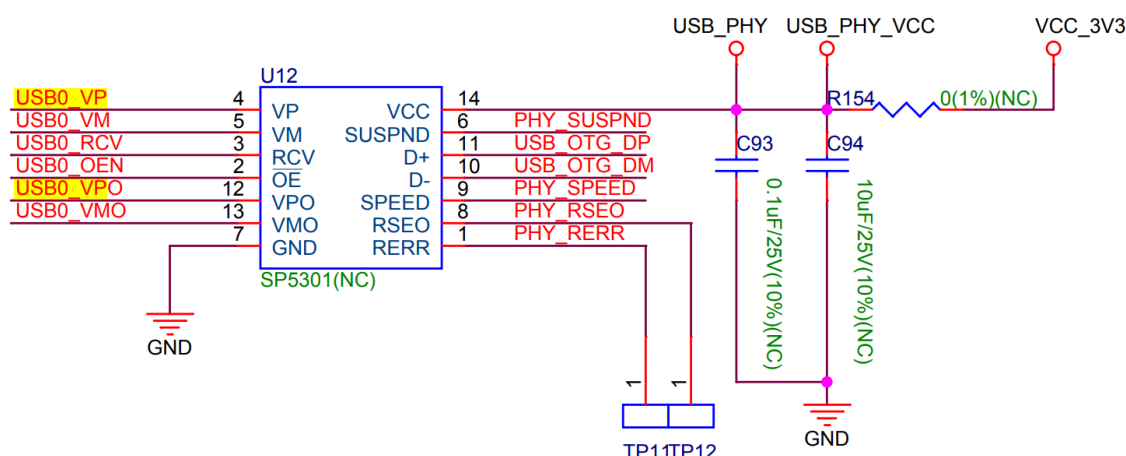
内部 USB-PHY 对应固定的 GPIO，如下表所示：

	D+	D-
ESP32-S2	20	19
ESP32-S3	20	19
ESP32-C3	19	18
ESP32-C6	13	12

#### 使用外部 PHY

通过增加一个外部 PHY，可以实现 USB-OTG 和 USB-Serial-JTAG 两个外设同时工作。

ESP32S2/S3 支持 SP5301 或同等功能的 USB PHY。外部 PHY 的典型电路图如下：



使用外部 USB PHY，将占用至少 6 个 GPIO

### USB PHY 默认配置

1. 对于同时具有 USB-OTG 和 USB-Serial-JTAG 两个外设的芯片，默认情况下 USB-Serial-JTAG 与内部 USB-PHY 连接。用户可以直接通过 USB 接口进行下载或调试，无需额外配置。
2. 如需使用 USB Host Driver 或 TinyUSB 协议栈开发 USB-OTG 应用，在协议栈初始化时，USB-PHY 连接会自动切换为 USB-OTG，用户不必再进行配置。在 USB-OTG 模式下，如果用户需要使用 USB-Serial-JTAG 的下载功能，需要手动 Boot 到下载模式。

### 修改 USB PHY 默认配置

方法 1：通过配置寄存器，将 USB-PHY 连接切换为 USB-OTG。

- USB Host Driver 或 TinyUSB 协议栈内部通过配置 USB PHY 寄存器，将内部 USB-PHY 连接切换为 USB-OTG，如需了解更多信息，请参考 [USB PHY 配置 API](#)。

方法 2：通过烧写 efuse usb\_phy\_sel 位为 1，将 USB-PHY 默认连接切换为 USB-OTG：

- 仅当用户需要在 Boot 模式下使用 USB-OTG 功能时，才需要烧写此 efuse 位。烧写完成后，芯片进入 Boot 模式后，将使用 USB-OTG 提供的下载功能，例如 USB DFU。
- 注意，烧写 efuse 为 1 后不可恢复为 0。当 USB-PHY 默认连接切换为 USB-OTG，芯片进入 Boot 模式后，将使用 USB-OTG 功能，USB-Serial-JTAG 功能将无法使用。
- 注意：对于在 DateCode 2219 生产的 ESP32-S3 模组和开发板 (PW No. 早于 PW-2022-06-XXXX)，由于 EFUSE\_DIS\_USB\_OTG\_DOWNLOAD\_MODE (BLK0 B19[7]) 已经被烧录为 1 (USB OTG 下载被禁用)，用户如果再烧写 efuse\_usb\_phy\_sel 位为 1，将导致芯片进入 Boot 模式后，USB-Serial-JTAG 和 USB-OTG 下载功能均无法使用。

## 5.1.5 USB VID 和 PID

VID 和 PID 是 USB 设备的唯一标识符，用于区分不同的 USB 设备。一般 VID 和 PID 由 USB-IF 分配，USB-IF 是 USB 设备标准制定者。

### 以下情况您可以免申请 VID 和 PID

- 如果您的产品使用的是 USB Host 模式，您不需要申请 VID 和 PID。

- 如果您的产品使用的是 USB Device 模式，准备使用乐鑫的 VID (0x303A)，并且基于 TinyUSB 协议栈开发 USB 标准设备，那您不需要申请 PID，使用 TinyUSB 默认 PID 即可。

## 申请 VID 和 PID

如果您使用的产品需要使用 USB 设备模式，您可按照以下过程申请 VID (Vendor ID) 或 PID (Product ID)。

- 如果您的产品需要使用 USB-IF 分配的 VID，您需要先 [注册成为 USB-IF 成员](#)，然后按照 USB-IF 的流程申请 VID 和 PID。
- 如果您的产品考虑使用乐鑫的 VID，您可以直接申请 PID（免费），申请流程请参考 [申请乐鑫 PID](#)。

注意：使用乐鑫的 VID 和 TinyUSB 的默认 PID，并不意味着您的产品符合 USB 规范，您仍然需要进行 USB 认证。

## USB 认证

USB 认证是由 USB Implementers Forum (USB-IF) 进行管理的，旨在确保产品符合 USB 规范，以保证设备之间的互操作性和兼容性。

USB 认证是一个可选的过程，但在以下情况下必须进行产品认证：

- 如果产品标榜自己符合 USB 规范并使用官方 [USB 标志](#)。
- 如果产品打算使用 USB 标志、商标或宣传资料中提及 USB 认证。

具体认证流程和要求，请查阅 <https://www.usb.org> 或联系 [USB-IF 授权测试实验室](#)。

## 5.1.6 USB Host 方案

ESP32-S2/S3 等芯片内置 USB-OTG 外设，支持 USB 主机模式，基于 ESP-IDF 提供的 USB 主机协议栈和各种 USB 主机类驱动，可以通过 USB 接口连接多种多样的 USB 设备。以下介绍了 ESP32-S2/S3 芯片支持的 USB Host 解决方案。

### ESP USB Camera 视频方案

支持通过 USB 接口连接摄像头模组，实现 MJPEG 格式视频流获取和传输，最高可支持 [480\\*800@15fps](#)。适用于猫眼门铃、智能门锁、内窥镜、倒车影像等场景。

#### 特性：

- 快速启动
- 支持热插拔
- 支持 UVC1.1/1.5 规范的摄像头
- 支持自动解析描述符
- 支持动态配置分辨率
- 支持 MJPEG 视频流传输
- 支持批量和同步两种传输模式

#### 硬件：

- 芯片：ESP32-S2，ESP32-S3
- 外设：USB-OTG
- USB 摄像头：支持 MJPEG 格式，批量传输模式下 [800\\*480@15fps](#)，同步传输模式下 [480\\*320@15fps](#)，摄像头限制详见 [usb\\_stream API 说明](#)

**链接:**

- [usb\\_stream 组件](#)
- [usb\\_stream API 说明](#)
- [USB Camera Demo 视频](#)
- 示例代码: USB 摄像头 + WiFi 图传 [usb/host/usb\\_camera\\_mic\\_spk](#)
- 示例代码: USB 摄像头 + LCD 本地刷屏 [usb/host/usb\\_camera\\_lcd\\_display](#)

**ESP USB Audio 音频方案**

支持通过 USB 接口连接 USB 音频设备, 实现 PCM 格式音频流获取和传输, 可同时支持多路 48KHz 16bit 扬声器和多路 48KHz 16bit 麦克风。支持 Type-C 接口耳机, 适用于音频播放器等场景。支持和 UVC 同时工作, 适用于门铃对讲等场景。

**特性:**

- 快速启动
- 支持热插拔
- 支持自动解析描述符
- 支持 PCM 音频流传输
- 支持动态修改采样率
- 支持多通道扬声器
- 支持多通道麦克风
- 支持音量、静音控制
- 支持和 USB Camera 同时工作

**硬件:**

- 芯片: ESP32-S2, ESP32-S3
- 外设: USB-OTG
- USB 音频设备: 支持 PCM 格式

**链接:**

- [usb\\_stream 组件](#)
- [usb\\_stream API 说明](#)
- [USB Audio Demo 视频](#)
- 示例代码: MP3 音乐播放器 + USB 耳机 [usb/host/usb\\_audio\\_player](#)

**ESP USB 4G 联网方案**

支持通过 USB 接口连接 4G Cat.1, Cat.4 模组, 实现 PPP 拨号上网。支持通过 Wi-Fi SoftAP 热点共享互联网给其它设备。适用于物联网网关、MiFi 移动热点、智慧储能、广告灯箱等场景。

**特性:**

- 快速启动
- 支持热插拔
- 支持 Modem+AT 双接口 (需要模组支持)
- 支持 PPP 标准协议 (大部分 4G 模组均支持)
- 支持 4G 转 Wi-Fi 热点
- 支持 NAT 网络地址转换
- 支持电源管理
- 支持网络自动恢复
- 支持卡检测、信号质量检测
- 支持网页配置界面

**硬件:**

- 芯片: ESP32-S2, ESP32-S3
- 外设: USB-OTG
- 4G 模组: 支持 Cat.1 Cat.4 等网络制式 4G 模组, 需要模组支持 PPP 协议

**链接:**

- [USB 4G Demo 视频](#)
- [iot\\_usbh\\_modem 组件](#)
- 示例代码: 4G Wi-Fi 路由器 [usb/host/usb\\_cdc\\_4g\\_module](#)

**ESP USB 存储方案**

支持通过 USB 接口连接标准 U 盘设备 (兼容 USB3.1/3.0/2.0 协议 U 盘), 支持将 U 盘挂载到 FatFS 文件系统, 实现文件的读写。适用于户外广告灯牌、考勤机、移动音响、记录仪等应用场景。

**特性:**

- 兼容 USB3.1/3.0/2.0 U 盘
- 默认支持最大 32G
- 支持热插拔
- 支持 Fat32/exFat 格式
- 支持文件系统读写
- 支持 U 盘 OTA

**硬件:**

- 芯片: ESP32-S2, ESP32-S3
- 外设: USB-OTG
- U 盘: 格式化为 Fat32 格式, 默认支持 32GB 以内 U 盘。大于 32GB 需要在文件系统开启 exFat

**链接:**

- [usb\\_host\\_msc 组件](#)
- [U 盘 OTA 组件](#)
- [挂载 U 盘 + 文件系统访问示例](#)

**5.1.7 USB Device 方案**

ESP32-S2/S3 等芯片内置 USB-OTG 外设, 支持 USB 设备模式, 可以通过 USB 连接到 PC 或者其他 USB 主机设备。结合设备 TinyUSB 协议栈和设备类驱动, 可用于开发多种 USB 设备, 如 HID 设备、CDC 设备、复合设备等。以下介绍基于 USB 设备的相关解决方案。

**USB 音频设备方案**

USB 音频设备方案基于 UAC 2.0 (USB Audio Class) 协议标准, 使乐鑫 SoC 可作为音频设备, 提供便捷和高质量的音频传输功能, 例如将其作为麦克风或扬声器, 连接到计算机等支持 USB 音频的设备上, 实现音频的输入和输出。

**特性：**

- 支持 UAC 2.0
- 支持多种音频格式和采样率
- 支持音频输入和输出

**硬件：**

- 芯片：ESP32-S2, ESP32-S3
- 外设：USB-OTG

**链接：**

- [ESP-BOX 带屏 USB 音响](#)

**USB 视频设备方案**

USB UVC 设备方案基于 UVC (USB Video Class) 协议标准，使乐鑫 SoC 可作为视频设备，提供便捷和高质量的视频传输功能，可应用在 USB 门铃摄像头，或 USB + Wi-Fi 双模网络摄像头。

**特性：**

- 支持 UVC 1.5
- 支持同步和批量两种传输模式
- 支持作为虚拟摄像头设备

**硬件：**

- 芯片：ESP32-S2, ESP32-S3
- 外设：USB-OTG

**链接：**

- [USB 网络摄像头](#)

**USB 存储设备方案**

USB 存储设备方案基于 MSC (Mass Storage Class) 协议标准，结合 Wi-Fi 功能，还可构建无线共享存储设备，如 USB 无线 U 盘、读卡器、数字音乐播放器、数字多媒体播放器等。

**特性：**

- USB - Wi-Fi 双向访问
- 多设备接入
- 模拟 U 盘

**硬件：**

- 芯片：ESP32-S2, ESP32-S3
- 外设：USB-OTG

**链接：**

- [USB + Wi-Fi 无线 U 盘](#)



## USB HID 设备方案

USB HID 设备方案基于 HID (Human Interface Device) 协议标准, 可作为 USB 键盘、鼠标、游戏手柄等设备, 实现人机交互功能。结合 Wi-Fi, 蓝牙, ESP-Now 等无线功能, 还可构建无线 HID 设备。

### 特性:

- 支持多种 HID 设备
- 支持自定义 HID 设备
- 支持 USB HID 和 BLE HID 双模

### 硬件:

- 芯片: ESP32-S2, ESP32-S3
- 外设: USB-OTG

### 链接:

- [USB HID 键盘和鼠标示例](#)
- [USB HID Surface Dial 示例](#)
- [USB 客制化键盘示例](#)

## U 盘拖拽升级

基于 esp-tinyuf2 作为虚拟 U 盘, 支持拖拽 UF2 固件到 U 盘实现 OTA。同时支持将 NVS 数据映射到 U 盘文件, 通过修改文件改写 NVS。

### 特性:

- 拖拽 UF2 固件进行 OTA
- 通过虚拟文件修改 NVS

### 硬件:

- 芯片: ESP32-S2, ESP32-S3
- 外设: USB-OTG

### 链接:

- [U 盘读写 NVS](#)
- [虚拟 U 盘 UF2 升级](#)

## 5.1.8 自供电 USB 设备解决方案

按照 USB 协议要求, USB 自供电设备必须通过检测 5V VBUS 电压来判断设备是否拔出, 进而实现热插拔。对于主机供电设备, 由于主机 VBUS 断电之后, 设备直接掉电关机, 无需实现该逻辑。

USB 设备 VBUS 检测方法一般有两种方法: 由 USB PHY 硬件检测, 或借助 ADC/GPIO 由软件检测。

由于 ESP32S2/S3 内部 USB PHY 不支持硬件检测逻辑, 该功能需要借助 ADC/GPIO 由软件实现, 其中使用 GPIO 检测方法最为简便, 实现方法如下:

### 对于 ESP-IDF 4.4 及更早版本:

1. 硬件上, 需要额外占用一个 IO (任意指定, 特殊引脚除外), 通过两个电阻分压 (例如两个 100K $\Omega$ ) 与 ESP32S2/S3 相连 (ESP32S2/S3 IO 最大可输入电压为 3.3v);



2. 在 `tinyusb_driver_install` 之后, 需要调用 `usbd_vbus_detect_gpio_enable` 函数使能 VBUS 检测, 该函数实现代码如下, 请直接复制到需要调用的位置:

```
/**
 *
 * @brief For USB Self-power device, the VBUS voltage must be monitored to achieve
 *hot-plug,
 * The simplest solution is detecting GPIO level as voltage signal.
 * A divider resistance Must be used due to ESP32S2/S3 has no 5V tolerate
 *pin.
 *
 * 5V VBUS ???????? ???????? GND
 * ???????? 100K ???????? 100K ????????
 * ???????? ? ????????
 * ? GPIOX
 * ?????????????????
 * The API Must be Called after tinyusb_driver_install to overwrite the
 *default config.
 * @param gpio_num, The gpio number used for vbus detect
 *
 */
static void usbd_vbus_detect_gpio_enable(int gpio_num)
{
    gpio_config_t io_conf = {
        .intr_type = GPIO_INTR_DISABLE,
        .pin_bit_mask = (1ULL<<gpio_num),
        //set as input mode
        .mode = GPIO_MODE_INPUT,
        .pull_up_en = 0,
        .pull_down_en = 0,
    };
    gpio_config(&io_conf);
    esp_rom_gpio_connect_in_signal(gpio_num, USB_OTG_VBUSVALID_IN_IDX, 0);
    esp_rom_gpio_connect_in_signal(gpio_num, USB_SRP_BVALID_IN_IDX, 0);
    esp_rom_gpio_connect_in_signal(gpio_num, USB_SRP_SESEND_IN_IDX, 1);
    return;
}
```

对于 ESP-IDF 5.0 及以上版本:

- 同上, 硬件上需要额外占用一个 IO (任意指定, 特殊引脚除外), 通过两个电阻分压 (例如两个 100K $\Omega$ ) 与 ESP32S2/S3 相连;
- 将用于检测 VBUS 的 IO 初始化为 GPIO 输入模式;
- 直接将 IO 配置到 `tinyusb_config_t` 中 (详情可参考):

```
#define VBUS_MONITORING_GPIO_NUM GPIO_NUM_4
// Configure GPIO Pin for vbus monitoring
const gpio_config_t vbus_gpio_config = {
    .pin_bit_mask = BIT64(VBUS_MONITORING_GPIO_NUM),
    .mode = GPIO_MODE_INPUT,
    .intr_type = GPIO_INTR_DISABLE,
    .pull_up_en = false,
    .pull_down_en = false,
};
ESP_ERROR_CHECK(gpio_config(&vbus_gpio_config));
const tinyusb_config_t tusb_cfg = {
    .device_descriptor = &descriptor_config,
    .string_descriptor = string_desc_arr,
    .string_descriptor_count = sizeof(string_desc_arr) / sizeof(string_desc_
 *arr[0]),
    .external_phy = false,
    .configuration_descriptor = desc_configuration,
    .self_powered = true,
```

(下页继续)

(续上页)

```
.vbus_monitor_io = VBUS_MONITORING_GPIO_NUM,
};
ESP_ERROR_CHECK(tinyusb_driver_install(&tusb_cfg));
```

### 5.1.9 阻止 Windows 依据 USB 设备序列号递增 COM 编号

由于任何连接到 Windows PC 的设备都通过其 VID、PID 和 Serial 号进行识别。如果这 3 个参数中的任何一个发生了变化，那么 PC 将检测到新的硬件，并与该设备关联一个不同的 COM 端口，详情请参考 [Windows USB device registry entries](#)。

ESP ROM Code 中对 USB 描述符配置如下：

	ESP32S2	ESP32S3	ESP32C3
<b>VID</b>	0x303a	0x303a	0x303a
<b>PID</b>	0x0002	0x1001	0x1001
<b>Serial</b>	0	MAC 地址字符串	MAC 地址字符串

- ESP32S2 (usb-otg) Serial 为常量 0，每个设备相同，COM 号一致
- ESP32S3 (usb-serial-jtag)、ESP32C3 (usb-serial-jtag) Serial 为设备 MAC 地址，每个设备均不同，COM 号默认递增

递增 COM 编号，为量产烧录带来额外工作。对于需要使用 USB 进行固件下载的客户，建议修改 Windows 的递增 COM 编号的规则，阻止依据 Serial 号递增编号。

#### 解决方案

管理员方式打开 Windows CMD，执行以下指令。该指令将添加注册表项，阻止依据 Serial 号递增编号，设置完成后请重启电脑使能修改：

```
REG ADD HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\usbflags\303A10010101 /
↪V IgnoreHWSerNum /t REG_BINARY /d 01
```

用户也可下载 ignore\_hwserial\_esp32s3c3.bat 脚本，右键选择管理员方式运行。

### 5.1.10 TinyUSB 应用指南

本指南包含以下内容：

#### 目录

- *TinyUSB* 简介
  - 芯片选型
- *TinyUSB* 组件
  - esp\_tinyusb 组件
  - espressif/tinyusb 组件
- *USB Device* 介绍
  - USB 音频 (UAC)
  - USB 视频 (UVC)

## TinyUSB 简介

TinyUSB 是一个开源的嵌入式 USB 主机/设备栈库，主要用于支持小型微控制器上的 USB 功能。它由 Adafruit 开发和维护，旨在提供一个轻量级、跨平台、易于集成的 USB 协议栈。TinyUSB 支持多种 USB 设备类型，包括 HID（人机接口设备）、MSC（大容量存储）、CDC（通信设备类）、MIDI（音乐设备数字接口）等，适用于各种嵌入式系统和物联网设备。基于原生的 TinyUSB 封装了以下的组件。

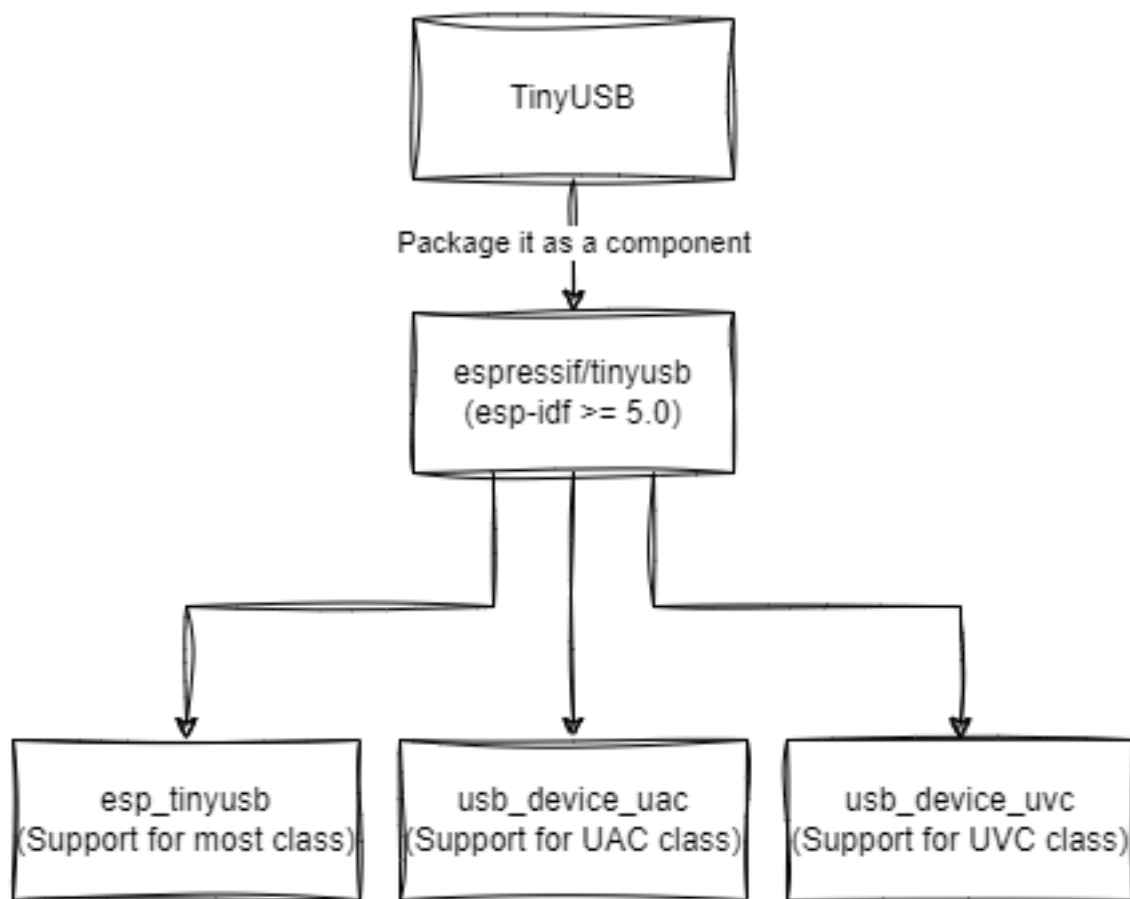


图 1: TinyUSB Components

## 芯片选型

Soc	USB1.1 Full Speed	USB2.0 High Speed
ESP32-S2	Supported	
ESP32-S3	Supported	
ESP32-P4	Supported	Supported

## TinyUSB 组件

**esp\_tinyusb 组件** `esp_tinyusb` 组件封装了一系列的 TinyUSB API，可以方便地集成 USB CDC-ACM, MSC, MIDI, HID, DFU, ECM/NCM/RNDIS 类在自己的工程中。

如何使用

- 在工程目录下运行 `idf.py add-dependency esp_tinyusb~1.0.0` 添加对 `esp_tinyusb` 库的依赖。

- 在 menuconfig 中配置需要使用的 USB 类。
- 在工程中使用 esp\_tinyusb API。

esp\_tinyusb 应用示例 提供了 U 盘，串口，HID 设备，复合设备等示例

- USB 串口和 U 盘复合设备示例
- USB 串口设备示例
- 通过 USB 输出 LOG 示例
- USB HID 设备示例
- USB MIDI 音乐设备示例
- USB U 盘设备示例
- USB 网卡设备示例

---

**备注：** c 库封装了许多 USB 类，使得开发一些被 esp\_tinyusb 支持的 USB 类非常容易，值得注意的是，这也使得进行某些改动很困难。适合于简单的 USB 应用。

---

**espressif/tinyusb 组件** 组件 espressif/tinyusb 是基于原生 tinyusb 仓库的一个组件，主要是将 tinyusb 仓库中的代码封装成一个组件，方便用户在自己的工程中使用。

---

**备注：** 该组件需要使用 ESP-IDF release/v5.0 及以上的版本。

---

如何使用：

- 在工程目录下运行 `idf.py add-dependency "tinyusb~0.15.10"` 添加对 espressif/tinyusb 库的依赖。
- 编写 `tusb_config.h` 文件，该文件通过定义一系列宏来决定 TinyUSB 的配置，并反向提供给 espressif/tinyusb 组件，同时在 main 组件的 CMakeLists.txt 添加以下代码：

```
idf_component_get_property(tusb_lib espressif__tinyusb COMPONENT_LIB)
target_include_directories(${tusb_lib} PRIVATE path_to_your_tusb_
↪config)
```

- 编写 `usb_descriptor.h` 文件，该文件定义了 USB 设备的描述符，包括设备描述符，配置描述符，接口描述符等。
- 编写 `usb_descriptors.c` 文件，该文件用于给 tinyusb 提供 USB 描述符回调‘

espressif/tinyusb 应用示例：

- USB 键盘鼠标设备示例
- USB 无线 U 盘示例
- Windows Surface Dial HID 示例
- 串口转 USB 示例

---

**备注：** espressif/tinyusb 库提供了更多的灵活性，可以更加方便的定制 USB 设备，适合于复杂的 USB 应用。在 idf release/v4.4 上可以使用 组件 `leeebo/tinyusb_src`，该组件与 espressif/tinyusb 作用相同。主要是补全了 espressif/tinyusb 对 ESP-IDF release/v4.4 的支持。

---

## USB Device 介绍

**USB 音频 (UAC)** TinyUSB 支持 USB UAC2.0 标准，用于通过 USB 传输音频数据。主要有以下特点。

- 最高支持 32 位/384kHz 的音频流
- 兼容 USB1.1 Full Speed 和 USB2.0 High Speed
- 延迟更低

**传输方式：** UAC 仅支持 USB 传输中的同步传输，所以 UAC 音频设备的数据端点都是同步端点。因为同步传输不进行重传，并且低延迟。同时因为主机和从机之间的传输是不同步的，可能会产生短暂静音/爆音，由此产生了三种同步方式。

- **SYNC 同步** 将输出时钟于每个 Frame 的 SOF 包同步
- **自适应** 根据主机传输数据的速率调整输出的采样率
- **ASYN 异步** 相比另外两种多了反馈端口，从机通过主机当前的速率来告知主机后续的发送速率，从而完成数据的补发或少发。从而不需要再适应主机的发送频率。

**关于 ASYN 异步传输的反馈端点** 通过启用宏 `CFG_TUD_AUDIO_ENABLE_FEEDBACK_EP` 来实现反馈速率的计算，TinyUSB 提供了多种反馈的数据计算，其中基于 FIFO 的反馈计算 (`AUDIO_FEEDBACK_METHOD_FIFO_COUNT`) 最为简单且实用。需要实现下面的虚函数，完成设置。

```
void tud_audio_feedback_params_cb(uint8_t func_id, uint8_t alt_itf, audio_feedback_
↪params_t* feedback_param)
{
    (void) func_id;
    (void) alt_itf;
    // Set feedback method to fifo counting
    feedback_param->method = AUDIO_FEEDBACK_METHOD_FIFO_COUNT;
    feedback_param->sample_freq = s_uac_device->current_sample_rate;

    ESP_LOGD(TAG, "Feedback method: %d, sample freq: %d", feedback_param->method, ↪
↪feedback_param->sample_freq);
}
```

工作原理是，UAC Class 内部维护了一块软件 FIFO 大小为 `CFG_TUD_AUDIO_FUNC_1_EP_OUT_SW_BUF_SZ`，通过将此内存大小设置为 10ms 的数据大小，让 UAC 驱动内部有一块缓冲区，驱动会通过反馈端点将 FIFO 的水位维持在二分之一大小，当数据缺失，主机会一包中多发数据，当数据缺少，主机会少发数据。

应用上建议，首先在每一次新音频传输的时候（例大于 100ms 没有数据到来，则认为是一个新音频）先让 UAC 内部 fifo 缓冲一半缓冲区大小的数据，再开始播放，这样可以保证 I2S 一直有数据可以取，不会产生爆破音和噪声。同时基于反馈端点，软件 FIFO 的大小会一直维持在一个稳定的水平。

具体可参考[ESP Device UAC](#)

**USB 视频 (UVC)** TinyUSB 支持 USB UVC1.5 标准，用于通过 USB 传输视频数据，能够传输多种视频格式，包括未压缩格式 YUV 格式，压缩格式 MJPEG，H264，H265 等

### 传输方式：

- 当视频流接口 (USB Video streaming) 传输视频的时候，其传输端点为同步传输或者批量传输端点。
- 当视频流接口传输静态图像的时候，其传输类型为批量传输端点。

**传输图像：** UVC 能够传输多种视频格式，这些图像格式是通过视频描述符的 Format 和 Frame 来定义的。

图像类型	Format	Frame
MJPEG	FORMAT_MJPEG: 0x06	FRAME_MJPEG: 0x07
YUV2/NV12/MJPEG	FORMAT_UNCOMPRESSED: 0x04	FRAME_UNCOMPRESSED: 0x05
H264	FORMAT_H264: 0x013	FRAME_H264: 0x014
H265	FORMAT_FRAME_BASED: 0x10	FRAME_FRAME_BASED: 0x11

其中 Frame based 格式比较特殊，可以存储任意的图像格式，只要图像是按照帧来存储的。如 MJPG，H264，H265 等。通过 GUID 字段来表示具体的图像格式。

**双摄像头** UVC 设备中，一个物理摄像头会有一个 VC（video control）描述符，而一个 VC 描述符可以有多个 VS（video streaming）描述符。表示这个摄像头可以传输多种格式图像。但是在一些特殊的硬件中，会有两个硬件摄像头，这时就需要有两个 VC 描述符。

```
USB Descriptor
|
|-- Video Control
|   |-- Video Streaming
|
|-- Video Control
|   |-- Video Streaming
```

具体可参考[USB Device UVC](#)

### 5.1.11 使用原生的 tinyusb 进行开发

本指南包含以下内容：

#### 目录

- [工程目录](#)
- [tusb\\_config.h](#) 文件
- [usb\\_descriptors.h](#) 文件（可选）
- [usb\\_descriptors.c](#) 文件
- [初始化 USB Phy](#)
- [初始化 TinyUSB 协议栈](#)
- [实现设备层的弱函数](#)
- [实现 USB Class 的特殊的回调函数。](#)

本篇将会介绍如何使用 tinyusb 的组件进行开发。

#### 工程目录

首先需要建立以下目录结构：

```
project_name
|
|-- main
|   |-- CMakeLists.txt
|   |-- idf_component.yml
|   |-- main.c
|
|-- tusb
|   |-- tusb_config.h
|   |-- usb_descriptors.c
|   |-- usb_descriptors.h
```

在该工程的 main 组件中添加组件依赖，[espressif/tinyusb](#)。

tusb 文件夹主要放置用于反向提供给 tinyusb 的文件，通过单独放置一个文件夹中，保证依赖关系的简单。然后需要在 main/CMakeLists.txt 文件中添加以下语句（放置于 idf\_component\_register 之后）

```
# espressif__tinyusb 应匹配当前依赖的 tinyusb 名称
idf_component_get_property(tusb_lib espressif__tinyusb COMPONENT_LIB)

target_include_directories(${tusb_lib} PUBLIC "${COMPONENT_DIR}/tusb")
target_sources(${tusb_lib} PUBLIC "${COMPONENT_DIR}/tusb/usb_descriptors.c")
```

**备注：**关于反向依赖问题，因为工程需要依赖 tinyusb，有需要向 tinyusb 提供头文件，所以不可避免的会导致反向依赖的问题。目前可以通过将 tinyusb 的全部关键文件作为源码直接编译到 main 组件中来解决。

### tusb\_config.h 文件

tinyusb 大部分的功能的启用和关闭都是通过宏来控制，所以需要在 tusb\_config.h 文件中声明所需要的功能。下面列出一些关键的宏：

#### 系统设置的宏：

- **CFG\_TUSB\_RHPORT0\_MODE**：用于定义连接到 USB Phy 的方式和速率，下面的方式表示是 USB device 设备，并且速率为 USB 全速。

```
#define CFG_TUSB_RHPORT0_MODE (OPT_MODE_DEVICE | OPT_MODE_FULL_
↪SPEED)
```

- **ESP\_PLATFORM**：使用 esp-idf 平台进行编译，需启用该宏。
- **CFG\_TUSB\_OS**：用于定义 tinyusb 的操作系统，如果使用的是 FreeRTOS，需要启用该宏。也可以不启用操作系统

```
#define CFG_TUSB_OS OPT_OS_FREERTOS
```

- **CFG\_TUSB\_OS\_INC\_PATH**：在 ESP-IDF 中要求需要添加 “freertos/” 前缀在 include 路径中。

```
#define CFG_TUSB_OS_INC_PATH freertos/
```

- **CFG\_TUSB\_DEBUG**：用于启用 tinyusb 的 LOG 打印等级。总共三级

```
#define CFG_TUSB_DEBUG 0
```

- **CFG\_TUD\_ENABLED**：设为 1 启用 tinyusb device 功能。
- **CFG\_TUSB\_MEM\_SECTION**：通过启用该宏，可以将 tinyusb 的内存分配到特定的内存段中。
- **CFG\_TUSB\_MEM\_ALIGN**：用于定义内存对齐方式。

```
#define CFG_TUSB_MEM_ALIGN __attribute__((aligned(4)))
```

#### USB 设备的宏：

- **CFG\_TUD\_ENDPOINT0\_SIZE**：用于定义端点 0 的最大包大小。

#### USB Class 的宏：

这里以 UVC Class 举例，每一个 USB Class 都有单独的宏定义：

- **CFG\_TUD\_VIDEO**：配置视频控制接口（video control interface）的数量
- **CFG\_TUD\_VIDEO\_STREAMING**：配置视频流接口（video streaming interface）的数量

可以参考以下文件示例：

- [../components/usb/usb\\_device\\_uac/tusb/tusb\\_config.h](#)
- [../components/usb/usb\\_device\\_uvc/tusb/tusb\\_config.h](#)
- [/usb/device/usb\\_hid\\_device/hid\\_device/tusb\\_config.h](#)

### usb\_descriptors.h 文件（可选）

该文件主要用来放置自定义的 USB 描述符。tinyusb 提供了很多描述符的模板，如果不满足需求，就需要自己定义一套 USB 描述符。需要注意的是尽量使用 tinyusb 中预定义好的一些描述符，这样可以很方便的进行描述符组装和计算长度。

可以参考以下文件示例：

- [../components/usb/usb\\_device\\_uac/tusb/uac\\_descriptors.h](#)
- [../components/usb/usb\\_device\\_uvc/tusb/usb\\_descriptors.h](#)



- [/usb/device/usb\\_hid\\_device/hid\\_device/usb\\_descriptors.h](#)

### usb\_descriptors.c 文件

该文件主要实现了几个获取描述符的弱函数，分别是获取设备描述符，或者配置描述符和获取字符串描述符。

```
uint8_t const *tud_descriptor_device_cb(void);

uint8_t const *tud_descriptor_configuration_cb(uint8_t index);

uint16_t const *tud_descriptor_string_cb(uint8_t index, uint16_t langid);
```

注意点：

- 配置描述符的长度一定要等于实际的长度
- 配置描述符使用的各个端点描述符的端点号要避免重复

可以参考以下文件示例：

- [../components/usb/usb\\_device\\_uvc/tusb/usb\\_descriptors.c](#)
- [../components/usb/usb\\_device\\_uac/tusb/usb\\_descriptors.c](#)
- [/usb/device/usb\\_hid\\_device/hid\\_device/usb\\_descriptors.c](#)

### 初始化 USB Phy

初始化内部 USB Phy:

```
static void usb_phy_init(void)
{
    // Configure USB PHY
    usb_phy_config_t phy_conf = {
        .controller = USB_PHY_CTRL_OTG,
        .otg_mode = USB_OTG_MODE_DEVICE,
        .target = USB_PHY_TARGET_INT,
    };
    usb_new_phy(&phy_conf, &s_uvc_device.phy_hdl);
}
```

如果使用外部 USB Phy，参考[使用外部 PHY](#)

### 初始化 TinyUSB 协议栈

使用以下的代码

```
static void tusb_device_task(void *arg)
{
    while (1) {
        tud_task();
    }
}

int main(void) {
    usb_phy_init();
    bool usb_init = tusb_init();
    if (!usb_init) {
        ESP_LOGE(TAG, "USB Device Stack Init Fail");
        return ESP_FAIL;
    }
}
```

(下页继续)



(续上页)

```
xTaskCreatePinnedToCore(tusb_device_task, "TinyUSB", 4096, NULL, 5, NULL, 0);
}
```

## 实现设备层的弱函数

可以获取设备的插入，拔出，暂停，恢复等事件。

```
// Invoked when device is mounted
void tud_mount_cb(void)
{
}

// Invoked when device is unmounted
void tud_umount_cb(void)
{
}

// Invoked when device is suspended
void tud_suspend_cb(bool remote_wakeup_en)
{
}

// Invoked when usb bus is resumed
void tud_resume_cb(void)
{
}
```

## 实现 USB Class 的特殊的回调函数。

USB Class 提供了一些弱函数来完成基本的功能，接下来会以 UVC 驱动为例。源码文件 [video device](#) 通过观察 API 可以发现 UVC Class 提供了两个函数和一个回调函数，

```
bool tud_video_n_streaming(uint_fast8_t ctl_idx, uint_fast8_t stm_idx);

bool tud_video_n_frame_xfer(uint_fast8_t ctl_idx, uint_fast8_t stm_idx, void_
↪*buffer, size_t bufsize);

TU_ATTR_WEAK void tud_video_frame_xfer_complete_cb(uint_fast8_t ctl_idx, uint_
↪fast8_t stm_idx);
```

通过调用 `tud_video_n_frame_xfer` 函数来传输一帧图像，并通过 `tud_video_frame_xfer_complete_cb` 来检查是否传输完成。

此外不同的 USB Class 还会有一些特殊的宏定义，用于定义软件 fifo 大小或者启用一些功能。比如 UVC Class 中的宏 `CFG_TUD_VIDEO_STREAMING_EP_BUFSIZE` 用于定义视频传输流（video streaming interface）端点的 buffer 的大小。

## 5.2 USB 主机驱动

### 5.2.1 USB Stream 组件说明

usb\_stream 是基于 ESP32-S2/ESP32-S3 的 USB UVC + UAC 主机驱动程序，支持从 USB 设备读取/写入/控制多媒体流。例如最多同时支持 1 路摄像头 + 1 路麦克风 + 1 路播放器数据流。

特性：

1. 支持通过 UVC Stream 接口获取视频流，支持批量和同步两种传输模式
2. 支持通过 UAC Stream 接口获取麦克风数据流，发送播放器数据流
3. 支持通过 UAC Control 接口控制麦克风音量、静音等特性
4. 支持自动解析设备配置描述符
5. 支持对数据流暂停和恢复

## USB Stream 用户指南

- 开发板
  1. 可以使用任何带有 USB 接口的 ESP32-S2/ESP32-S3 开发板，注意该 USB 接口需要能够向外供电
- USB UVC 功能
  1. 摄像头必须兼容 USB1.1 全速 (Fullspeed) 模式
  2. 摄像头需要自带 MJPEG 压缩
  3. 用户可通过 `uvc_streaming_config()` 函数手动指定相机接口、传输模式和图像帧参数
  4. 如使用同步传输摄像头，需要支持设置接口 MPS (Max Packet Size) 为 512
  5. 同步传输模式，图像数据流 USB 传输总带宽应小于 4 Mbps (500 KB/s)
  6. 批量传输模式，图像数据流 USB 传输总带宽应小于 8.8 Mbps (1100 KB/s)
  7. 其它特殊相机要求，请参考示例程序 README.md
- USB UAC 功能
  1. 音频功能必须兼容 UAC 1.0 协议
  2. 用户需要通过 `uac_streaming_config()` 函数手动指定 spk/mic 采样率，位宽参数
- USB UVC + UAC 功能
  1. UVC 和 UAC 功能可以单独启用，例如仅配置 UAC 来驱动一个 USB 耳机，或者仅配置 UVC 来驱动一个 USB 摄像头
  2. 如需同时启用 UVC + UAC，该驱动程序目前仅支持具有摄像头和音频接口的复合设备 (Composite Device)，不支持同时连接两个单独的设备

## USB Stream API 参考

1. 用户可通过 `uvc_config_t` 配置摄像头分辨率、帧率参数。通过 `uac_config_t` 配置音频采样率、位宽等参数。参数说明如下：

```
uvc_config_t uvc_config = {
    .frame_width = 320, // mjpeg width pixel, for example 320
    .frame_height = 240, // mjpeg height pixel, for example 240
    .frame_interval = FPS2INTERVAL(15), // frame interval (100μs units), such as ↪
    ↪ 15fps
    .xfer_buffer_size = 32 * 1024, // single frame image size, need to be ↪
    ↪ determined according to actual testing, 320 * 240 generally less than 35KB
    .xfer_buffer_a = pointer_buffer_a, // the internal transfer buffer
    .xfer_buffer_b = pointer_buffer_b, // the internal transfer buffer
    .frame_buffer_size = 32 * 1024, // single frame image size, need to determine ↪
    ↪ according to actual test
    .frame_buffer = pointer_frame_buffer, // the image frame buffer
    .frame_cb = &camera_frame_cb, //camera callback, can block in here
    .frame_cb_arg = NULL, // camera callback args
}

uac_config_t uac_config = {
    .mic_bit_resolution = 16, //microphone resolution, bits
    .mic_samples_frequency = 16000, //microphone frequency, Hz
    .spk_bit_resolution = 16, //speaker resolution, bits
    .spk_samples_frequency = 16000, //speaker frequency, Hz
```

(下页继续)

(续上页)

```

        .spk_buf_size = 16000, //size of speaker send buffer, should be a multiple of_
        ↪ spk_ep_mps
        .mic_buf_size = 0, //mic receive buffer size, 0 if not use, else should be a_
        ↪ multiple of mic_min_bytes
        .mic_cb = &mic_frame_cb, //mic callback, can not block in here
        .mic_cb_arg = NULL, //mic callback args
    };

```

2. 使用 `uvc_streaming_config()` 配置 UVC 驱动，如果设备同时支持音频，可使用 `uac_streaming_config()` 配置 UAC 驱动
3. 使用 `usb_streaming_start()` 打开数据流，之后驱动将响应设备连接和协议协商
4. 之后，主机将根据用户参数配置，匹配已连接设备的描述符，如果设备无法满足配置要求，驱动将以警告提示
5. 如果设备满足用户配置要求，主机将持续接收 IN 数据流（UVC 和 UAC 麦克风），并在新帧准备就绪后调用用户的回调
  1. 接收到新的 MJPEG 图像后，将触发 UVC 回调函数。用户可在回调函数中阻塞，因为它在独立的任务上下文中工作
  2. 接收到 mic\_min\_bytes 字节数据后，将触发 mic 回调。但是这里的回调一定不能以任何方式阻塞，否则会影响下一帧的接收。如果需要对 mic 进行阻塞操作，可以通过 `uac_mic_streaming_read` 轮询方式代替回调方式
6. 发送扬声器数据时，用户可通过 `uac_spk_streaming_write()` 将数据写入内部 ringbuffer，主机将在 USB 空闲时从中获取数据并发送 OUT 数据
7. 使用 `usb_streaming_control()` 控制流挂起/恢复。如果 UAC 支持特性单元，可通过其分别控制麦克风和播放器的音量和静音
8. 使用 `usb_streaming_stop()` 停止 USB 流，USB 资源将被完全释放。

## Bug 报告

### ESP32-S2 ECO0 芯片 SPI 屏幕和 USB 同时启用，可能导致屏幕抖动

1. 在最早版本的 ESP32-S2 ECO0 芯片上，USB 可能污染 SPI 数据 (ESP32-S2 新版本 >=ECO1 和 ESP32-S3 均不存在该 Bug)
2. 软件解决方案
  - spi\_ll.h 添加以下接口

```

//components/hal/esp32s2/include/hal/spi_ll.h
static inline uint32_t spi_ll_tx_get_fifo_cnt(spi_dev_t *hw)
{
    return hw->dma_out_status.out_fifo_cnt;
}

```

- 如下在 `spi_new_trans` 中添加额外检查

```

// The function is called to send a new transaction, in ISR or in the task.
// Setup the transaction-specified registers and linked-list used by the DMA (or_
    ↪ FIFO if DMA is not used)
static void SPI_MASTER_ISR_ATTR spi_new_trans(spi_device_t *dev, spi_trans_priv_t_
    ↪ *trans_buf)
{
    //.....
    spi_hal_setup_trans(hal, hal_dev, &hal_trans);
    spi_hal_prepare_data(hal, hal_dev, &hal_trans);

    //Call pre-transmission callback, if any
    if (dev->cfg.pre_cb) dev->cfg.pre_cb(trans);
#ifdef 1
    //USB Bug workaround
    //while (((spi_ll_tx_get_fifo_cnt(SPI_LL_GET_HW(host->id)) == 12) || (spi_ll_
    ↪ tx_get_fifo_cnt(SPI_LL_GET_HW(host->id)) == trans->length / 8))) {

```

(下页继续)

(续上页)

```

    while (trans->length && spi_ll_tx_get_fifo_cnt(SPI_LL_GET_HW(host->id)) == 0) {
        __asm__ __volatile__ ("nop");
        __asm__ __volatile__ ("nop");
        __asm__ __volatile__ ("nop");
    }
#endif
    //Kick off transfer
    spi_hal_user_start(hal);
}

```

## Examples

1. [usb/host/usb\\_camera\\_mic\\_spk](#)
2. [usb/host/usb\\_camera\\_lcd\\_display](#)
3. [usb/host/usb\\_audio\\_player](#)

## API Reference

### Header File

- [components/usb/usb\\_stream/include/usb\\_stream.h](#)

### Functions

`esp_err_t uvc_streaming_config` (const [uvc\\_config\\_t](#) \*config)

Config UVC streaming with user defined parameters. For normal use, user only need to specify no-optional parameters, and set optional parameters to 0 (the driver will find the correct value from the device descriptors). For quick start mode, user should specify all parameters manually to skip get and process descriptors steps.

**参数** `config` -parameters defined in `uvc_config_t`

**返回** `esp_err_t` `ESP_ERR_INVALID_STATE` USB streaming is running, user need to stop streaming first `ESP_ERR_INVALID_ARG` Invalid argument `ESP_OK` Success

`esp_err_t uac_streaming_config` (const [uac\\_config\\_t](#) \*config)

Config UAC streaming with user defined parameters. For normal use, user only need to specify no-optional parameters, and set optional parameters to 0 (the driver will find the correct value from the device descriptors). For quick start mode, user should specify all parameters manually to skip get and process descriptors steps.

**参数** `config` -parameters defined in `uvc_config_t`

**返回** `esp_err_t` `ESP_ERR_INVALID_STATE` USB streaming is running, user need to stop streaming first `ESP_ERR_INVALID_ARG` Invalid argument `ESP_OK` Success

`esp_err_t usb_streaming_start` (void)

Start usb streaming with pre-configs, usb driver will create internal tasks to handle usb data from stream pipe, and run user's callback after new frame ready.

**返回** `ESP_ERR_INVALID_STATE` streaming not configured, or streaming running already `ESP_FAIL` start failed `ESP_OK` start succeed

`esp_err_t usb_streaming_stop` (void)

Stop current usb streaming, internal tasks will be delete, related resource will be free.

**返回** `ESP_ERR_INVALID_STATE` streaming not started `ESP_ERR_TIMEOUT` stop wait timeout `ESP_OK` stop succeed

`esp_err_t usb_streaming_connect_wait` (size\_t timeout\_ms)

Wait for USB device connection.

**参数** `timeout_ms` -timeout in ms

返回 `esp_err_t ESP_ERR_INVALID_STATE`: usb streaming not started `ESP_ERR_TIMEOUT`: timeout `ESP_OK`: device connected

`esp_err_t usb_streaming_state_register` (*state\_callback\_t* cb, void \*user\_ptr)

This function registers a callback for USB streaming, please note that only one callback can be registered, the later registered callback will overwrite the previous one.

#### 参数

- **cb** –A pointer to a function that will be called when the USB streaming state changes.
- **user\_ptr** –user\_ptr is a void pointer.

返回 `esp_err_t`

- `ESP_OK` Success
- `ESP_ERR_INVALID_STATE` USB streaming is running, callback need register before start

`esp_err_t usb_streaming_control` (*usb\_stream\_t* stream, *stream\_ctrl\_t* ctrl\_type, void \*ctrl\_value)

Control USB streaming with specific stream and control type.

#### 参数

- **stream** –stream type defined in `usb_stream_t`
- **ctrl\_type** –control type defined in `stream_ctrl_t`
- **ctrl\_value** –control value

返回 `ESP_ERR_INVALID_ARG` invalid arg `ESP_ERR_INVALID_STATE` driver not configured or not started `ESP_ERR_NOT_SUPPORTED` current device not support this control type `ESP_FAIL` control failed `ESP_OK` succeed

`esp_err_t uac_spk_streaming_write` (void \*data, size\_t data\_bytes, size\_t timeout\_ms)

Write data to the speaker buffer, will be send out when USB device is ready.

#### 参数

- **data** –The data to be written.
- **data\_bytes** –The size of the data to be written.
- **timeout\_ms** –The timeout value for writing data to the buffer.

返回 `ESP_ERR_INVALID_STATE` spk stream not config `ESP_ERR_NOT_FOUND` spk interface not found `ESP_ERR_TIMEOUT` spk ringbuf full `ESP_OK` succeed

`esp_err_t uac_mic_streaming_read` (void \*buf, size\_t buf\_size, size\_t \*data\_bytes, size\_t timeout\_ms)

Read data from internal mic buffer, the actual size will be returned.

#### 参数

- **buf** –pointer to the buffer to store the received data
- **buf\_size** –The size of the data buffer.
- **data\_bytes** –The actual size read from buffer
- **timeout\_ms** –The timeout value for the read operation.

返回 `ESP_ERR_INVALID_ARG` parameter error `ESP_ERR_INVALID_STATE` mic stream not config `ESP_ERR_NOT_FOUND` mic interface not found `ESP_TIMEOUT` timeout `ESP_OK` succeed

`esp_err_t uac_frame_size_list_get` (*usb\_stream\_t* stream, *uac\_frame\_size\_t* \*frame\_list, size\_t \*list\_size, size\_t \*cur\_index)

Get the audio frame size list of current stream, the list contains audio channel number, bit resolution and samples frequency. IF list\_size equals 1 and the samples\_frequence equals 0, which means the frequency can be set to any value between samples\_frequence\_min and samples\_frequence\_max.

#### 参数

- **stream** –the stream type
- **frame\_list** –the output frame list, NULL to only get the list size
- **list\_size** –frame list size
- **cur\_index** –current frame index

返回 `esp_err_t`

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_ERR_INVALID_STATE` USB device not active

- ESP\_OK Success

esp\_err\_t **uac\_frame\_size\_reset** (*usb\_stream\_t* stream, uint8\_t ch\_num, uint16\_t bit\_resolution, uint32\_t samples\_frequency)

Reset audio channel number, bit resolution and samples frequency, please reset when the streaming in suspend state. The new configs will be effective after streaming resume.

#### 参数

- **stream** –stream type
- **ch\_num** –audio channel numbers
- **bit\_resolution** –audio bit resolution
- **samples\_frequency** –audio samples frequency

返回 esp\_err\_t

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_INVALID\_STATE USB device not active
- ESP\_ERR\_NOT\_FOUND frequency not found
- ESP\_OK Success
- ESP\_FAIL Reset failed

esp\_err\_t **uvc\_frame\_size\_list\_get** (*uvc\_frame\_size\_t* \*frame\_list, size\_t \*list\_size, size\_t \*cur\_index)

Get the frame size list of current connected camera.

#### 参数

- **frame\_list** –the frame size list, can be NULL if only want to get list size
- **list\_size** –the list size
- **cur\_index** –current frame index

返回 esp\_err\_t ESP\_ERR\_INVALID\_ARG parameter error ESP\_ERR\_INVALID\_STATE uvc stream not config or not active ESP\_OK succeed

esp\_err\_t **uvc\_frame\_size\_reset** (uint16\_t frame\_width, uint16\_t frame\_height, uint32\_t frame\_interval)

Reset the expected frame size and frame interval, please reset when uvc streaming in suspend state. The new configs will be effective after streaming resume.

Note: frame\_width and frame\_height can be set to 0 at the same time, which means no change on frame size.

#### 参数

- **frame\_width** –frame width, FRAME\_RESOLUTION\_ANY means any width
- **frame\_height** –frame height, FRAME\_RESOLUTION\_ANY means any height
- **frame\_interval** –frame interval, 0 means no change

返回 esp\_err\_t

## Structures

struct **uvc\_config**

UVC configurations, for params with (optional) label, users do not need to specify manually, unless there is a problem with descriptors, or users want to skip the get and process descriptors steps.

## Public Members

uint16\_t **frame\_width**

Picture width, set FRAME\_RESOLUTION\_ANY for any resolution

uint16\_t **frame\_height**

Picture height, set FRAME\_RESOLUTION\_ANY for any resolution

uint32\_t **frame\_interval**

Frame interval in 100-ns units, 666666 ~ 15 Fps

uint32\_t **xfer\_buffer\_size**

Transfer buffer size, using double buffer here, must larger than one frame size

uint8\_t \***xfer\_buffer\_a**

Buffer a for usb payload

uint8\_t \***xfer\_buffer\_b**

Buffer b for usb payload

uint32\_t **frame\_buffer\_size**

Frame buffer size, must larger than one frame size

uint8\_t \***frame\_buffer**

Buffer for one frame

uvc\_frame\_callback\_t **frame\_cb**

callback function to handle incoming frame

void \***frame\_cb\_arg**

callback function arg

*uvc\_format\_t* **format**

(optional) UVC stream format, default using MJPEG Optional configs, Users need to specify parameters manually when they want to skip the get and process descriptors steps (used to speed up startup)

*uvc\_xfer\_t* **xfer\_type**

(optional) UVC stream transfer type, UVC\_XFER\_ISOC or UVC\_XFER\_BULK

uint8\_t **format\_index**

(optional) Format index

uint8\_t **frame\_index**

(optional) Frame index, to choose resolution

uint16\_t **interface**

(optional) UVC stream interface number

uint16\_t **interface\_alt**

(optional) UVC stream alternate interface, to choose MPS (Max Packet Size), bulk fix to 0

uint8\_t **ep\_addr**

(optional) endpoint address of selected alternate interface

uint32\_t **ep\_mps**

(optional) MPS of selected interface\_alt

uint32\_t **flags**

(optional) flags to control the driver behaviors

struct **mic\_frame\_t**  
mic frame type

#### **Public Members**

void \***data**  
mic data

uint32\_t **data\_bytes**  
mic data size

uint16\_t **bit\_resolution**  
bit resolution in buffer

uint32\_t **samples\_frequence**  
mic sample frequency

struct **uvc\_frame\_size\_t**  
uvc frame type

#### **Public Members**

uint16\_t **width**  
frame width

uint16\_t **height**  
frame height

uint32\_t **interval**  
frame interval

uint32\_t **interval\_min**  
frame min interval

uint32\_t **interval\_max**  
frame max interval

uint32\_t **interval\_step**  
frame interval step

struct **uac\_frame\_size\_t**  
uac frame type

#### **Public Members**



`uint8_t ch_num`

channel numbers

`uint16_t bit_resolution`

bit resolution in buffer

`uint32_t samples_frequency`

sample frequency

`uint32_t samples_frequency_min`

min sample frequency

`uint32_t samples_frequency_max`

max sample frequency

struct `uac_config_t`

UAC configurations, for params with (optional) label, users do not need to specify manually, unless there is a problem with descriptor parse, or a problem with the device descriptor.

### Public Members

`uint8_t spk_ch_num`

speaker channel numbers, UAC\_CH\_ANY for any channel number

`uint8_t mic_ch_num`

microphone channel numbers, UAC\_CH\_ANY for any channel number

`uint16_t mic_bit_resolution`

microphone resolution(bits), UAC\_BITS\_ANY for any bit resolution

`uint32_t mic_samples_frequency`

microphone frequency(Hz), UAC\_FREQUENCY\_ANY for any frequency

`uint16_t spk_bit_resolution`

speaker resolution(bits), UAC\_BITS\_ANY for any

`uint32_t spk_samples_frequency`

speaker frequency(Hz), UAC\_FREQUENCY\_ANY for any frequency

`uint32_t spk_buf_size`

size of speaker send buffer, should be a multiple of spk\_ep\_mps

`uint32_t mic_buf_size`

mic receive buffer size, 0 if not use

`mic_callback_t mic_cb`

mic callback, can not block in here!, NULL if not use

void **\*mic\_cb\_arg**

mic callback args, NULL if not use Optional configs, Users need to specify parameters manually when they want to skip the get and process descriptors steps (used to speed up startup)

uint16\_t **mic\_interface**

(optional) microphone stream interface number, set 0 if not use

uint8\_t **mic\_ep\_addr**

(optional) microphone interface endpoint address

uint32\_t **mic\_ep\_mps**

(optional) microphone interface endpoint mps

uint16\_t **spk\_interface**

(optional) speaker stream interface number, set 0 if not use

uint8\_t **spk\_ep\_addr**

(optional) speaker interface endpoint address

uint32\_t **spk\_ep\_mps**

(optional) speaker interface endpoint mps

uint16\_t **ac\_interface**

(optional) audio control interface number, set 0 if not use

uint8\_t **mic\_fu\_id**

(optional) microphone feature unit id, set 0 if not use

uint8\_t **spk\_fu\_id**

(optional) speaker feature unit id, set 0 if not use

uint32\_t **flags**

(optional) flags to control the driver behaviors

## Macros

**FRAME\_RESOLUTION\_ANY**

any uvc frame resolution

**UAC\_FREQUENCY\_ANY**

any uac sample frequency

**UAC\_BITS\_ANY**

any uac bit resolution

**UAC\_CH\_ANY**

any uac channel number

**FPS2INTERVAL** (fps)

convert fps to uvc frame interval

**FRAME\_INTERVAL\_FPS\_5**

5 fps

**FRAME\_INTERVAL\_FPS\_10**

10 fps

**FRAME\_INTERVAL\_FPS\_15**

15 fps

**FRAME\_INTERVAL\_FPS\_20**

20 fps

**FRAME\_INTERVAL\_FPS\_30**

25 fps

**FLAG\_UVC\_SUSPEND\_AFTER\_START**

suspend uvc after usb\_streaming\_start

**FLAG\_UAC\_SPK\_SUSPEND\_AFTER\_START**

suspend uac speaker after usb\_streaming\_start

**FLAG\_UAC\_MIC\_SUSPEND\_AFTER\_START**

suspend uac microphone after usb\_streaming\_start

### Type Definitions

typedef struct *uvc\_config* **uvc\_config\_t**

UVC configurations, for params with (optional) label, users do not need to specify manually, unless there is a problem with descriptors, or users want to skip the get and process descriptors steps.

typedef void (\***mic\_callback\_t**)(*mic\_frame\_t* \*frame, void \*user\_ptr)

user callback function to handle incoming mic frames

typedef void (\***state\_callback\_t**)(*usb\_stream\_state\_t* state, void \*user\_ptr)

user callback function to handle usb device connection status

### Enumerations

enum **uvc\_xfer\_t**

UVC stream usb transfer type, most camera using isochronous mode, bulk mode can also be support for higher bandwidth.

*Values:*

enumerator **UVC\_XFER\_ISOC**

Isochronous Transfer Mode

enumerator **UVC\_XFER\_BULK**

Bulk Transfer Mode

enumerator **UVC\_XFER\_UNKNOWN**  
Unknown Mode

enum **usb\_stream\_t**  
UVC stream format type, default using MJPEG format.  
Stream id, used for control  
*Values:*

enumerator **STREAM\_UVC**  
usb video stream

enumerator **STREAM\_UAC\_SPK**  
usb audio speaker stream

enumerator **STREAM\_UAC\_MIC**  
usb audio microphone stream

enumerator **STREAM\_MAX**  
max stream id

enum **usb\_stream\_state\_t**  
USB device connection status.  
*Values:*

enumerator **STREAM\_CONNECTED**

enumerator **STREAM\_DISCONNECTED**

enum **stream\_ctrl\_t**  
Stream control type, which also depends on if device support.  
*Values:*

enumerator **CTRL\_NONE**  
None

enumerator **CTRL\_SUSPEND**  
streaming suspend control. ctrl\_data NULL

enumerator **CTRL\_RESUME**  
streaming resume control. ctrl\_data NULL

enumerator **CTRL\_UAC\_MUTE**  
mute control. ctrl\_data (false/true)

enumerator **CTRL\_UAC\_VOLUME**  
volume control. ctrl\_data (0~100)

enumerator **CTRL\_MAX**  
max type value

## 5.2.2 ESP MSC OTA

*esp\_msc\_ota* 是基于 USB MSC 的 OTA 驱动程序，它支持通过 USB 从 U 盘中读取程序，烧录到指定 OTA 分区，从而实现 OTA 升级的功能。

特性：

1. 支持通过 USB 接口读取 U 盘，并实现 OTA 升级
2. 支持 U 盘热插拔

### 用户指南

硬件需求：

- 任何带有 USB 接口的 ESP32-S2/ESP32-S3 开发板，且 USB 接口需要能够向外供电
- 使用 BOT（仅限大容量传输）协议和 Transparent SCSI 命令集的 U 盘。

分区表：

- 具有 OTA 分区

### 代码示例

1. 调用 *esp\_msc\_host\_install* 初始化 MSC 主机驱动程序

```
esp_msc_host_config_t msc_host_config = {
    .base_path = "/usb",
    .host_driver_config = DEFAULT_MSC_HOST_DRIVER_CONFIG(),
    .vfs_fat_mount_config = DEFAULT_ESP_VFS_FAT_MOUNT_CONFIG(),
    .host_config = DEFAULT_USB_HOST_CONFIG()
};
esp_msc_host_handle_t host_handle = NULL;
esp_msc_host_install(&msc_host_config, &host_handle);
```

2. 调用 *esp\_msc\_ota* 完成 OTA 升级。通过 *ota\_bin\_path* 指定 OTA 文件路径，通过 *wait\_msc\_connect* 指定等待 U 盘插入的时间，单位为 freertos tick。

```
esp_msc_ota_config_t config = {
    .ota_bin_path = "/usb/ota_test.bin",
    .wait_msc_connect = pdMS_TO_TICKS(5000),
};
esp_msc_ota(&config);
```

3. 调用 *esp\_event\_handler\_register* 注册事件处理程序，获取 ota 过程细节。

```
esp_event_loop_create_default();
esp_event_handler_register(ESP_MSC_OTA_EVENT, ESP_EVENT_ANY_ID, &event_handler,
↪↪↪NULL);
```

### API Reference

#### Header File

- [components/usb/esp\\_msc\\_ota/include/esp\\_msc\\_ota.h](#)

## Functions

`esp_err_t esp_msc_ota_begin (esp_msc_ota_config_t *config, esp_msc_ota_handle_t *handle)`

Start MSC OTA Firmware upgrade.

If this function succeeds, then call `esp_msc_ota_perform` to continue with the OTA process otherwise call `esp_msc_ota_end`.

### 参数

- **config** –[in] pointer to `esp_msc_ota_config_t` structure
- **handle** –[out] pointer to an allocated data of type `esp_msc_ota_handle_t` which will be initialised in this function

### 返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG: Invalid argument (missing/incorrect config, handle, etc.)
- ESP\_ERR\_NO\_MEM: Failed to allocate memory for msc\_ota handle
- ESP\_FAIL: For generic failure.

`esp_err_t esp_msc_ota_perform (esp_msc_ota_handle_t handle)`

Read data from the firmware on the USB flash drive and start the upgrade,.

It is necessary to call this function several times and ensure that the value returned each time is ESP\_OK. and call `esp_msc_ota_is_complete_data_received` to monitor whether the firmware upgrade is complete or not. Make sure that the VFS file system is not unmounted during the `fread` process. If you manually unplug the USB flash drive or log out of the USB HOST, stop calling `esp_msc_ota_perform` before and call `esp_msc_ota_abort` afterwards.

参数 **handle** –[in] Handle for the MSC ota

### 返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG: Invalid argument
- ESP\_ERR\_INVALID\_STATE: Invalid state (handle not initialized, etc.)
- ESP\_ERR\_INVALID\_SIZE: Fread failed
- ESP\_FAIL: For generic failure.
- For other errors, please check the API for the specific error.

`esp_err_t esp_msc_ota_end (esp_msc_ota_handle_t handle)`

Clean-up MSC OTA Firmware upgrade.

---

**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_msc_ota_abort`

---

参数 **handle** –[in] Handle for the MSC ota

### 返回

- ESP\_ERR\_INVALID\_ARG: Invalid argument
- ESP\_ERR\_INVALID\_STATE: Incorrect status
- ESP\_OK: Success
- For other errors, please check the API for the specific error.

`esp_err_t esp_msc_ota_abort (esp_msc_ota_handle_t handle)`

Clean-up MSC OTA Firmware upgrade and call `esp_ota_abort`

---

**备注:** `esp_msc_ota_abort` should not be called after calling `esp_msc_ota_finish`

---

参数 **handle** –[in] Handle for the MSC ota

### 返回

- ESP\_ERR\_INVALID\_ARG: Invalid argument
- ESP\_ERR\_INVALID\_STATE: Incorrect status
- ESP\_OK: Success

- For other errors, please check the API for the specific error.

esp\_err\_t **esp\_msc\_ota** (*esp\_msc\_ota\_config\_t* \*config)

MSC OTA Firmware upgrade.

This function provides a complete set of MSC\_OTA upgrade procedures. When the USB flash disk is inserted, it will be upgraded automatically. After the upgrade is completed, please call `esp_restart()`

**参数** **config** –[in] pointer to *esp\_msc\_ota\_config\_t* structure

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG: Invalid argument
- ESP\_OK: Success
- For other errors, please check the API for the specific error.

esp\_err\_t **esp\_msc\_ota\_get\_img\_desc** (*esp\_msc\_ota\_handle\_t* 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.

**参数**

- **handle** –[in] pointer to *esp\_msc\_ota\_config\_t* structure
- **new\_app\_info** –[out] pointer to an allocated esp\_app\_desc\_t structure

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG: Invalid argument
- ESP\_ERR\_INVALID\_STATE: Incorrect status
- ESP\_FAIL: Fail to read image header

esp\_err\_t **esp\_msc\_ota\_set\_msc\_connect\_state** (*esp\_msc\_ota\_handle\_t* handle, bool if\_connect)

When you manage the MSC HOST function, call this api to notify msc\_ota that the u-disk has been inserted and the VFS filesystem has been mounted.

**参数**

- **handle** –[in] Handle for the MSC ota
- **if\_connect** –[in]

**返回** always return ESP\_OK

*esp\_msc\_ota\_status\_t* **esp\_msc\_ota\_get\_status** (*esp\_msc\_ota\_handle\_t* handle)

Get the status of the MSC ota.

**参数** **handle** –[in] Handle for the MSC ota

**返回** esp\_msc\_ota\_status\_t

bool **esp\_msc\_ota\_is\_complete\_data\_received** (*esp\_msc\_ota\_handle\_t* handle)

Checks if complete data was received or not.

This API can be called just before `esp_msc_ota_end()` to validate if the complete image was indeed received.

**参数** **handle** –[in] Handle for the MSC ota

**返回** true

**返回** false

## Structures

struct **esp\_msc\_ota\_config\_t**

esp msc ota config

## Public Members

const char \***ota\_bin\_path**

OTA binary name, must be an exact match. Note: By default file names cannot exceed 11 bytes e.g. “/usb/ota.bin”

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

TickType\_t **wait\_msc\_connect**

Wait time for MSC device to connect

size\_t **buffer\_size**

Buffer size for OTA write operation, must larger than 1024

### Type Definitions

typedef void \***esp\_msc\_ota\_handle\_t**

Handle for the MSC ota.

### Enumerations

enum **esp\_msc\_ota\_event\_t**

Declare Event Base for ESP MSC OTA.

*Values:*

enumerator **ESP\_MSC\_OTA\_START**

Start update

enumerator **ESP\_MSC\_OTA\_READY\_UPDATE**

Ready to update

enumerator **ESP\_MSC\_OTA\_WRITE\_FLASH**

Flash write operation

enumerator **ESP\_MSC\_OTA\_FAILED**

Update failed

enumerator **ESP\_MSC\_OTA\_GET\_IMG\_DESC**

Get image description

enumerator **ESP\_MSC\_OTA\_VERIFY\_CHIP\_ID**

Verify chip id

enumerator **ESP\_MSC\_OTA\_UPDATE\_BOOT\_PARTITION**

Boot partition update after successful ota update

enumerator **ESP\_MSC\_OTA\_FINISH**

OTA finished



```

enumerator ESP_MSC_OTA_ABORT
    OTA aborted

enum esp_msc_ota_status_t
    Values:

enumerator ESP_MSC_OTA_INIT

enumerator ESP_MSC_OTA_BEGIN

enumerator ESP_MSC_OTA_IN_PROGRESS

enumerator ESP_MSC_OTA_SUCCESS

```

### 5.2.3 USB 主机 CDC

iot\_usbhc\_cdc 组件实现了一个 USB 主机 CDC 驱动的简化版本。该 API 的设计类似于 [ESP-IDF UART 驱动程序](#)，可在一些应用中替代 UART，快速实现从 UART 到 USB 的迁移。

#### 使用指南

1. 使用 `usbhc_cdc_driver_install` 配置，用户可以简单配置 bulk 端点地址和内部 ringbuffer 的大小，除此之外，用户还可以配置热插拔相关的回调函数 `conn_callback` `disconn_callback`：

```

/* 安装 USB 主机 CDC 驱动程序，配置 bulk 端点地址和内部 ringbuffer 的大小 */
static usbhc_cdc_config_t config = {
    /* use default endpoint descriptor with user address */
    .bulk_in_ep_addr = EXAMPLE_BULK_IN_EP_ADDR,
    .bulk_out_ep_addr = EXAMPLE_BULK_OUT_EP_ADDR,
    .rx_buffer_size = IN_RINGBUF_SIZE,
    .tx_buffer_size = OUT_RINGBUF_SIZE,
    .conn_callback = usb_connect_callback,
    .disconn_callback = usb_disconnect_callback,
};
/* 如果用户想要使用多个接口，可以像这样配置 */
#ifdef EXAMPLE_BULK_ITF_NUM > 1
config.itf_num = 2;
config.bulk_in_ep_addrs[1] = EXAMPLE_BULK_IN1_EP_ADDR;
config.bulk_out_ep_addrs[1] = EXAMPLE_BULK_OUT1_EP_ADDR;
config.rx_buffer_sizes[1] = IN_RINGBUF_SIZE;
config.tx_buffer_sizes[1] = OUT_RINGBUF_SIZE;
#endif

/* 安装 USB 主机 CDC 驱动程序 */
usbhc_cdc_driver_install(&config);

/* 等待 USB 设备连接 */
usbhc_cdc_wait_connect(portMAX_DELAY);

```

2. 驱动程序初始化后，内部状态机将自动处理 USB 的热插拔。
3. `usbhc_cdc_wait_connect` 可以用于阻塞任务，直到 USB CDC 设备连接或超时。
4. 成功连接后，主机将自动从 CDC 设备接收 USB 数据到内部 ringbuffer，用户可以轮询 `usbhc_cdc_get_buffered_data_len` 以读取缓冲数据大小，或者注册接收回调以在数据准备就绪时得到通知。然后 `usbhc_cdc_read_bytes` 可以用于读取缓冲数据。

5. `usbh_cdc_write_bytes` 可以用于向 USB 设备发送数据。数据首先被写入内部传输 ringbuffer，然后在 USB 总线空闲时发送出去。
6. `usbh_cdc_driver_delete` 可以完全卸载 USB 驱动程序以释放所有资源。
7. 如果配置多个 CDC 接口，每个接口都包含一个 IN 和 OUT 端点。用户可以使用 `usbh_cdc_itf_read_bytes` 和 `usbh_cdc_itf_write_bytes` 与指定的接口通信。

## 示例代码

[usb/host/usb\\_cdc\\_basic](#)

## API 参考

### Header File

- [components/usb/iot\\_usbh\\_cdc/include/iot\\_usbh\\_cdc.h](#)

### Functions

`esp_err_t usbh_cdc_driver_install` (const [usbh\\_cdc\\_config\\_t](#) \*config)

Install USB CDC driver.

参数 **config** –USB Host CDC configs

返回 ESP\_ERR\_INVALID\_STATE driver has been installed ESP\_ERR\_INVALID\_ARG args not supported ESP\_FAIL driver install failed ESP\_OK driver install succeed

`esp_err_t usbh_cdc_driver_delete` (void)

Uninstall USB driver.

返回 ESP\_ERR\_INVALID\_STATE driver not installed ESP\_OK start succeed

`esp_err_t usbh_cdc_wait_connect` (TickType\_t ticks\_to\_wait)

Waiting until CDC device connected.

参数 **ticks\_to\_wait** –timeout value, count in RTOS ticks

返回 ESP\_ERR\_INVALID\_STATE driver not installed ESP\_ERR\_TIMEOUT wait timeout ESP\_OK device connect succeed

`int usbh_cdc_write_bytes` (const uint8\_t \*buf, size\_t length)

Send data to interface 0 of connected USB device from a given buffer and length, this function will return after copying all the data to tx ringbuffer.

参数

- **buf** –data buffer address
- **length** –data length to send

返回 int The number of bytes pushed to the tx buffer

`int usbh_cdc_itf_write_bytes` (uint8\_t itf, const uint8\_t \*buf, size\_t length)

Send data to specified interface of connected USB device from a given buffer and length, this function will return after copying all the data to tx buffer.

参数

- **itf** –the interface index
- **buf** –data buffer address
- **length** –data length to send

返回 int The number of bytes pushed to the tx buffer

`esp_err_t usbh_cdc_get_buffered_data_len` (size\_t \*size)

Get USB interface 0 rx buffered data length.

参数 **size** –data length buffered

返回 ESP\_ERR\_INVALID\_STATE cdc not configured, or not running  
ESP\_ERR\_INVALID\_ARG args not supported ESP\_OK start succeed

`esp_err_t usbh_cdc_itf_get_buffered_data_len (uint8_t itf, size_t *size)`

Get USB specified interface rx buffered data length.

**参数**

- **itf** –the interface index
- **size** –data length buffered

**返回** ESP\_ERR\_INVALID\_STATE cdc not configured, or not running  
ESP\_ERR\_INVALID\_ARG args not supported ESP\_OK start succeed

`int usbh_cdc_read_bytes (uint8_t *buf, size_t length, TickType_t ticks_to_wait)`

Read data bytes from interface 0 rx buffer.

**参数**

- **buf** –data buffer address
- **length** –data buffer size
- **ticks\_to\_wait** –timeout value, count in RTOS ticks

**返回** int the number of bytes actually read from rx buffer

`int usbh_cdc_itf_read_bytes (uint8_t itf, uint8_t *buf, size_t length, TickType_t ticks_to_wait)`

Read data bytes from specified interface rx buffer.

**参数**

- **itf** –the interface index
- **buf** –data buffer address
- **length** –data buffer size
- **ticks\_to\_wait** –timeout value, count in RTOS ticks

**返回** int the number of bytes actually read from rx buffer

`int usbh_cdc_get_itf_state (uint8_t itf)`

Get the connect state of given interface.

**参数** **itf** –the interface index

**返回** \*\* int true is ready, false is not ready

`esp_err_t usbh_cdc_flush_rx_buffer (uint8_t itf)`

Flush rx buffer, discard all the data in the ring-buffer.

**参数** **itf** –the interface index

**返回** \*\* esp\_err\_t ESP\_ERR\_INVALID\_STATE cdc not configured, or not running  
ESP\_ERR\_INVALID\_ARG args not supported ESP\_OK start succeed

`esp_err_t usbh_cdc_flush_tx_buffer (uint8_t itf)`

Flush tx buffer, discard all the data in the ring-buffer.

**参数** **itf** –the interface index

**返回** \*\* esp\_err\_t ESP\_ERR\_INVALID\_STATE cdc not configured, or not running  
ESP\_ERR\_INVALID\_ARG args not supported ESP\_OK start succeed

## Structures

`struct usbh_cdc_config`

USB host CDC configuration type `itf_num` is the total enabled interface numbers, can not exceed `CDC_INTERFACE_NUM_MAX`, `itf_num = 0` is same as `itf_num = 1` for back compatibility, if `itf_num > 1`, user should config params with `s` ending like `bulk_in_ep_addrs` to config each interface, callbacks should not in block state.

## Public Members

`int itf_num`

interface numbers enabled

`uint8_t bulk_in_ep_addr`

USB CDC bulk in endpoint address, will be overwritten if `bulk_in_ep` is specified

`uint8_t bulk_in_ep_addrs[CDC_INTERFACE_NUM_MAX]`

USB CDC bulk in endpoints addresses, saved as an array, will be overwritten if `bulk_in_ep` is specified

`uint8_t bulk_out_ep_addr`

USB CDC bulk out endpoint address, will be overwritten if `bulk_out_ep` is specified

`uint8_t bulk_out_ep_addrs[CDC_INTERFACE_NUM_MAX]`

USB CDC bulk out endpoint addresses, saved as an array, will be overwritten if `bulk_out_ep` is specified

`int rx_buffer_size`

USB receive/in ringbuffer size

`int rx_buffer_sizes[CDC_INTERFACE_NUM_MAX]`

USB receive/in ringbuffer size of each interface

`int tx_buffer_size`

USB transmit/out ringbuffer size

`int tx_buffer_sizes[CDC_INTERFACE_NUM_MAX]`

USB transmit/out ringbuffer size of each interface

`usb_ep_desc_t *bulk_in_ep`

USB CDC bulk in endpoint descriptor, set NULL if using default param

`usb_ep_desc_t *bulk_in_eps[CDC_INTERFACE_NUM_MAX]`

USB CDC bulk in endpoint descriptors of each interface, set NULL if using default param

`usb_ep_desc_t *bulk_out_ep`

USB CDC bulk out endpoint descriptor, set NULL if using default param

`usb_ep_desc_t *bulk_out_eps[CDC_INTERFACE_NUM_MAX]`

USB CDC bulk out endpoint descriptors of each interface, set NULL if using default param

`usbh_cdc_cb_t conn_callback`

USB connect callback, set NULL if not use

`usbh_cdc_cb_t disconn_callback`

USB disconnect callback, set NULL if not use

`usbh_cdc_cb_t rx_callback`

packet receive callback, set NULL if not use

`usbh_cdc_cb_t rx_callbacks[CDC_INTERFACE_NUM_MAX]`

packet receive callbacks of each interface, set NULL if not use

void **\*conn\_callback\_arg**

USB connect callback arg, set NULL if not use

void **\*disconn\_callback\_arg**

USB disconnect callback arg, set NULL if not use

void **\*rx\_callback\_arg**

packet receive callback arg, set NULL if not use

void **\*rx\_callback\_args**[CDC\_INTERFACE\_NUM\_MAX]

packet receive callback arg of each interface, set NULL if not use

## Macros

**CDC\_INTERFACE\_NUM\_MAX**

## Type Definitions

typedef void (\***usbh\_cdc\_cb\_t**)(void \*arg)

USB receive callback type.

typedef struct *usbh\_cdc\_config* **usbh\_cdc\_config\_t**

USB host CDC configuration type itf\_num is the total enabled interface numbers, can not exceed CDC\_INTERFACE\_NUM\_MAX, itf\_num = 0 is same as itf\_num = 1 for back compatibility, if itf\_num > 1, user should config params with s ending like bulk\_in\_ep\_addrs to config each interface, callbacks should not in block state.

## 5.3 USB 设备驱动

### 5.3.1 USB Device UVC

usb\_device\_uvc 是用于 ESP32-S2/ESP32-S3 的 USB UVC 设备驱动程序，它支持将 JPEG 帧流传输到 USB 主机。用户可以通过回调函数将相机或任何设备包装成符合 UVC 标准的设备。

特性：

1. 支持通过 UVC 流接口进行视频流传输
2. 支持 Isochronous 和 Bulk 模式
3. 支持多种分辨率和帧率

### 将组件添加到项目

请使用组件管理器命令 add-dependency 将 usb\_device\_uvc 添加到项目的依赖项中，在 CMake 步骤中会自动下载该组件

```
idf.py add-dependency "espressif/usb_device_uvc=*
```

## 用户参考

该组件仅提供一个 API 用于配置 UVC 设备。由于驱动基于 TinyUSB 堆栈，因此未提供 deinit API。

```
#include "usb_device_uvc.h"

static esp_err_t camera_start_cb(uvc_format_t format, int width, int height, int_
↪rate, void *cb_ctx)
{
    // 用户可以在这里初始化相机
    // 相机应根据给定的格式、宽度、高度和帧率进行初始化
    return ESP_OK;
}

static void camera_stop_cb(void *cb_ctx)
{
    // 用户代码
    return;
}

static uvc_fb_t* camera_fb_get_cb(void *cb_ctx)
{
    // 用户代码以返回图像帧缓冲区
    // 相机应准备下一帧，并返回帧缓冲区
    return uvc_fb;
}

static void camera_fb_return_cb(uvc_fb_t *fb, void *cb_ctx)
{
    // 在帧缓冲区被复制到传输缓冲区后返回
    // 用户代码以回收帧缓冲区
    return;
}

// 缓冲区用于存储要发送到主机的数据
const size_t buff_size = 30 * 1024;
uint8_t *uvc_buffer = (uint8_t *)heap_caps_malloc(buff_size, MALLOC_CAP_DEFAULT);
assert(uvc_buffer != NULL);

uvc_device_config_t config = {
    .uvc_buffer = uvc_buffer,
    .uvc_buffer_size = 40 * 1024,
    .start_cb = camera_start_cb,
    .fb_get_cb = camera_fb_get_cb,
    .fb_return_cb = camera_fb_return_cb,
    .stop_cb = camera_stop_cb,
    .cb_ctx = NULL,
};

ESP_ERROR_CHECK(uvc_device_config(0, &config));
ESP_ERROR_CHECK(uvc_device_init());
```

## 示例

[usb/device/usb\\_webcam](#) [usb/device/usb\\_dual\\_uvc\\_device](#)

## API 参考

### Header File

- [components/usb/usb\\_device\\_uvc/include/usb\\_device\\_uvc.h](#)

### Functions

`esp_err_t uvc_device_config` (int index, *uvc\_device\_config\_t* \*config)

Configure the UVC device by uvc device number.

#### 参数

- **index** –UVC device index number [0,1]
- **config** –Configuration for the UVC device

**返回** ESP\_OK on success ESP\_ERR\_INVALID\_ARG if the configuration is invalid ESP\_FAIL if the UVC device could not be initialized

`esp_err_t uvc_device_init` (void)

Initialize the UVC device, after this function is called, the UVC device will be visible to the host and the host can open the UVC device with the specific format and resolution.

**返回** ESP\_OK on success ESP\_FAIL if the UVC device could not be initialized

### Structures

struct **uvc\_fb\_t**

Frame buffer structure.

### Public Members

uint8\_t \***buf**

Pointer to the frame data

size\_t **len**

Length of the buffer in bytes

size\_t **width**

Width of the image frame in pixels

size\_t **height**

Height of the image frame in pixels

*uvc\_format\_t* **format**

Format of the frame data

struct timeval **timestamp**

Timestamp since boot of the frame

struct **uvc\_device\_config\_t**

Configuration for the UVC device.

### Public Members

uint8\_t \***uvc\_buffer**

UVC transfer buffer

uint32\_t **uvc\_buffer\_size**

UVC transfer buffer size, should bigger than one frame size

*uvc\_input\_start\_cb\_t* **start\_cb**

callback function of host open the UVC device with the specific format and resolution

*uvc\_input\_fb\_get\_cb\_t* **fb\_get\_cb**

callback function of host request a new frame buffer

*uvc\_input\_fb\_return\_cb\_t* **fb\_return\_cb**

callback function of the frame buffer is no longer used

*uvc\_input\_stop\_cb\_t* **stop\_cb**

callback function of host close the UVC device

void \***cb\_ctx**

callback context, for user specific usage

### Type Definitions

typedef esp\_err\_t (\***uvc\_input\_start\_cb\_t**)(*uvc\_format\_t* format, int width, int height, int rate, void \*cb\_ctx)

type of callback function when host open the UVC device

typedef *uvc\_fb\_t* (\***uvc\_input\_fb\_get\_cb\_t**)(void \*cb\_ctx)

type of callback function when host request a new frame buffer

typedef void (\***uvc\_input\_fb\_return\_cb\_t**)(*uvc\_fb\_t* \*fb, void \*cb\_ctx)

type of callback function when the frame buffer is no longer used

typedef void (\***uvc\_input\_stop\_cb\_t**)(void \*cb\_ctx)

type of callback function when host close the UVC device

### Enumerations

enum **uvc\_format\_t**

UVC format.

*Values:*

enumerator **UVC\_FORMAT\_JPEG**

JPEG format

enumerator **UVC\_FORMAT\_H264**

H264 format

## 5.3.2 ESP Device UAC

*esp\_device\_uac* 是基于 TinyUSB 的 USB Audio Class 驱动库，它支持将 ESP 芯片模拟成为一个音频设备，支持自定义音频采样率，麦克风通道数，扬声器通道数等。

特性：

1. 默认支持 ISO FeedBack 通信接口，根据 UAC FIFO 内存剩余大小自动向主机端同步，[参考](#)。



2. 支持自定义音频采样率，麦克风通道数，扬声器通道数。
3. 支持扬声器数据到来时候先缓冲一段数据，再进行传输，有助于减少音频数据传输的中断频率。

## USB Device UAC 用户指南

- 开发板
  1. 可以使用任何带有 USB 接口的 ESP32-S2/ESP32-S3 开发板
- USB MIC 回调函数
  1. `uac_input_cb_t` 回调函数用于将音频数据传输到 USB 主机端，用户应该按照时间轴传输音频，或者在该回调函数中堵塞的等待音频数据到来。
  2. 通过设置 `CONFIG_UAC_MIC_INTERVAL_MS` 宏定义来设置回调函数读取音频数据的长度。
    - 设置 `CONFIG_UAC_MIC_INTERVAL_MS=10` 在 48000HZ 采样率，16 位精度，单通道情况下，每次读取的数据量为  $10\text{ms} * 48000\text{HZ} / 1000 * 2\text{byte} = 960\text{byte}$
  3. 通过设置 `UAC_SPK_INTERVAL_MS` 宏定义来设置回调函数第一次写入音频的长度，为了防止音频数据传输的中断频率过高，默认为 10ms，后续音频写入会按照大约 1ms 的数据量进行传输。
  4. `UAC_SPK_NEW_PLAY_INTERVAL` 宏定义用于判断到来的音频是否是新音频，如果是新音频，会先缓冲一段数据，再进行传输，有助于减少音频数据传输的中断频率。
  5. `UAC_SUPPORT_MACOS` 宏用于支持 MacOS 系统，注意开启该宏定义后，windows 系统可能无法识别设备。

## USB Device UAC API 参考

1. 用户可通过调用 `uac_device_config_t` 函数配置音频输入输出，静音，音量四个回调函数。

```
uac_device_config_t config = {
    .output_cb = uac_device_output_cb,           // Speaker output callback
    .input_cb = uac_device_input_cb,             // Microphone input callback
    .set_mute_cb = uac_device_set_mute_cb,       // Set mute callback
    .set_volume_cb = uac_device_set_volume_cb,   // Set volume callback
    .cb_ctx = NULL,
};
uac_device_init(&config);
```

## Example

- [usb/device/usb\\_uac](#)

## API Reference

### Header File

- [components/usb/usb\\_device\\_uac/include/usb\\_device\\_uac.h](#)

### Functions

`esp_err_t uac_device_init(uac_device_config_t *config)`

Initialize the USB Audio Class (UAC) device.

**参数** `config` – Pointer to the UAC device configuration structure.

**返回**

- `ESP_OK` on success
- `ESP_FAIL` on failure

## Structures

struct **uac\_device\_config\_t**

USB UAC Device Config.

## Public Members

bool **skip\_tinyusb\_init**

if true, the Tinyusb and usb phy will not be initialized

*uac\_output\_cb\_t* **output\_cb**

callback function for UAC data output, if NULL, output will be disabled

*uac\_input\_cb\_t* **input\_cb**

callback function for UAC data input, if NULL, input will be disabled

*uac\_set\_mute\_cb\_t* **set\_mute\_cb**

callback function for set mute, if NULL, the set mute request will be ignored

*uac\_set\_volume\_cb\_t* **set\_volume\_cb**

callback function for set volume, if NULL, the set volume request will be ignored

void \***cb\_ctx**

callback context, for user specific usage

## Type Definitions

typedef esp\_err\_t (\***uac\_output\_cb\_t**)(uint8\_t \*buf, size\_t len, void \*cb\_ctx)

typedef esp\_err\_t (\***uac\_input\_cb\_t**)(uint8\_t \*buf, size\_t len, size\_t \*bytes\_read, void \*cb\_ctx)

typedef void (\***uac\_set\_mute\_cb\_t**)(uint32\_t mute, void \*cb\_ctx)

typedef void (\***uac\_set\_volume\_cb\_t**)(uint32\_t volume, void \*cb\_ctx)



## Chapter 6

# 音频

### 6.1 PWM 音频

支持的芯片	ESP32	ESP32-S2	ESP32-S3	ESP32-C3
-------	-------	----------	----------	----------

PWM 音频功能使用 ESP32 内部的 LEDC 外设产生 PWM 播放音频，无需使用外部的音频 Codec 芯片，适用于对音质要求不高而对成本敏感的应用。

#### 6.1.1 特性

- 允许使用任意具有输出功能的 GPIO 作为音频输出管脚
- 支持 8 ~ 10 位的 PWM 分辨率
- 支持立体声
- 支持 8 ~ 48 KHz 采样率

#### 6.1.2 结构

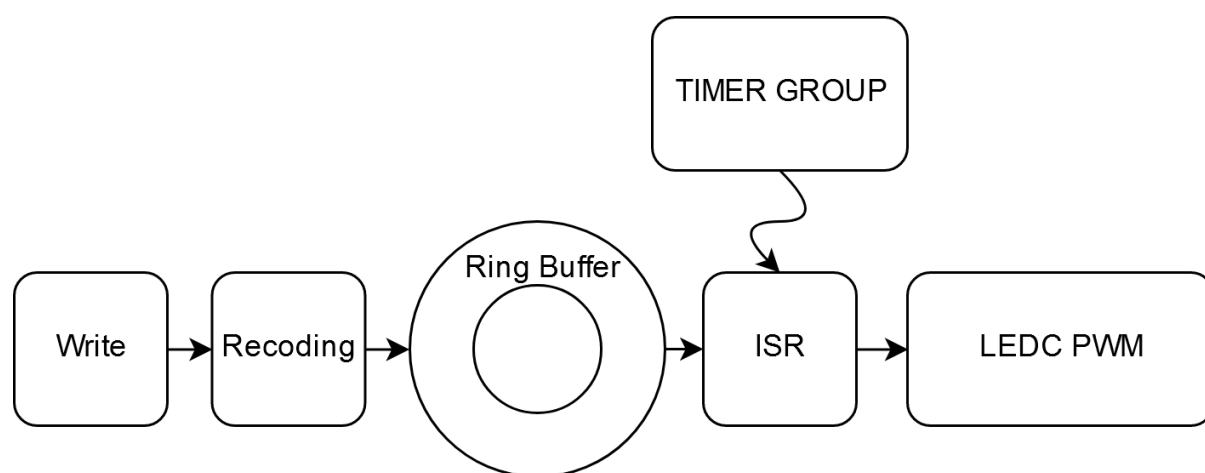


图 1: 结构

1. 首先对写入的数据进行编码调整以满足 PWM 的输入要求，包括数据的移位和偏移；
2. 通过一个 Ring Buffer 将数据发送到 Timer Group 的 ISR (Interrupt Service Routines) 函数；
3. Timer Group 按照设定的采样率从 Ring Buffer 读出数据并写入到 LEDC。

**备注：** 由于输出的是 PWM 信号，需要将输出进行低通滤波后才能得到音频波形。

### 6.1.3 PWM 频率

输出 PWM 的频率不支持直接配置，需通过配置 PWM 分辨率位数来计算，计算公式如下：

$$f_{pwm} = \frac{f_{APB\_CLK}}{2^{res\_bits}} - \left( \frac{f_{APB\_CLK}}{2^{res\_bits}} MOD 1000 \right)$$

其中  $f_{APB\_CLK}$  为 80 MHz,  $res\_bits$  为 PWM 分辨率位数，当分辨率为 LEDC\_TIMER\_10\_BIT 时，PWM 频率为 78 KHz。更高的 PWM 频率和分辨率可以更好地还原音频信号，但是从公式可以看出，增大 PWM 频率意味着分辨率降低，提高分辨率意味着频率减小，可根据实际应用调整达到良好的平衡。

### 6.1.4 应用示例

```
pwm_audio_config_t pac;
pac.duty_resolution      = LEDC_TIMER_10_BIT;
pac.gpio_num_left        = LEFT_CHANNEL_GPIO;
pac.ledc_channel_left    = LEDC_CHANNEL_0;
pac.gpio_num_right       = RIGHT_CHANNEL_GPIO;
pac.ledc_channel_right   = LEDC_CHANNEL_1;
pac.ledc_timer_sel       = LEDC_TIMER_0;
pac.tg_num               = TIMER_GROUP_0;
pac.timer_num            = TIMER_0;
pac.ringbuf_len          = 1024 * 8;

pwm_audio_init(&pac);           /**< Initialize pwm audio */
pwm_audio_set_param(48000, 8, 2); /**< Set sample rate, bits and channel
↪number */
pwm_audio_start();              /**< Start to run */

while(1) {

    /**< Prepare audio data, such as decode mp3/wav file

    /**< Write data to play */
    pwm_audio_write(audio_data, length, &written, 1000 / portTICK_PERIOD_
↪MS);
}
```

### 6.1.5 API 参考

#### Header File

- [components/audio/pwm\\_audio/include/pwm\\_audio.h](#)

#### Functions

esp\_err\_t **pwm\_audio\_init** (const [pwm\\_audio\\_config\\_t](#) \*cfg)

Initializes and configure the pwm audio.

**参数** **cfg** –configurations - see [pwm\\_audio\\_config\\_t](#) struct

**返回**

- ESP\_OK Success
- ESP\_FAIL timer\_group or ledc initialize failed
- ESP\_ERR\_INVALID\_ARG if argument wrong
- ESP\_ERR\_INVALID\_STATE The pwm audio already configure
- ESP\_ERR\_NO\_MEM Memory allocate failed

esp\_err\_t **pwm\_audio\_deinit** (void)

Deinitialize LEDC timer\_group and output gpio.

**返回**

- ESP\_OK Success
- ESP\_FAIL pwm\_audio not initialized

esp\_err\_t **pwm\_audio\_start** (void)

Start to run pwm audio.

**返回**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE pwm\_audio not initialized or it already running

esp\_err\_t **pwm\_audio\_stop** (void)

Stop pwm audio.

**Attention** Only stop timer, and the pwm will keep to output. If you want to stop pwm output, call pwm\_audio\_deinit function

**返回**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE pwm\_audio not initialized or it already idle

esp\_err\_t **pwm\_audio\_write** (uint8\_t \*inbuf, size\_t len, size\_t \*bytes\_written, TickType\_t ticks\_to\_wait)

Write data to play.

**参数**

- **inbuf** –Pointer source data to write
- **len** –length of data in bytes
- **bytes\_written** –[out] Number of bytes written, if timeout, the result will be less than the size passed in.
- **ticks\_to\_wait** –TX buffer wait timeout in RTOS ticks. If this many ticks pass without space becoming available in the DMA transmit buffer, then the function will return (note that if the data is written to the DMA buffer in pieces, the overall operation may still take longer than this timeout.) Pass portMAX\_DELAY for no timeout.

**返回**

- ESP\_OK Success
- ESP\_FAIL Write encounter error
- ESP\_ERR\_INVALID\_ARG Parameter error

esp\_err\_t **pwm\_audio\_set\_param** (int rate, ledc\_timer\_bit\_t bits, int ch)

Set audio parameter, Similar to pwm\_audio\_set\_sample\_rate(), but also sets bit width.

**Attention** After start pwm audio, it can't modify parameters by call function pwm\_audio\_set\_param. If you want to change the parameters, must stop pwm audio by call function pwm\_audio\_stop.

**参数**

- **rate** –sample rate (ex: 8000, 44100...)
- **bits** –bit width

- **ch** –channel number

返回

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

esp\_err\_t **pwm\_audio\_set\_sample\_rate** (int rate)

Set sample rate.

参数 **rate** –sample rate (ex: 8000, 44100...)

返回

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

esp\_err\_t **pwm\_audio\_set\_volume** (int8\_t volume)

Set volume for pwm audio.

**Attention** when the volume is too small, it will produce serious distortion

参数 **volume** –Volume to set (-16 ~ 16). Set to 0 for original output; Set to less than 0 for attenuation, and -16 is mute; Set to more than 0 for enlarge, and 16 is double output

返回

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

esp\_err\_t **pwm\_audio\_get\_volume** (int8\_t \*volume)

Get current volume.

参数 **volume** –Pointer to volume

返回

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

esp\_err\_t **pwm\_audio\_get\_param** (int \*rate, int \*bits, int \*ch)

Get parameter for pwm audio.

参数

- **rate** –sample rate, if you don't care about this parameter, set it to NULL
- **bits** –bit width, if you don't care about this parameter, set it to NULL
- **ch** –channel number, if you don't care about this parameter, set it to NULL

返回

- Always return ESP\_OK

esp\_err\_t **pwm\_audio\_get\_status** (*pwm\_audio\_status\_t* \*status)

get pwm audio status

参数 **status** –current pwm\_audio status

返回

- ESP\_OK Success

## Structures

struct **pwm\_audio\_config\_t**

Configuration parameters for pwm\_audio\_init function.

## Public Members

int **gpio\_num\_left**  
the LEDC output gpio\_num, Left channel

int **gpio\_num\_right**  
the LEDC output gpio\_num, Right channel

ledc\_channel\_t **ledc\_channel\_left**  
LEDC channel (0 - 7), Corresponding to left channel

ledc\_channel\_t **ledc\_channel\_right**  
LEDC channel (0 - 7), Corresponding to right channel

ledc\_timer\_t **ledc\_timer\_sel**  
Select the timer source of channel (0 - 3)

ledc\_timer\_bit\_t **duty\_resolution**  
ledc pwm bits

uint32\_t **ringbuf\_len**  
ringbuffer size

### Enumerations

enum **pwm\_audio\_status\_t**  
pwm audio status  
*Values:*

enumerator **PWM\_AUDIO\_STATUS\_UN\_INIT**  
pwm audio uninitialized

enumerator **PWM\_AUDIO\_STATUS\_IDLE**  
pwm audio idle

enumerator **PWM\_AUDIO\_STATUS\_BUSY**  
pwm audio busy

enum **pwm\_audio\_channel\_t**  
pwm audio channel define  
*Values:*

enumerator **PWM\_AUDIO\_CH\_MONO**  
1 channel (mono)

enumerator **PWM\_AUDIO\_CH\_STEREO**  
2 channel (stereo)

enumerator **PWM\_AUDIO\_CH\_MAX**



## 6.2 DAC 音频

ESP32 拥有两个独立的 DAC 通道，并可直接使用 I2S 通过 DMA 播放音频。这里在 ESP-IDF 的基础上简化了 API，并对数据进行了重新编码以支持更多类型的采样位宽。

### 6.2.1 API 参考

#### Header File

- [components/audio/dac\\_audio/include/dac\\_audio.h](#)

#### Functions

`esp_err_t dac_audio_init(dac\_audio\_config\_t *cfg)`  
initialize i2s built-in dac to play audio with

**Attention** only support ESP32, because i2s of ESP32S2 not have a built-in dac

**参数** `cfg` –configurations - see [dac\\_audio\\_config\\_t](#) struct

**返回**

- ESP\_OK Success
- ESP\_FAIL Encounter error
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NO\_MEM Out of memory

`esp_err_t dac_audio_deinit(void)`  
deinitialize dac

**返回**

- ESP\_OK Success
- ESP\_FAIL Encounter error

`esp_err_t dac_audio_start(void)`  
Start dac to play.

**返回**

- ESP\_OK Success
- ESP\_FAIL Encounter error

`esp_err_t dac_audio_stop(void)`  
Stop play.

**返回**

- ESP\_OK Success
- ESP\_FAIL Encounter error

`esp_err_t dac_audio_set_param(int rate, int bits, int ch)`  
Configuration dac parameter.

**参数**

- **rate** –sample rate (ex: 8000, 44100...)
- **bits** –bit width
- **ch** –channel number

**返回**

- ESP\_OK Success
- ESP\_FAIL Encounter error

- ESP\_ERR\_INVALID\_ARG Parameter error

esp\_err\_t **dac\_audio\_set\_volume** (int8\_t volume)

Set volume.

**Attention** Using volume greater than 0 may cause variable overflow and distortion. Usually you should enter a volume less than or equal to 0.

**参数 volume** –Volume to set (-16 ~ 16), see Macro VOLUME\_0DB. Set to 0 for original output; Set to less than 0 for attenuation, and -16 is mute; Set to more than 0 for enlarge, and 16 is double output.

**返回**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

esp\_err\_t **dac\_audio\_write** (uint8\_t \*inbuf, size\_t len, size\_t \*bytes\_written, TickType\_t ticks\_to\_wait)

Write data to play.

**参数**

- **inbuf** –Pointer source data to write
- **len** –length of data in bytes
- **bytes\_written** –[out] Number of bytes written, if timeout, the result will be less than the size passed in.
- **ticks\_to\_wait** –TX buffer wait timeout in RTOS ticks. If this many ticks pass without space becoming available in the DMA transmit buffer, then the function will return (note that if the data is written to the DMA buffer in pieces, the overall operation may still take longer than this timeout.) Pass portMAX\_DELAY for no timeout.

**返回**

- ESP\_OK Success
- ESP\_FAIL Write encounter error
- ESP\_ERR\_INVALID\_ARG Parameter error

## Structures

struct **dac\_audio\_config\_t**

Configuration parameters for dac\_audio\_init function.

## Public Members

i2s\_port\_t **i2s\_num**

I2S\_NUM\_0, I2S\_NUM\_1

int **sample\_rate**

I2S sample rate

i2s\_bits\_per\_sample\_t **bits\_per\_sample**

I2S bits per sample

i2s\_dac\_mode\_t **dac\_mode**

DAC mode configurations - see i2s\_dac\_mode\_t

int **dma\_buf\_count**

DMA buffer count, number of buffer

int **dma\_buf\_len**

DMA buffer length, length of each buffer

uint32\_t **max\_data\_size**

one time max write data size

## Chapter 7

# 图形界面

### 7.1 LVGL 图形库

LVGL 是一个 C 语言编写的免费的开源图形库，提供了用于嵌入式 GUI 的各种元素。用户可以利用丰富的图形库资源，在消耗极低内存的情况下构建视觉效果丰富多彩的 GUI。

#### 7.1.1 主要特性

LVGL 具有以下特性：

- 超过 30 多种丰富的用户自定义控件，如按钮，滑条，文本框，键盘等
- 支持各种分辨率的屏幕，适配性好
- 接口简单，内存占用少
- 支持多个输入设备
- 提供抗锯齿，多边形，阴影等多种绘图元素
- 采用 UTF-8 编码，支持多语言，多字体的文本
- 支持各种图片类型，可从 Flash 和 SD 卡中读取图片显示
- 提供在线图片取模工具
- 支持 Micropython

#### 7.1.2 配置要求

运行 LVGL 的最低配置要求如下：

- 16、32、64 位的微控制器或处理器
- 时钟频率：大于 16 MHz
- Flash/ROM：大于 64 kB（推荐 180 kB）
- RAM：8 kB（推荐 24 kB）
- 需要一个帧缓存区
- 显示缓存：至少大于水平分辨率的像素
- C99 或更高版本的编译器

#### 7.1.3 在线工具

LVGL 提供了在线的 [字模提取工具](#) 和 [图片取模工具](#)。

### 7.1.4 示例方案

---

**备注：** 以下示例不再维护，LCD 以及 LVGL 示例请参考：[i80\\_controller](#)、[rgb\\_panel](#) 和 [spi\\_lcd\\_touch](#)

---

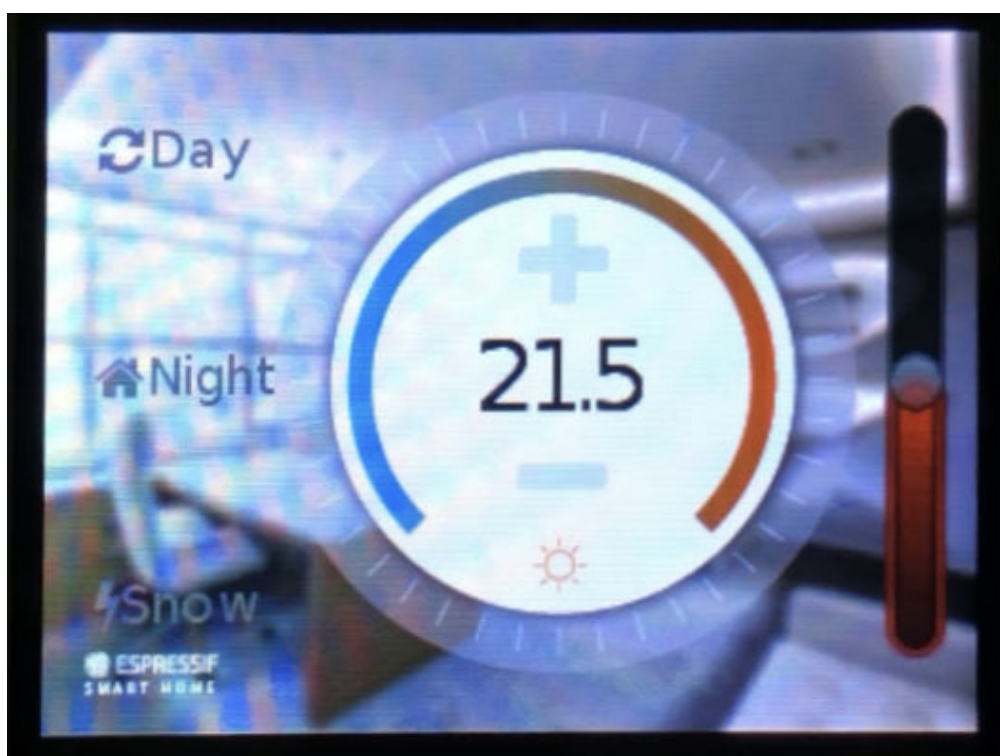
#### 官方例程

LVGL 官方提供了 ESP32 上使用 LVGL 的 [LVGL ESP32 示例程序](#)。

除此之外，在 ESP-IoT-Solution 中也提供了一些应用 LVGL 的实例：

#### thermostat

使用 LVGL 设计了一个恒温计控制的界面：



相应例程在 [hmi/lvgl\\_thermostat](#)

#### coffee

使用 LVGL 绘制了一个咖啡机的交互界面：

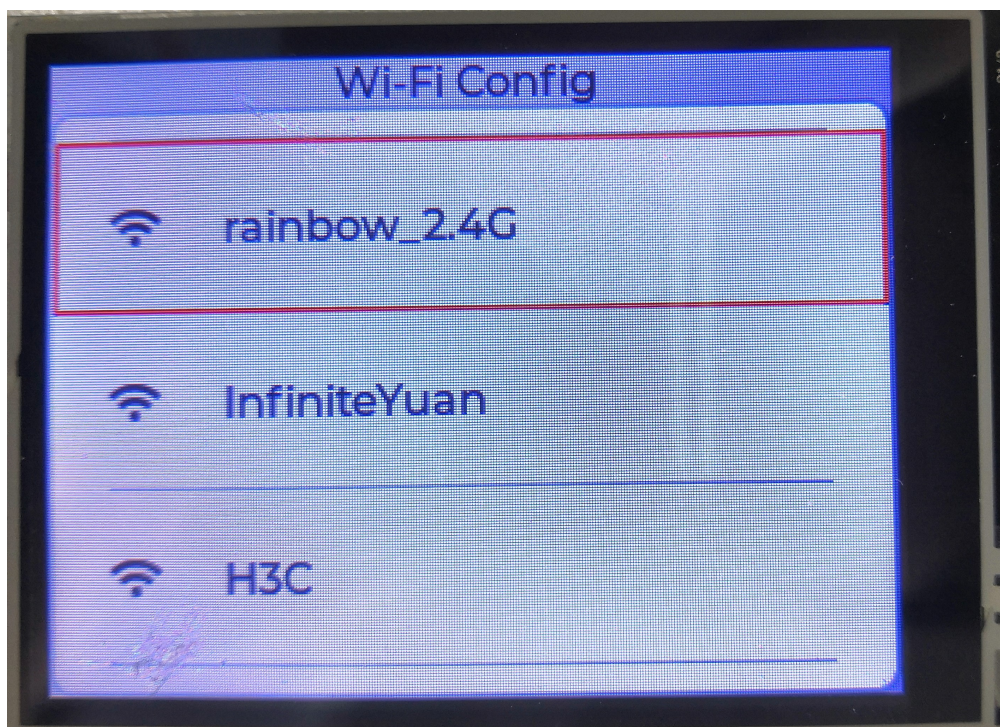
相应例程在 [hmi/lvgl\\_coffee](#)

#### wificonfig

ESP32 连接 Wi-Fi，利用 LVGL 绘制的 Wi-Fi 连接界面，可以显示附近 Wi-Fi 信息，在屏幕上输入密码等。

相应例程在 [hmi/lvgl\\_wificonfig](#)









## Chapter 8

# 人工智能

### 8.1 OpenAI

*openai* 组件是 [OpenAI-ESP32](#) Arduino 库的移植版本，它简化了在 *ESP-IDF* 上调用 OpenAI API 的过程。该库广泛支持大部分 OpenAI API 的功能，但不包括 *files* 和 *fine-tunes*。更多详情，请参考 [OpenAI](#) 的 API 参考文档。

#### 8.1.1 FAQ

使用 *audioTranscription*->*file* 报错 *Failed to allocate request buffer!*

该错误表明动态内存分配失败，通常是由于可用内存不足导致的。所需的内存大小取决于当前剩余的 RAM 或 PSRAM 大小。

解决方法

1. 开启 **PSRAM**: 如果设备支持 PSRAM，可以开启 PSRAM，以增加可用内存，并打开宏 *CONFIG\_SPIRAM\_USE\_MALLOC* 以使用 PSRAM。
2. 减少音频的数据量: 减少音频的数据量，例如减少音频的采样率、音频的长度等。

#### 8.1.2 API 参考

Header File

- [components/openai/include/OpenAI.h](#)

Functions

[OpenAI\\_t](#) \***OpenAICreate** (const char \*api\_key)

Create an *OpenAI* object.

参数 **api\_key** -The key of openai

返回 *OpenAI\_t*\* The *OpenAI* object



void **OpenAIDelete** (*OpenAI\_t* \*oai)

Clear the *OpenAI* object and release resources.

**参数** *oai* –The *OpenAI* object

void **OpenAIChangeBaseURL** (*OpenAI\_t* \*oai, const char \*baseURL)

Modify the Base URL of the *OpenAI* object.

## Structures

struct **OpenAI\_EmbeddingData\_t**

Struct for Embedding data.

## Public Members

uint32\_t **len**

Length of the data

double \***data**

Pointer to the data

struct **OpenAI\_EmbeddingResponse**

To get an embedding, send your text string to the embeddings API endpoint along with a choice of embedding model ID (e.g., text-embedding-ada-002). The response will contain an embedding, which you can extract, save, and use.

## Public Members

uint32\_t (\***getUsage**)(struct *OpenAI\_EmbeddingResponse* \*stringResponse)

Get the usage of *OpenAI\_EmbeddingResponse*.

**Param** *stringResponse[in]* The pointer to *OpenAI\_EmbeddingResponse*

**Return** uint32\_t

uint32\_t (\***getLen**)(struct *OpenAI\_EmbeddingResponse* \*embeddingData)

Get the length of *OpenAI\_EmbeddingResponse*.

**Param** *embeddingData[in]* The pointer to *OpenAI\_EmbeddingResponse*

**Return** uint32\_t

*OpenAI\_EmbeddingData\_t* (\***getData**)(struct *OpenAI\_EmbeddingResponse* \*embeddingData, uint32\_t index)

Get the data of *OpenAI\_EmbeddingResponse*.

**Param** *embeddingData[in]* The pointer to *OpenAI\_EmbeddingResponse*

**Param** *index[in]* The index of the data

**Return** *OpenAI\_EmbeddingData\_t*\*

char \*(\***getError**)(struct *OpenAI\_EmbeddingResponse* \*embeddingData)

Get the error of *OpenAI\_EmbeddingResponse*.

**Param** *embeddingData[in]* The pointer to *OpenAI\_EmbeddingResponse*

**Return** char\*

void (\***deleteResponse**)(struct *OpenAI\_EmbeddingResponse* \*embeddingData)

delete the embedding response, should free it after use.

**Param embeddingData[in]** the point of *OpenAI\_EmbeddingResponse*

struct **OpenAI\_ModerationResponse**

The moderations endpoint is a tool you can use to check whether content complies with *OpenAI*'s usage policies. Developers can thus identify content that our usage policies prohibits and take action, for instance by filtering it.

### Public Members

uint32\_t (\***getLen**)(struct *OpenAI\_ModerationResponse* \*moderationResponse)

Get the length of *OpenAI\_ModerationResponse*.

**Param moderationResponse[in]** The pointer to *OpenAI\_ModerationResponse*

**Return** uint32\_t

bool (\***getData**)(struct *OpenAI\_ModerationResponse* \*moderationResponse, uint32\_t index)

Get the moderation result of *OpenAI\_ModerationResponse*.

**Param moderationResponse[in]** The pointer to *OpenAI\_ModerationResponse*

**Param index[in]** The index of the moderation result

**Return** bool

char (\***getError**)(struct *OpenAI\_ModerationResponse* \*moderationResponse)

Get the error message of *OpenAI\_ModerationResponse*.

**Param moderationResponse[in]** The pointer to *OpenAI\_ModerationResponse*

**Return** char\*

void (\***deleteResponse**)(struct *OpenAI\_ModerationResponse* \*moderationResponse)

delete the moderation response, should free it after use.

**Param moderationResponse[in]** the point of } OpenAI\_ModerationResponse\_t

struct **OpenAI\_ImageResponse**

Save the image which is generated by *OpenAI*.

### Public Members

uint32\_t (\***getLen**)(struct *OpenAI\_ImageResponse* \*imageResponse)

Get the length of *OpenAI\_ImageResponse*.

**Param imageResponse[in]** The pointer to *OpenAI\_ImageResponse*

**Return** uint32\_t

char (\***getData**)(struct *OpenAI\_ImageResponse* \*imageResponse, uint32\_t index)

Get the data of *OpenAI\_ImageResponse*.

**Param imageResponse[in]** The pointer to *OpenAI\_ImageResponse*

**Param index[in]** The index of the image data

**Return** char\*

char **(\*getError)**(struct *OpenAI\_ImageResponse* \*imageResponse)

Get the error message of *OpenAI\_ImageResponse*.

**Param** imageResponse[in] The pointer to *OpenAI\_ImageResponse*  
**Return** char\*

void **(\*deleteResponse)**(struct *OpenAI\_ImageResponse* \*imageResponse)

delete the image response

**Param** imageResponse[in] the point of } OpenAI\_ImageResponse\_t

struct **OpenAI\_StringResponse**

Parse the returned json data into OpenAI\_StringResponse\_t.

### Public Members

uint32\_t **(\*getUsage)**(struct *OpenAI\_StringResponse* \*stringResponse)

get the usage of openai response

**Param** stringResponse[in] the point of OpenAI\_StringResponse\_t  
**Return** uint32\_t

uint32\_t **(\*getLen)**(struct *OpenAI\_StringResponse* \*stringResponse)

get the len of openai response

**Param** stringResponse[in] the point of OpenAI\_StringResponse\_t  
**Return** uint32\_t

char **(\*getData)**(struct *OpenAI\_StringResponse* \*stringResponse, uint32\_t index)

get the data of openai response

**Param** stringResponse[in] the point of OpenAI\_StringResponse\_t  
**Param** index[in] the index of data  
**Return** char\*

char **(\*getError)**(struct *OpenAI\_StringResponse* \*stringResponse)

get the error of openai response

**Param** stringResponse[in] the point of OpenAI\_StringResponse\_t  
**Return** char\*

void **(\*deleteResponse)**(struct *OpenAI\_StringResponse* \*stringResponse)

delete the openai response

**Param** stringResponse[in] the point of OpenAI\_StringResponse\_t

struct **OpenAI\_SpeechResponse**

Store the returned data into a OpenAI\_SpeechResponse\_t structure.

### Public Members

uint32\_t **(\*getLen)**(struct *OpenAI\_SpeechResponse* \*SpeechResponse)

get the len of openai speech response

**Param speechResponse[in]** the point of OpenAI\_SpeechResponse\_t  
**Return** uint32\_t

char **(\*getData)**(struct *OpenAI\_SpeechResponse* \*SpeechResponse)  
get the data of openai response

**Param SpeechResponse[in]** the point of OpenAI\_SpeechResponse\_t  
**Return** char\*

void **(\*deleteResponse)**(struct *OpenAI\_SpeechResponse* \*SpeechResponse)  
delete the openai response

**Param SpeechResponse[in]** the point of OpenAI\_SpeechResponse\_t

struct **OpenAI\_Completion**

Given a prompt, the model will return one or more predicted completions, and can also return the probabilities of alternative tokens at each position.

### Public Members

void **(\*setModel)**(struct *OpenAI\_Completion* \*completion, const char \*m)  
Set the model to use for completion.

**Param completion[in]** the point of OpenAI\_Completion\_t  
**Param m[in]** the name of the model to use for completion

void **(\*setMaxTokens)**(struct *OpenAI\_Completion* \*completion, uint32\_t mt)  
Set the maximum number of tokens to generate in the completion.

**Param completion[in]** the point of OpenAI\_Completion\_t  
**Param mt[in]** the maximum number of tokens to generate in the completion

void **(\*setTemperature)**(struct *OpenAI\_Completion* \*completion, float t)  
Set the temperature of the completion.

**Param completion[in]** the point of OpenAI\_Completion\_t  
**Param t[in]** float between 0 and 2. Higher value gives more random results.

void **(\*setTopP)**(struct *OpenAI\_Completion* \*completion, float tp)  
Set the value of top\_p for the completion.

**Param completion[in]** the point of OpenAI\_Completion\_t  
**Param tp[in]** float between 0 and 1. recommended to alter this or temperature but not both.

void **(\*setN)**(struct *OpenAI\_Completion* \*completion, uint32\_t n)  
Set the number of completions to generate for each prompt.

**Param completion[in]** the point of OpenAI\_Completion\_t  
**Param n[in]** the number of completions to generate for each prompt

void **(\*setEcho)**(struct *OpenAI\_Completion* \*completion, bool e)  
Echo back the prompt in addition to the completion.

**Param completion[in]** the point of OpenAI\_Completion\_t  
**Param e[in]** true if the prompt should be echoed back, false otherwise

void (\***setStop**)(struct *OpenAI\_Completion* \*completion, const char \*s)

Set up to 4 sequences where the API will stop generating further tokens.

**Param completion[in]** the point of *OpenAI\_Completion\_t*

**Param s[in]** the sequences where the API will stop generating further tokens

void (\***setPresencePenalty**)(struct *OpenAI\_Completion* \*completion, float pp)

Set the presence penalty for the completion.

**Param completion[in]** the point of *OpenAI\_Completion\_t*

**Param pp[in]** float between -2.0 and 2.0. Positive values increase the model's likelihood to talk about new topics.

void (\***setFrequencyPenalty**)(struct *OpenAI\_Completion* \*completion, float fp)

Set the frequency penalty for the completion.

**Param completion[in]** the point of *OpenAI\_Completion\_t*

**Param fp[in]** float between -2.0 and 2.0. Positive values decrease the model's likelihood to repeat the same line verbatim.

void (\***setBestOf**)(struct *OpenAI\_Completion* \*completion, uint32\_t bo)

Generates best\_of completions server-side and returns the "best". "best\_of" must be greater than "n".

**Param completion[in]** the point of *OpenAI\_Completion\_t*

**Param bo[in]** the number of best\_of completions to generate server-side and return the "best"

void (\***setUser**)(struct *OpenAI\_Completion* \*completion, const char \*u)

A unique identifier representing your end-user, which can help *OpenAI* to monitor and detect abuse.

**Param completion[in]** the point of *OpenAI\_Completion\_t*

**Param u[in]** the unique identifier representing your end-user

*OpenAI\_StringResponse\_t* \*(\***prompt**)(struct *OpenAI\_Completion* \*completion, char \*p)

Send the prompt for completion.

**Param completion[in]** the point of *OpenAI\_Completion\_t*

**Param p[in]** the prompt for completion

**Return** *OpenAI\_StringResponse\_t*\*

struct **OpenAI\_ChatCompletion**

Given a list of messages comprising a conversation, the model will return a response.

### Public Members

void (\***setModel**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, const char \*m)

Set the model to use for completion.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param m[in]** the name of the model to use for chatCompletion

void (\***setSystem**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, const char \*s)

Set the system to use for completion.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param s[in]** description of the required assistant

void (**\*setMaxTokens**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, uint32\_t mt)

Set the maximum number of tokens to generate in the completion.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param mt[in]** the maximum number of tokens to generate in the completion

void (**\*setTemperature**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, float t)

Set the temperature for the completion.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param t[in]** float between 0 and 2. Higher value gives more random results.

void (**\*setTopP**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, float tp)

Set the top\_p for the completion.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param tp[in]** float between 0 and 1. recommended to alter this or temperature but not both.

void (**\*setStop**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, const char \*s)

Set up to 4 sequences where the API will stop generating further tokens.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param s[in]** the sequences where the API will stop generating further tokens

void (**\*setPresencePenalty**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, float pp)

Set the presence penalty for the completion.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param pp[in]** float between -2.0 and 2.0. Positive values increase the model' s likelihood to talk about new topics.

void (**\*setFrequencyPenalty**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, float fp)

Set the frequency penalty for the completion.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param fp[in]** float between -2.0 and 2.0. Positive values decrease the model' s likelihood to repeat the same line verbatim.

void (**\*setUser**)(struct *OpenAI\_ChatCompletion* \*chatCompletion, const char \*u)

A unique identifier representing your end-user, which can help *OpenAI* to monitor and detect abuse.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param u[in]** the unique identifier representing your end-user

void (**\*clearConversation**)(struct *OpenAI\_ChatCompletion* \*chatCompletion)

Clears the accumulated conversation.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

*OpenAI\_StringResponse\_t* **\*(message)**(struct *OpenAI\_ChatCompletion* \*chatCompletion, const char \*p, bool save)

Send the message for completion. Save it with the first response if selected.

**Param chatCompletion[in]** the point of *OpenAI\_ChatCompletion*

**Param p[in]** the message for completion

**Param save[in]** save it with the first response if selected

**Return** *OpenAI\_StringResponse\_t*\*

struct **OpenAI\_Edit**

Given a prompt and an instruction, the model will return an edited version of the prompt.

### Public Members

void (\***setModel**)(struct *OpenAI\_Edit* \*edit, const char \*m)

Set the model to use for edit.

**Param edit[in]** the point of OpenAI\_Edit\_t

**Param m[in]** the name of the model to use for edit

void (\***setTemperature**)(struct *OpenAI\_Edit* \*edit, float t)

Set the temperature for the edit.

**Param edit[in]** the point of OpenAI\_Edit\_t

**Param t[in]** float between 0 and 2. Higher value gives more random results.

void (\***setTopP**)(struct *OpenAI\_Edit* \*edit, float tp)

Set the top\_p for the edit.

**Param edit[in]** the point of OpenAI\_Edit\_t

**Param tp[in]** float between 0 and 1. recommended to alter this or temperature but not both.

void (\***setN**)(struct *OpenAI\_Edit* \*edit, uint32\_t n)

Set the number of edits to generate for the input and instruction.

**Param edit[in]** the point of OpenAI\_Edit\_t

**Param n[in]** the number of edits to generate for the input and instruction

*OpenAI\_StringResponse\_t* \*(\***process**)(struct *OpenAI\_Edit* \*edit, char \*instruction, char \*input)

Creates a new edit for the provided input, instruction, and parameters.

**Param edit[in]** the point of OpenAI\_Edit\_t

**Param instruction[in]** the instruction for the edit

**Param input[in]** the input text to be edited

**Return** OpenAI\_StringResponse\_t\* the edited text

struct **OpenAI\_ImageGeneration**

Creates an image given a prompt.

### Public Members

void (\***setSize**)(struct *OpenAI\_ImageGeneration* \*imageGeneration, *OpenAI\_Image\_Size* s)

Set the size of the generated images.

**Param imageGeneration[in]** the point of *OpenAI\_ImageGeneration*

**Param s[in]** the size of the generated images

void (\***setResponseFormat**)(struct *OpenAI\_ImageGeneration* \*imageGeneration, *OpenAI\_Image\_Response\_Format* rf)

Set the format in which the generated images are returned.

**Param imageGeneration[in]** the point of *OpenAI\_ImageGeneration*

**Param rf[in]** the format in which the generated images are returned

void (\***setN**)(struct *OpenAI\_ImageGeneration* \*imageGeneration, uint32\_t n)

Set the number of images to generate. Must be between 1 and 10.

**Param imageGeneration[in]** the point of *OpenAI\_ImageGeneration*

**Param n[in]** the number of images to generate

void (\***setUser**)(struct *OpenAI\_ImageGeneration* \*imageGeneration, const char \*u)

Set a unique identifier representing your end-user, which can help *OpenAI* to monitor and detect abuse.

**Param imageGeneration[in]** the point of *OpenAI\_ImageGeneration*

**Param u[in]** the unique identifier representing your end-user

*OpenAI\_ImageResponse\_t* \*(\***prompt**)(struct *OpenAI\_ImageGeneration* \*imageGeneration, char \*p)

Creates image/images from given a prompt.

**Param imageGeneration[in]** the point of *OpenAI\_ImageGeneration*

**Param p[in]** the prompt for image generation

**Return** *OpenAI\_ImageResponse\_t*\* the generated image/images

struct **OpenAI\_ImageVariation**

Creates a variation of a given image.

### Public Members

void (\***setSize**)(struct *OpenAI\_ImageVariation* \*imageVariation, *OpenAI\_Image\_Size* s)

Set the size of the generated images.

**Param imageVariation[in]** the point of *OpenAI\_ImageVariation*

**Param s[in]** the size of the generated images

void (\***setResponseFormat**)(struct *OpenAI\_ImageVariation* \*imageVariation, *OpenAI\_Image\_Response\_Format* rf)

Set the format in which the generated images are returned.

**Param imageVariation[in]** the point of *OpenAI\_ImageVariation*

**Param rf[in]** the format in which the generated images are returned

void (\***setN**)(struct *OpenAI\_ImageVariation* \*imageVariation, uint32\_t n)

Set the number of images to generate. Must be between 1 and 10.

**Param imageVariation[in]** the point of *OpenAI\_ImageVariation*

**Param n[in]** the number of images to generate

void (\***setUser**)(struct *OpenAI\_ImageVariation* \*imageVariation, const char \*u)

Set a unique identifier representing your end-user, which can help *OpenAI* to monitor and detect abuse.

**Param imageVariation[in]** the point of *OpenAI\_ImageVariation*

**Param u[in]** the unique identifier representing your end-user

*OpenAI\_ImageResponse\_t* \*(\***image**)(struct *OpenAI\_ImageVariation* \*imageVariation, uint8\_t \*data, size\_t len)

Creates an image variation from given image data.

**Param imageVariation[in]** the point of *OpenAI\_ImageVariation*

**Param data[in]** the input image data

**Param len[in]** the length of the input image data



**Return** `OpenAI_ImageResponse_t*` the generated image variation

struct **OpenAI\_ImageEdit**

Creates an edited or extended image given an original image and a prompt.

### Public Members

void (\***setPrompt**)(struct *OpenAI\_ImageEdit* \*imageEdit, const char \*p)

Set the prompt for the image edit.

**Param imageEdit[in]** the point of `OpenAI_ImageEdit_t`

**Param p[in]** the prompt for the image edit

void (\***setSize**)(struct *OpenAI\_ImageEdit* \*imageEdit, *OpenAI\_Image\_Size* s)

Set the size of the generated images.

**Param imageEdit[in]** the point of `OpenAI_ImageEdit_t`

**Param s[in]** the size of the generated images

void (\***setResponseFormat**)(struct *OpenAI\_ImageEdit* \*imageEdit, *OpenAI\_Image\_Response\_Format* rf)

Set the format in which the generated images are returned.

**Param imageEdit[in]** the point of `OpenAI_ImageEdit_t`

**Param rf[in]** the format in which the generated images are returned

void (\***setN**)(struct *OpenAI\_ImageEdit* \*imageEdit, uint32\_t n)

Set the number of images to generate. Must be between 1 and 10.

**Param imageEdit[in]** the point of `OpenAI_ImageEdit_t`

**Param n[in]** the number of images to generate

void (\***setUser**)(struct *OpenAI\_ImageEdit* \*imageEdit, const char \*u)

Set a unique identifier representing your end-user, which can help *OpenAI* to monitor and detect abuse.

**Param imageEdit[in]** the point of `OpenAI_ImageEdit_t`

**Param u[in]** the unique identifier representing your end-user

*OpenAI\_ImageResponse\_t* \*(\***image**)(struct *OpenAI\_ImageEdit* \*imageEdit, uint8\_t \*data, size\_t len, uint8\_t \*mask\_data, size\_t mask\_len)

Creates an edited or extended image given an original image, a mask, and a prompt.

**Param imageEdit[in]** the point of `OpenAI_ImageEdit_t`

**Param data[in]** the input image data

**Param len[in]** the length of the input image data

**Param mask\_data[in]** the input mask data

**Param mask\_len[in]** the length of the input mask data

**Return** `OpenAI_ImageResponse_t*` the edited or extended image

struct **OpenAI\_AudioTranscription**

Transcribes audio into the input language.

### Public Members

void (\***setPrompt**)(struct *OpenAI\_AudioTranscription* \*audioTranscription, const char \*p)

Set the prompt for the audio transcription.

**Param audioTranscription[in]** the point of OpenAI\_AudioTranscription\_t  
**Param p[in]** the prompt for the audio transcription

void (\***setResponseFormat**)(struct *OpenAI\_AudioTranscription* \*audioTranscription,  
*OpenAI\_Audio\_Response\_Format* rf)

Set the format of the transcript output.

**Param audioTranscription[in]** the point of OpenAI\_AudioTranscription\_t  
**Param rf[in]** the format of the transcript output

void (\***setTemperature**)(struct *OpenAI\_AudioTranscription* \*audioTranscription, float t)

Set the temperature for the audio transcription.

**Param audioTranscription[in]** the point of OpenAI\_AudioTranscription\_t  
**Param t[in]** float between 0 and 1

void (\***setLanguage**)(struct *OpenAI\_AudioTranscription* \*audioTranscription, const char \*l)

Set the language of the input audio.

**Param audioTranscription[in]** the point of OpenAI\_AudioTranscription\_t  
**Param l[in]** the language in ISO-639-1 format of the input audio. NULL for Auto.

char \*(\***file**)(struct *OpenAI\_AudioTranscription* \*audioTranscription, uint8\_t \*data, size\_t len,  
*OpenAI\_Audio\_Input\_Format* f)

Transcribe an audio file.

**Param audioTranscription[in]** the point of OpenAI\_AudioTranscription\_t  
**Param data[in]** the input audio data  
**Param len[in]** the length of the input audio data  
**Param f[in]** the format of the input audio data  
**Return** char\* the transcribed text, you should free it after use.

struct **OpenAI\_AudioSpeech**

Given a list of messages comprising a conversation, the model will return a response.

### Public Members

void (\***setModel**)(struct *OpenAI\_AudioSpeech* \*createSpeech, const char \*m)

Set the model to use for completion.

**Param createSpeech[in]** the point of OpenAI\_SpeechResponse\_t  
**Param m[in]** the name of the model to use for audio response

void (\***setVoice**)(struct *OpenAI\_AudioSpeech* \*createSpeech, const char \*m)

Set the voice to use for completion.

**Param createSpeech[in]** the point of OpenAI\_SpeechResponse\_t  
**Param m[in]** the name of the model to use for audio response

void (\***setSpeed**)(struct *OpenAI\_AudioSpeech* \*createSpeech, float t)

Set the speed of the output audio.

**Param createSpeech[in]** the point of OpenAI\_SpeechResponse\_t

**Param t[in]** float between 0.25 to 4.0

void (\***setResponseFormat**)(struct *OpenAI\_AudioSpeech* \*createSpeech,  
*OpenAI\_Audio\_Output\_Format* rf)

Set the format of the output.

**Param createSpeech[in]** the point of OpenAI\_SpeechResponse\_t

**Param rf[in]** the format of the output audio

*OpenAI\_SpeechResponse\_t* \*(\***speech**)(struct *OpenAI\_AudioSpeech* \*createSpeech, char \*p)

Send the message for completion. Save it with the first response if selected.

**Param createSpeech[in]** the point of OpenAI\_SpeechResponse\_t

**Param p[in]** the message for audio generation

**Return** \*

struct **OpenAI\_AudioTranslation**

Translates audio into English.

### Public Members

void (\***setPrompt**)(struct *OpenAI\_AudioTranslation* \*audioTranslation, const char \*p)

Set the prompt for the audio translation.

**Param audioTranslation[in]** the point of OpenAI\_AudioTranslation\_t

**Param p[in]** the prompt for the audio translation

void (\***setResponseFormat**)(struct *OpenAI\_AudioTranslation* \*audioTranslation,  
*OpenAI\_Audio\_Response\_Format* rf)

Set the format of the transcript output.

**Param audioTranslation[in]** the point of OpenAI\_AudioTranslation\_t

**Param rf[in]** the format of the transcript output

void (\***setTemperature**)(struct *OpenAI\_AudioTranslation* \*audioTranslation, float t)

Set the temperature for the audio translation.

**Param audioTranslation[in]** the point of OpenAI\_AudioTranslation\_t

**Param t[in]** float between 0 and 2. Higher value gives more random results.

char \*(\***file**)(struct *OpenAI\_AudioTranslation* \*audioTranslation, uint8\_t \*data, size\_t len,  
*OpenAI\_Audio\_Input\_Format* f)

Transcribe and translate an audio file into English.

**Param audioTranslation[in]** the point of OpenAI\_AudioTranslation\_t

**Param data[in]** the input audio data

**Param len[in]** the length of the input audio data

**Param f[in]** the format of the input audio data

**Return** char\* the translated text in English, you should free it after use.

struct **OpenAI**

The entry point for calling the Openai api.

## Type Definitions

typedef struct *OpenAI\_EmbeddingResponse* **OpenAI\_EmbeddingResponse\_t**

To get an embedding, send your text string to the embeddings API endpoint along with a choice of embedding model ID (e.g., text-embedding-ada-002). The response will contain an embedding, which you can extract, save, and use.

typedef struct *OpenAI\_ModerationResponse* **OpenAI\_ModerationResponse\_t**

The moderations endpoint is a tool you can use to check whether content complies with *OpenAI*'s usage policies. Developers can thus identify content that our usage policies prohibits and take action, for instance by filtering it.

typedef struct *OpenAI\_ImageResponse* **OpenAI\_ImageResponse\_t**

Save the image which is generated by *OpenAI*.

typedef struct *OpenAI\_StringResponse* **OpenAI\_StringResponse\_t**

Parse the returned json data into OpenAI\_StringResponse\_t.

typedef struct *OpenAI\_SpeechResponse* **OpenAI\_SpeechResponse\_t**

Store the returned data into a OpenAI\_SpeechResponse\_t structure.

typedef struct *OpenAI\_Completion* **OpenAI\_Completion\_t**

Given a prompt, the model will return one or more predicted completions, and can also return the probabilities of alternative tokens at each position.

typedef struct *OpenAI\_ChatCompletion* **OpenAI\_ChatCompletion\_t**

Given a list of messages comprising a conversation, the model will return a response.

typedef struct *OpenAI\_Edit* **OpenAI\_Edit\_t**

Given a prompt and an instruction, the model will return an edited version of the prompt.

typedef struct *OpenAI\_ImageGeneration* **OpenAI\_ImageGeneration\_t**

Creates an image given a prompt.

typedef struct *OpenAI\_ImageVariation* **OpenAI\_ImageVariation\_t**

Creates a variation of a given image.

typedef struct *OpenAI\_ImageEdit* **OpenAI\_ImageEdit\_t**

Creates an edited or extended image given an original image and a prompt.

typedef struct *OpenAI\_AudioTranscription* **OpenAI\_AudioTranscription\_t**

Transcribes audio into the input language.

typedef struct *OpenAI\_AudioSpeech* **OpenAI\_AudioSpeech\_t**

Given a list of messages comprising a conversation, the model will return a response.

typedef struct *OpenAI\_AudioTranslation* **OpenAI\_AudioTranslation\_t**

Translates audio into English.

typedef struct *OpenAI* **OpenAI\_t**

The entry point for calling the Openai api.

## Enumerations

enum **OpenAI\_Image\_Size**

*Values:*

enumerator **OPENAI\_IMAGE\_SIZE\_1024x1024**

enumerator **OPENAI\_IMAGE\_SIZE\_512x512**

enumerator **OPENAI\_IMAGE\_SIZE\_256x256**

enum **OpenAI\_Image\_Response\_Format**

*Values:*

enumerator **OPENAI\_IMAGE\_RESPONSE\_FORMAT\_URL**

enumerator **OPENAI\_IMAGE\_RESPONSE\_FORMAT\_B64\_JSON**

enum **OpenAI\_Audio\_Response\_Format**

*Values:*

enumerator **OPENAI\_AUDIO\_RESPONSE\_FORMAT\_JSON**

enumerator **OPENAI\_AUDIO\_RESPONSE\_FORMAT\_TEXT**

enumerator **OPENAI\_AUDIO\_RESPONSE\_FORMAT\_SRT**

enumerator **OPENAI\_AUDIO\_RESPONSE\_FORMAT\_VERBOSE\_JSON**

enumerator **OPENAI\_AUDIO\_RESPONSE\_FORMAT\_VTT**

enum **OpenAI\_Audio\_Input\_Format**

*Values:*

enumerator **OPENAI\_AUDIO\_INPUT\_FORMAT\_MP3**

enumerator **OPENAI\_AUDIO\_INPUT\_FORMAT\_MP4**

enumerator **OPENAI\_AUDIO\_INPUT\_FORMAT\_MPEG**

enumerator **OPENAI\_AUDIO\_INPUT\_FORMAT\_MPGA**

enumerator **OPENAI\_AUDIO\_INPUT\_FORMAT\_M4A**

enumerator **OPENAI\_AUDIO\_INPUT\_FORMAT\_WAV**

enumerator **OPENAI\_AUDIO\_INPUT\_FORMAT\_WEBM**

enum **OpenAI\_Audio\_Output\_Format**

*Values:*

enumerator **OPENAI\_AUDIO\_OUTPUT\_FORMAT\_MP3**

enumerator **OPENAI\_AUDIO\_OUTPUT\_FORMAT\_OPUS**

enumerator **OPENAI\_AUDIO\_OUTPUT\_FORMAT\_AAC**

enumerator **OPENAI\_AUDIO\_OUTPUT\_FORMAT\_FLAC**

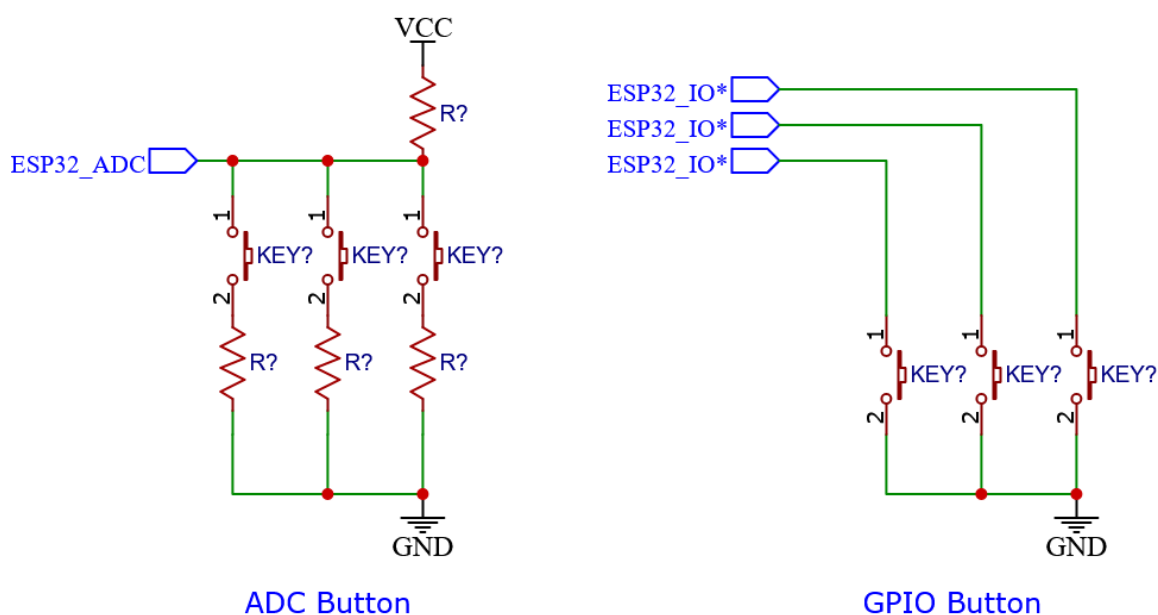


## Chapter 9

# 输入设备

### 9.1 按键

按键组件实现了 GPIO 和 ADC 两种按键，并允许同时创建两种不同的按键。下图显示了两两种按键的硬件设计：



- GPIO 按键优点有：每一个按键占用独立的 IO，之间互不影响，稳定性高；缺点有：按键数量多时占用太多 IO 资源。
- ADC 按键优点有：可多个按键共用一个 ADC 通道，占用 IO 资源少；缺点有：不能同时按下多按键，当按键因氧化等因素导致闭合电阻增大时，容易误触，稳定性不高。

#### 备注：

- GPIO 按键需注意上下拉问题，组件内部会启用芯片内部的上下拉电阻，但是在仅支持输入的 IO 内部没有电阻，**需要外部连接**。
- ADC 按键需注意电压不能超过 ADC 量程。



### 9.1.1 按键事件

每个按键拥有下表的 8 个事件：

事件	触发条件
BUTTON_PRESS_DOWN	按下
BUTTON_PRESS_UP	弹起
BUTTON_PRESS_REPEAT	按下弹起次数 $\geq 2$ 次
BUTTON_PRESS_REPEAT_DONE	重复按下结束
BUTTON_SINGLE_CLICK	按下弹起 1 次
BUTTON_DOUBLE_CLICK	按下弹起 2 次
BUTTON_MULTIPLE_CLICK	指定重复按下次数 N 次，达成时触发
BUTTON_LONG_PRESS_START	按下时间达到阈值的瞬间
BUTTON_LONG_PRESS_HOLD	长按期间一直触发
BUTTON_LONG_PRESS_UP	长按弹起
BUTTON_PRESS_REPEAT_DONE	多次按下弹起结束
BUTTON_PRESS_END	表示 button 此次检测已结束

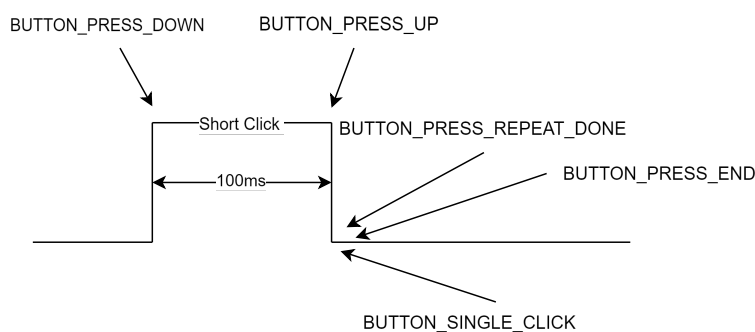
每个按键可以有 **回调**和 **轮询**两种使用方式：

- 回调：一个按键的每个事件都可以为其注册一个回调函数，产生事件时回调函数将会被调用。这种方式的效率和实时性高，不会丢失事件。
- 轮询：在程序中周期性调用 `iot_button_get_event()` 查询按键当前的事件。这种方式使用简单，适合任务简单的场合

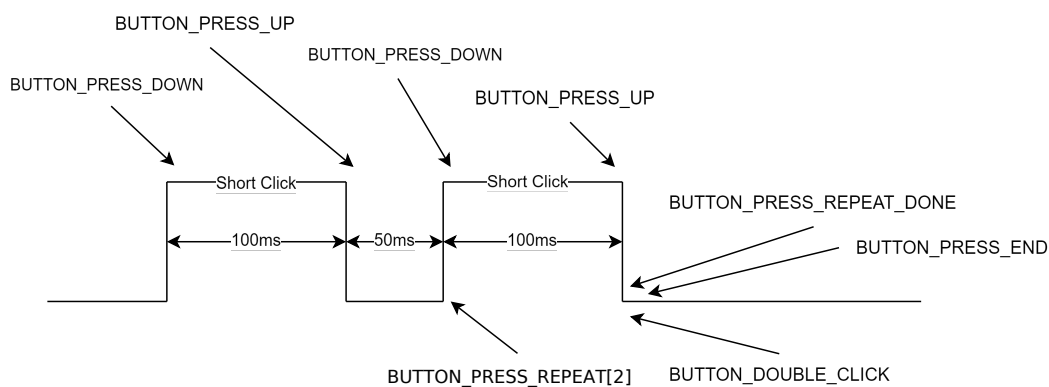
当然你也可以将以上两种方式组合使用。

**注意：**回调函数中不能有 `TaskDelay` 等阻塞的操作

## Single Click

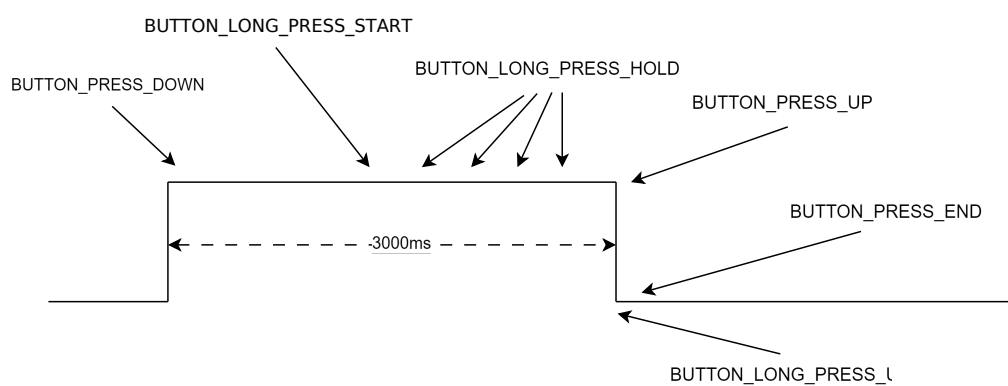


## Double Click



## LongPress

`BUTTON_LONG_PRESS_TIME_MS: 1500ms`



### 9.1.2 配置项

- `BUTTON_PERIOD_TIME_MS` : 扫描周期
- `BUTTON_DEBOUNCE_TICKS` : 消抖次数
- `BUTTON_SHORT_PRESS_TIME_MS` : 连续短按有效时间
- `BUTTON_LONG_PRESS_TIME_MS` : 长按有效时间
- `ADC_BUTTON_MAX_CHANNEL` : ADC 按钮的最大通道数

- `ADC_BUTTON_MAX_BUTTON_PER_CHANNEL` : ADC 一个通道最多的按钮数
- `ADC_BUTTON_SAMPLE_TIMES` : 每次扫描的样本数
- `BUTTON_SERIAL_TIME_MS` : 长按期间触发的 `CALLBACK` 间隔时间
- `BUTTON_LONG_PRESS_TOLERANCE_MS` : 用于设置长按的容错时间。

### 9.1.3 应用示例

#### 创建按键

```
// create gpio button
button_config_t gpio_btn_cfg = {
    .type = BUTTON_TYPE_GPIO,
    .long_press_time = CONFIG_BUTTON_LONG_PRESS_TIME_MS,
    .short_press_time = CONFIG_BUTTON_SHORT_PRESS_TIME_MS,
    .gpio_button_config = {
        .gpio_num = 0,
        .active_level = 0,
    },
};
button_handle_t gpio_btn = iot_button_create(&gpio_btn_cfg);
if(NULL == gpio_btn) {
    ESP_LOGE(TAG, "Button create failed");
}

// create adc button
button_config_t adc_btn_cfg = {
    .type = BUTTON_TYPE_ADC,
    .long_press_time = CONFIG_BUTTON_LONG_PRESS_TIME_MS,
    .short_press_time = CONFIG_BUTTON_SHORT_PRESS_TIME_MS,
    .adc_button_config = {
        .adc_channel = 0,
        .button_index = 0,
        .min = 100,
        .max = 400,
    },
};
button_handle_t adc_btn = iot_button_create(&adc_btn_cfg);
if(NULL == adc_btn) {
    ESP_LOGE(TAG, "Button create failed");
}

// create matrix keypad button
button_config_t matrix_button_cfg = {
    .type = BUTTON_TYPE_MATRIX,
    .long_press_time = CONFIG_BUTTON_LONG_PRESS_TIME_MS,
    .short_press_time = CONFIG_BUTTON_SHORT_PRESS_TIME_MS,
    .matrix_button_config = {
        .row_gpio_num = 0,
        .col_gpio_num = 1,
    }
};
button_handle_t matrix_button = iot_button_create(&matrix_button_cfg);
if(NULL == matrix_button) {
    ESP_LOGE(TAG, "Button create failed");
}
```

**备注：**当 IDF 版本大于等于 `release/5.0` 时，ADC 按钮使用的是 `ADC1`，当项目中还有其他地方使用到了 `ADC1` 时，请传入 `adc_handle` 和 `adc_channel` 来配置 ADC 按钮。

## 注册回调函数

Button 组件支持为多个事件注册回调函数，每个事件都可以注册一个回调函数，当事件发生时，回调函数将会被调用。

其中，

- `BUTTON_LONG_PRESS_START` 和 `BUTTON_LONG_PRESS_UP` 支持设置特殊的长按时间。
- `BUTTON_MULTIPLE_CLICK` 支持设置多次按下的次数。
- 简单写法

```
static void button_single_click_cb(void *arg, void *usr_data)
{
    ESP_LOGI(TAG, "BUTTON_SINGLE_CLICK");
}

iot_button_register_cb(gpio_btn, BUTTON_SINGLE_CLICK, button_single_
↪click_cb, NULL);
```

- 多个回调函数写法

```
static void button_long_press_1_cb(void *arg, void *usr_data)
{
    ESP_LOGI(TAG, "BUTTON_LONG_PRESS_START_1");
}

static void button_long_press_2_cb(void *arg, void *usr_data)
{
    ESP_LOGI(TAG, "BUTTON_LONG_PRESS_START_2");
}

button_event_config_t cfg = {
    .event = BUTTON_LONG_PRESS_START,
    .event_data.long_press.press_time = 2000,
};

iot_button_register_event_cb(gpio_btn, cfg, BUTTON_LONG_PRESS_START, ↪
↪button_long_press_1_cb, NULL);

cfg.event_data.long_press.press_time = 5000;
iot_button_register_event_cb(gpio_btn, cfg, BUTTON_LONG_PRESS_START, ↪
↪button_long_press_2_cb, NULL);
```

## 查询按键事件

```
button_event_t event;
event = iot_button_get_event(button_handle);
```

## 动态修改按键默认值

```
iot_button_set_param(btn, BUTTON_LONG_PRESS_TIME_MS, 5000);
```

## 低功耗支持

在 `light_sleep` 模式下，`esp_timer` 定时器会定时触发，导致 `cpu` 整体功耗居高不下。为了解决这个问题，`button` 组件提供了低功耗模式。

所需配置：

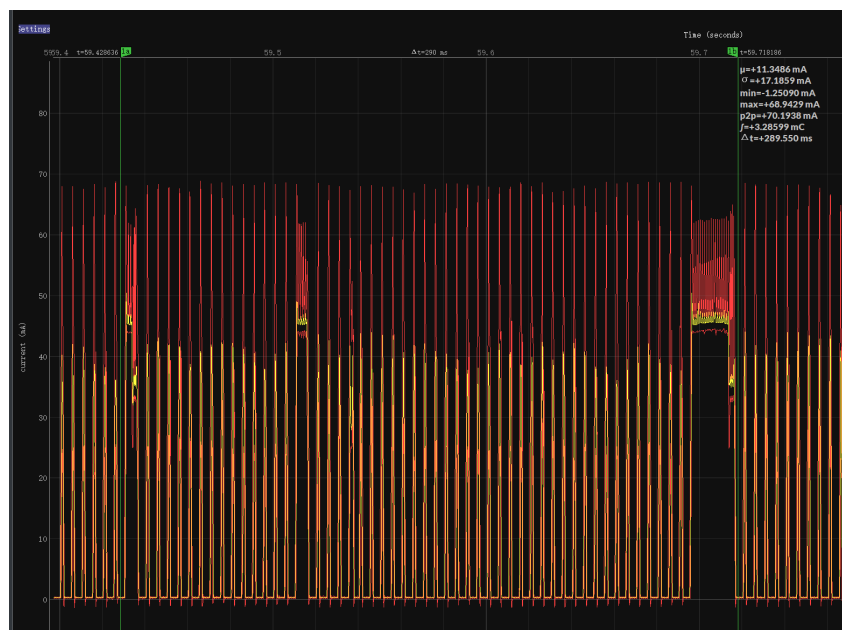
- 打开 `CONFIG_GPIO_BUTTON_SUPPORT_POWER_SAVE` 选项，会在组件中增加低功耗相关代码

- 确保创建的所有按键类型为 GPIO 按键，并且都开启了 `enable_power_save`，如存在其他按键，会导致低功耗模式失效

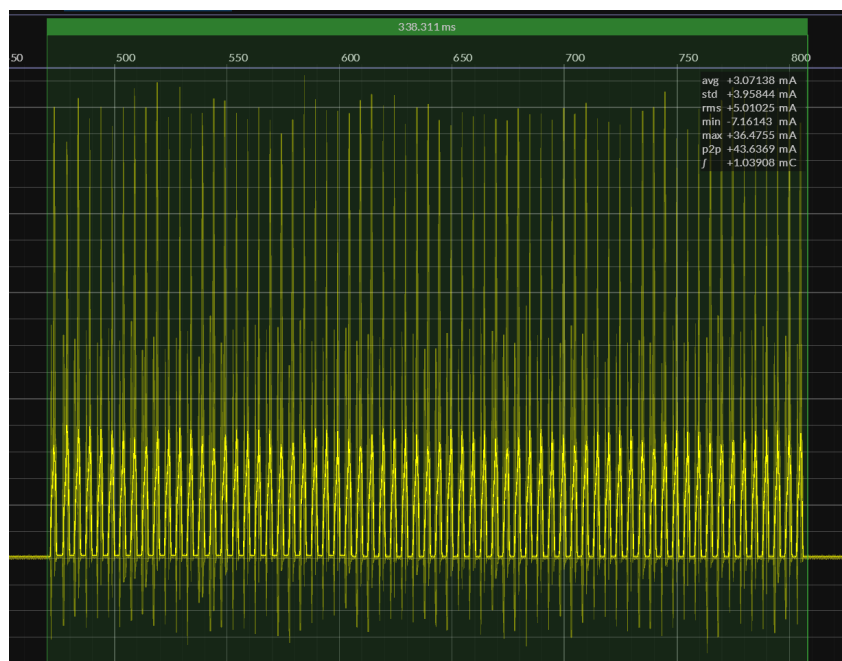
**备注：**该功能只保证 Button 组件只在使用中才唤醒 CPU，不保证 CPU 一定会进入低功耗模式

功耗对比：

- 未开启低功耗模式，按下一次按键



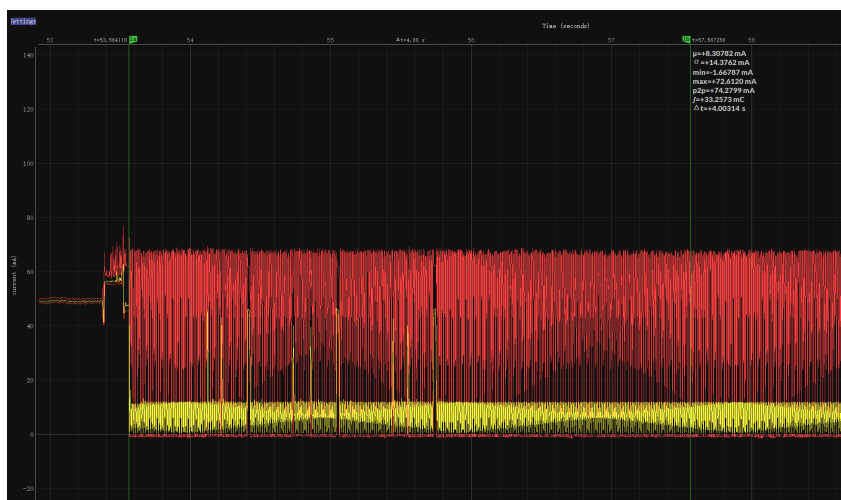
- 开启低功耗模式，按下一次按键



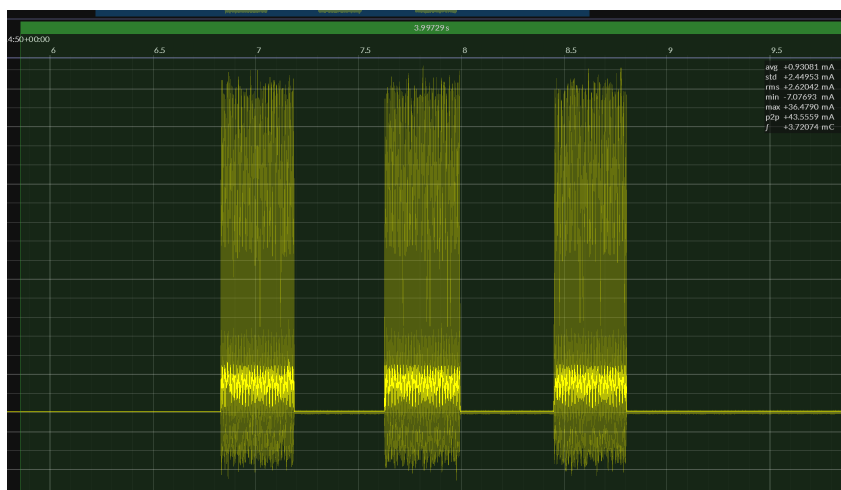
因为 GPIO 唤醒 CPU，仅支持电平触发，所以当按键为工作电平时，CPU 会支持的被唤醒，取决于按下去的时长，因此在低功耗模式下，单次按下的平均电流高于未开启低功耗模式。但是在大的工作周期中，

会比未开启低功耗模式更加省电。

- 未开启低功耗模式下，在 4s 内按下三次按键



- 低功耗模式下，在 4s 内按下三次按键



如图，低功耗模式下更加的省电。

```
button_config_t btn_cfg = {
    .type = BUTTON_TYPE_GPIO,
    .gpio_button_config = {
        .gpio_num = button_num,
        .active_level = BUTTON_ACTIVE_LEVEL,
        .enable_power_save = true,
    },
};
button_handle_t btn = iot_button_create(&btn_cfg);
```

什么时候进入 Light Sleep

- 使用 Auto Light Sleep: 会在 button 自动关闭 esp\_timer 后进入 Light Sleep
- 用户控制 Light Sleep: 需要在 enter\_power\_save\_cb 回调到来时进入 Light Sleep

```
void btn_enter_power_save(void *usr_data)
{
    ESP_LOGI(TAG, "Can enter power save now");
}

button_power_save_config_t config = {
    .enter_power_save_cb = btn_enter_power_save,
};

iot_button_register_power_save_cb(&config);
```

## 开启和关闭

组件支持在任意时刻开启和关闭。

```
// stop button
iot_button_stop();
// resume button
iot_button_resume();
```

## 9.1.4 API Reference

### Header File

- [components/button/include/iot\\_button.h](#)

### Functions

***button\_handle\_t* iot\_button\_create** (const *button\_config\_t* \*config)

Create a button.

**参数** **config** –pointer of button configuration, must corresponding the button type

**返回** A handle to the created button, or NULL in case of error.

**esp\_err\_t iot\_button\_delete** (*button\_handle\_t* btn\_handle)

Delete a button.

**参数** **btn\_handle** –A button handle to delete

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

**esp\_err\_t iot\_button\_register\_cb** (*button\_handle\_t* btn\_handle, *button\_event\_t* event, *button\_cb\_t* cb, void \*usr\_data)

Register the button event callback function.

**参数**

- **btn\_handle** –A button handle to register
- **event** –Button event
- **cb** –Callback function.
- **usr\_data** –user data

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is invalid.
- ESP\_ERR\_INVALID\_STATE The Callback is already registered. No free Space for another Callback.
- ESP\_ERR\_NO\_MEM No more memory allocation for the event

`esp_err_t iot_button_register_event_cb` ([button\\_handle\\_t](#) btn\_handle, [button\\_event\\_config\\_t](#) event\_cfg, [button\\_cb\\_t](#) cb, void \*usr\_data)

Register the button event callback function.

**参数**

- **btn\_handle** –A button handle to register
- **event\_cfg** –Button event configuration
- **cb** –Callback function.
- **usr\_data** –user data

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is invalid.
- ESP\_ERR\_INVALID\_STATE The Callback is already registered. No free Space for another Callback.
- ESP\_ERR\_NO\_MEM No more memory allocation for the event

`esp_err_t iot_button_unregister_event` ([button\\_handle\\_t](#) btn\_handle, [button\\_event\\_config\\_t](#) event\_cfg, [button\\_cb\\_t](#) cb)

Unregister the button event callback function. In case event\_data is also passed it will unregister function for that particular event\_data only.

**参数**

- **btn\_handle** –A button handle to unregister
- **event\_cfg** –Button event
- **cb** –callback to unregister

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is invalid.
- ESP\_ERR\_INVALID\_STATE The Callback was never registered with the event

`esp_err_t iot_button_unregister_cb` ([button\\_handle\\_t](#) btn\_handle, [button\\_event\\_t](#) event)

Unregister all the callbacks associated with the event.

**参数**

- **btn\_handle** –A button handle to unregister
- **event** –Button event

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is invalid.
- ESP\_ERR\_INVALID\_STATE No callbacks registered for the event

`size_t iot_button_count_cb` ([button\\_handle\\_t](#) btn\_handle)

counts total callbacks registered

**参数** **btn\_handle** –A button handle to the button

**返回**

- 0 if no callbacks registered, or 1 .. (BUTTON\_EVENT\_MAX-1) for the number of Registered Buttons.
- ESP\_ERR\_INVALID\_ARG if btn\_handle is invalid

`size_t iot_button_count_event` ([button\\_handle\\_t](#) btn\_handle, [button\\_event\\_t](#) event)

how many callbacks are registered for the event

**参数**

- **btn\_handle** –A button handle to the button
- **event** –Button event

**返回**

- 0 if no callbacks registered, or 1 .. (BUTTON\_EVENT\_MAX-1) for the number of Registered Buttons.
- ESP\_ERR\_INVALID\_ARG if btn\_handle is invalid



*button\_event\_t* **iot\_button\_get\_event** (*button\_handle\_t* btn\_handle)

Get button event.

参数 **btn\_handle** –Button handle

返回 Current button event. See *button\_event\_t*

uint8\_t **iot\_button\_get\_repeat** (*button\_handle\_t* btn\_handle)

Get button repeat times.

参数 **btn\_handle** –Button handle

返回 button pressed times. For example, double-click return 2, triple-click return 3, etc.

uint16\_t **iot\_button\_get\_ticks\_time** (*button\_handle\_t* btn\_handle)

Get button ticks time.

参数 **btn\_handle** –Button handle

返回 Actual time from press down to up (ms).

uint16\_t **iot\_button\_get\_long\_press\_hold\_cnt** (*button\_handle\_t* btn\_handle)

Get button long press hold count.

参数 **btn\_handle** –Button handle

返回 Count of trigger cb(BUTTON\_LONG\_PRESS\_HOLD)

esp\_err\_t **iot\_button\_set\_param** (*button\_handle\_t* btn\_handle, *button\_param\_t* param, void \*value)

Dynamically change the parameters of the iot button.

参数

- **btn\_handle** –Button handle
- **param** –Button parameter
- **value** –new value

返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is invalid.

uint8\_t **iot\_button\_get\_key\_level** (*button\_handle\_t* btn\_handle)

Get button key level.

参数 **btn\_handle** –Button handle

返回

- 1 if key is pressed
- 0 if key is released or invalid button handle

esp\_err\_t **iot\_button\_resume** (void)

resume button timer, if button timer is stopped. Make sure *iot\_button\_create()* is called before calling this API.

返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE timer state is invalid.

esp\_err\_t **iot\_button\_stop** (void)

stop button timer, if button timer is running. Make sure *iot\_button\_create()* is called before calling this API.

返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE timer state is invalid

## Unions

union **button\_event\_data\_t**

*#include <iot\_button.h>* Button events data.

### Public Members

struct *button\_event\_data\_t::long\_press\_t* **long\_press**

long press struct, for event BUTTON\_LONG\_PRESS\_START and BUTTON\_LONG\_PRESS\_UP

struct *button\_event\_data\_t::multiple\_clicks\_t* **multiple\_clicks**

multiple clicks struct, for event BUTTON\_MULTIPLE\_CLICK

struct **long\_press\_t**

*#include <iot\_button.h>* Long press time event data.

### Public Members

uint16\_t **press\_time**

press time(ms) for the corresponding callback to trigger

struct **multiple\_clicks\_t**

*#include <iot\_button.h>* Multiple clicks event data.

### Public Members

uint16\_t **clicks**

number of clicks, to trigger the callback

## Structures

struct **button\_event\_config\_t**

Button events configuration.

### Public Members

*button\_event\_t* **event**

button event type

*button\_event\_data\_t* **event\_data**

event data corresponding to the event

struct **button\_custom\_config\_t**

custom button configuration

### Public Members

uint8\_t **active\_level**

active level when press down

esp\_err\_t (\***button\_custom\_init**)(void \*param)

user defined button init

uint8\_t (\***button\_custom\_get\_key\_value**)(void \*param)

user defined button get key value

esp\_err\_t (\***button\_custom\_deinit**)(void \*param)

user defined button deinit

void \***priv**

private data used for custom button, MUST be allocated dynamically and will be auto freed in  
iot\_button\_delete

struct **button\_config\_t**

Button configuration.

## Public Members

*button\_type\_t* **type**

button type, The corresponding button configuration must be filled

uint16\_t **long\_press\_time**

Trigger time(ms) for long press, if 0 default to BUTTON\_LONG\_PRESS\_TIME\_MS

uint16\_t **short\_press\_time**

Trigger time(ms) for short press, if 0 default to BUTTON\_SHORT\_PRESS\_TIME\_MS

button\_gpio\_config\_t **gpio\_button\_config**

gpio button configuration

button\_matrix\_config\_t **matrix\_button\_config**

matrix key button configuration

*button\_custom\_config\_t* **custom\_button\_config**

custom button configuration

union *button\_config\_t::*[anonymous] [**anonymous**]

button configuration

## Type Definitions

typedef void (\***button\_cb\_t**)(void \*button\_handle, void \*usr\_data)

typedef void \***button\_handle\_t**

## Enumerations

enum **button\_event\_t**

Button events.

*Values:*

enumerator **BUTTON\_PRESS\_DOWN**

enumerator **BUTTON\_PRESS\_UP**

enumerator **BUTTON\_PRESS\_REPEAT**

enumerator **BUTTON\_PRESS\_REPEAT\_DONE**

enumerator **BUTTON\_SINGLE\_CLICK**

enumerator **BUTTON\_DOUBLE\_CLICK**

enumerator **BUTTON\_MULTIPLE\_CLICK**

enumerator **BUTTON\_LONG\_PRESS\_START**

enumerator **BUTTON\_LONG\_PRESS\_HOLD**

enumerator **BUTTON\_LONG\_PRESS\_UP**

enumerator **BUTTON\_PRESS\_END**

enumerator **BUTTON\_EVENT\_MAX**

enumerator **BUTTON\_NONE\_PRESS**

enum **button\_type\_t**

Supported button type.

*Values:*

enumerator **BUTTON\_TYPE\_GPIO**

enumerator **BUTTON\_TYPE\_ADC**

enumerator **BUTTON\_TYPE\_MATRIX**

enumerator **BUTTON\_TYPE\_CUSTOM**

```
enum button_param_t
```

Button parameter.

Values:

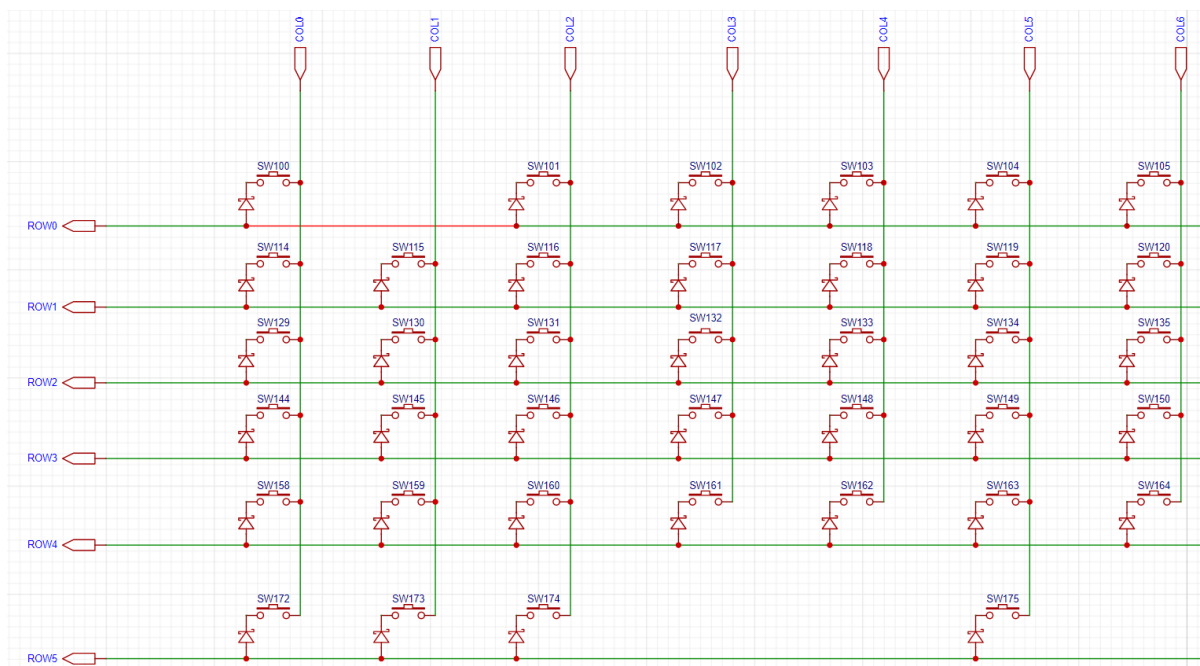
```
enumerator BUTTON_LONG_PRESS_TIME_MS
```

```
enumerator BUTTON_SHORT_PRESS_TIME_MS
```

```
enumerator BUTTON_PARAM_MAX
```

## 9.2 键盘扫描

[键盘扫描组件](#) 实现了快速高效的键盘扫描，支持按键消抖，按键松开按下事件上报，以及组合键。该组件采用了矩阵按键的行列扫描，通过特殊的电路设计，实现了全键无冲的电路检测。



- 此电路中，行依次输出高电平，检测列是否有高电平，如果有，则表示该按键被按下。

### 备注:

- 由于该组件逻辑并不会交换行列扫描，所以不适用于传统的行列扫描电路，只适用于键盘的全键无冲电路。

### 9.2.1 组件事件

- `KBD_EVENT_PRESSED`: 当有按键状态变化时，上报数据。
  - `key_pressed_num`: 按键按下的数量。
  - `key_release_num`: 按键松开的数量。

- *key\_change\_num*: 和上一次状态相比, 状态变化的按键数量。**>0** 按下按键增加, **<0** 按下按键减少。
- *key\_data*: 当前按下的按键信息, 按下按键的位置 (x, y), 索引从小到大为按下的顺序, 索引越小越早按下
- *key\_release\_data*: 和上一次相比松开的按键信息, 松开按键的位置 (x, y)。
- *KBD\_EVENT\_COMBINATION*: 组合按键事件, 当组合键被按下时, 触发回调。
  - *key\_num*: 组合按键的数量
  - *key\_data*: 组合按键的位置信息, 如设置组合键 (1, 1) (2, 2), 那么必须先按下 (1, 1) 再按下 (2, 2) 才会触发组合按键事件。组合按键只触发正向增长的组合键。

## 9.2.2 应用示例

### 初始化键盘扫描

```
keyboard_btn_config_t cfg = {
    .output_gpios = (int[])
    {
        40, 39, 38, 45, 48, 47
    },
    .output_gpio_num = 6,
    .input_gpios = (int[])
    {
        21, 14, 13, 12, 11, 10, 9, 4, 5, 6, 7, 15, 16, 17, 18
    },
    .input_gpio_num = 15,
    .active_level = 1,
    .debounce_ticks = 2,
    .ticks_interval = 500,          // us
    .enable_power_save = false,    // enable power save
};
keyboard_btn_handle_t kbd_handle = NULL;
keyboard_button_create(&cfg, &kbd_handle);
```

### 注册回调函数

- *KBD\_EVENT\_PRESSED* 事件的注册如下

```
keyboard_btn_cb_config_t cb_cfg = {
    .event = KBD_EVENT_PRESSED,
    .callback = keyboard_cb,
};
keyboard_button_register_cb(kbd_handle, cb_cfg, NULL);
```

- *KBD\_EVENT\_COMBINATION* 事件的注册如下, 需要传递组合键的信息通过 *combination* 成员

```
keyboard_btn_cb_config_t cb_cfg = {
    .event = KBD_EVENT_COMBINATION,
    .callback = keyboard_combination_cb1,
    .event_data.combination.key_num = 2,
    .event_data.combination.key_data = (keyboard_btn_data_t[]) {
        {5, 1},
        {1, 1},
    },
};
keyboard_button_register_cb(kbd_handle, cb_cfg, NULL);
```

---

**备注：** 此外事件都支持注册多个回调，在注册多个回调时，最好保存 `keyboard_btn_cb_handle_t *rtn_cb_hdl` 以方便后续解绑指定回调。

---

### 按键扫描效率

- 测试使用 *ESP32S3* 芯片扫描 *5\*16* 的矩阵键盘，最大扫描速率可达 20K。

### 低功耗支持

- 在初始化时将 *enable\_power\_save* 设置为 *true*，即可开启低功耗模式，此模式将在没有按键改变的时候不进行按键扫描，CPU 同时进入休眠状态，在有按键按下时唤醒 CPU。

---

**备注：** 该功能只保证不占用 CPU，不保证 CPU 一定会进入低功耗模式。且目前只支持 *Light Sleep* 模式。

---

## 9.2.3 API Reference

### Header File

- [components/keyboard\\_button/include/keyboard\\_button.h](#)

### Functions

`esp_err_t keyboard_button_create` (*keyboard\_btn\_config\_t* \*kbd\_cfg, *keyboard\_btn\_handle\_t* \*kbd\_handle)

Create a keyboard instance.

#### 参数

- **kbd\_cfg** –keyboard configuration
- **kbd\_handle** –keyboard handle

#### 返回

- *ESP\_OK* on success
- *ESP\_ERR\_INVALID\_ARG* Arguments is invalid.
- *ESP\_ERR\_NO\_MEM* No more memory allocation.

`esp_err_t keyboard_button_delete` (*keyboard\_btn\_handle\_t* kbd\_handle)

Delete the keyboard instance.

参数 **kbd\_handle** –keyboard handle

#### 返回

- *ESP\_OK* on success
- *ESP\_ERR\_INVALID\_ARG* Arguments is invalid.

`esp_err_t keyboard_button_register_cb` (*keyboard\_btn\_handle\_t* kbd\_handle, *keyboard\_btn\_cb\_config\_t* cb\_cfg, *keyboard\_btn\_cb\_handle\_t* \*rtn\_cb\_hdl)

Register the button callback function.

#### 参数

- **kbd\_handle** –keyboard handle
- **cb\_cfg** –callback configuration
- **rtn\_cb\_hdl** –callback handle for unregister

#### 返回

- *ESP\_OK* on success

- ESP\_ERR\_INVALID\_ARG Arguments is invalid.
- ESP\_ERR\_NO\_MEM No more memory allocation for the event

esp\_err\_t **keyboard\_button\_unregister\_cb** ([keyboard\\_btn\\_handle\\_t](#) kbd\_handle, [keyboard\\_btn\\_event\\_t](#) event, [keyboard\\_btn\\_cb\\_handle\\_t](#) rtn\_cb\_hdl)

Unregister the button callback function.

---

**备注:** If only the event is provided, all callbacks associated with this event will be canceled. If rtn\_cb\_hdl is provided, only the specified callback will be unregistered.

---

#### 参数

- **kbd\_handle** –keyboard handle
- **event** –event type
- **rtn\_cb\_hdl** –callback handle for unregister

#### 返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is invalid.
- ESP\_ERR\_NO\_MEM No more memory allocation for the event

esp\_err\_t **keyboard\_button\_get\_index\_by\_gpio** ([keyboard\\_btn\\_handle\\_t](#) kbd\_handle, uint32\_t gpio\_num, kbd\_gpio\_mode\_t gpio\_mode, uint32\_t \*index)

Get index by gpio number.

#### 参数

- **kbd\_handle** –keyboard handle
- **gpio\_num** –gpio number
- **gpio\_mode** –gpio mode, input or output
- **index** –return index

#### 返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is invalid.
- ESP\_ERR\_NOT\_FOUND The gpio number is not found.

esp\_err\_t **keyboard\_button\_get\_gpio\_by\_index** ([keyboard\\_btn\\_handle\\_t](#) kbd\_handle, uint32\_t index, kbd\_gpio\_mode\_t gpio\_mode, uint32\_t \*gpio\_num)

Get gpio number by index.

#### 参数

- **kbd\_handle** –keyboard handle
- **index** –index
- **gpio\_mode** –gpio mode, input or output
- **gpio\_num** –return gpio number

#### 返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is invalid.
- ESP\_ERR\_NOT\_FOUND The index is not found.

## Unions

union **keyboard\_btn\_event\_data\_t**

*#include <keyboard\_button.h>* keyboard button event data

## Public Members



struct *keyboard\_btn\_event\_data\_t::combination\_t* **combination**  
combination event

struct **combination\_t**  
*#include <keyboard\_button.h>* combination event data eg: Set key\_data = {(1,1), (2,2)} means that the button sequence is (1,1) -> (2,2)

### Public Members

uint32\_t **key\_num**  
Number of keys

*keyboard\_btn\_data\_t* \***key\_data**  
Array, contains key codes by index. The button sequence is also provided through this

## Structures

struct **keyboard\_btn\_data\_t**  
keyboard button data

### Public Members

uint8\_t **output\_index**  
key position' s output gpio number

uint8\_t **input\_index**  
key position' s input gpio number

struct **keyboard\_btn\_report\_t**  
keyboard button report data

### Public Members

int **key\_change\_num**  
Number of key changes

uint32\_t **key\_pressed\_num**  
Number of keys pressed

uint32\_t **key\_release\_num**  
Number of keys released

*keyboard\_btn\_data\_t* \***key\_data**  
Array, contains key codes

*keyboard\_btn\_data\_t* \***key\_release\_data**

Array, contains key codes

struct **keyboard\_btn\_cb\_config\_t**

keyboard button callback config

### Public Members

*keyboard\_btn\_event\_t* **event**

Event type

*keyboard\_btn\_event\_data\_t* **event\_data**

Event data

*keyboard\_btn\_callback\_t* **callback**

Callback function

void \***user\_data**

Callback user data

struct **keyboard\_btn\_config\_t**

keyboard button config

### Public Members

const int \***output\_gpios**

Array, contains output GPIO numbers used by rom/col line

const int \***input\_gpios**

Array, contains input GPIO numbers used by rom/col line

uint32\_t **output\_gpio\_num**

output\_gpios array size

uint32\_t **input\_gpio\_num**

input\_gpios array size

uint32\_t **active\_level**

active level for the input gpios

uint32\_t **debounce\_ticks**

debounce time in ticks

uint32\_t **ticks\_interval**

interval time in us

bool **enable\_power\_save**

enable power save mode

UBaseType\_t **priority**

FreeRTOS task priority

BaseType\_t **core\_id**

ESP32 core ID

## Type Definitions

typedef struct keyboard\_btn\_t \***keyboard\_btn\_handle\_t**

keyboard handle type

typedef void \***keyboard\_btn\_cb\_handle\_t**

keyboard callback handle for unregister

typedef void (\***keyboard\_btn\_callback\_t**)(*keyboard\_btn\_handle\_t* kbd\_handle, *keyboard\_btn\_report\_t* kbd\_report, void \*user\_data)

## Enumerations

enum **keyboard\_btn\_event\_t**

Keyboard button event.

*Values:*

enumerator **KBD\_EVENT\_PRESSED**

Report all currently pressed keys when a key is either pressed or released.

enumerator **KBD\_EVENT\_COMBINATION**

When the component buttons are pressed in sequence, report.

enumerator **KBD\_EVENT\_MAX**

## 9.3 旋钮

**Knob** 是提供软件 PCNT 的组件，可以用在没有 PCNT 硬件功能的芯片（esp32c2, esp32c3）上。使用 Knob 可以快速适配物理编码器，如 EC11 编码器。

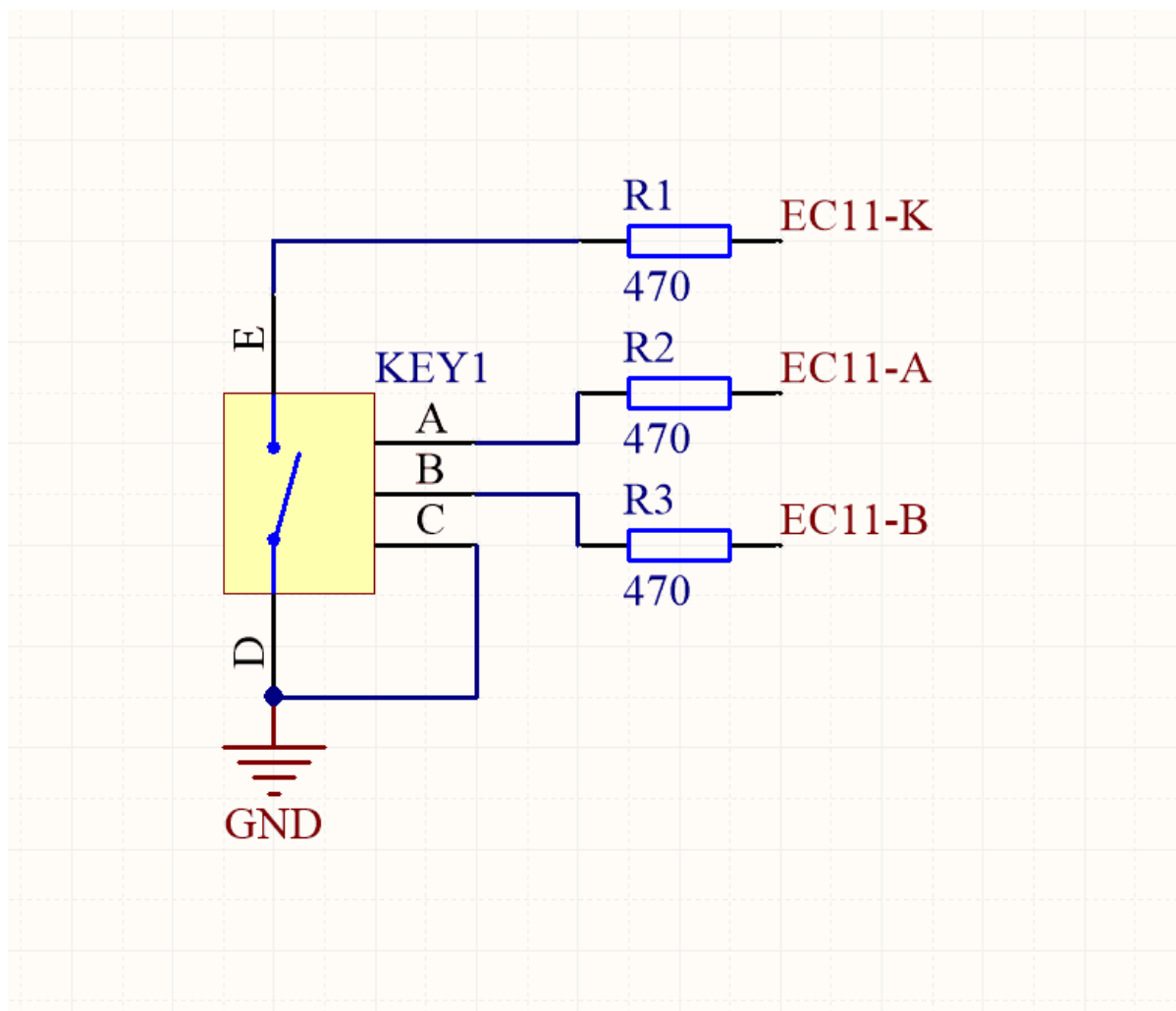
### 9.3.1 适用的场景

适用于每秒脉冲数小于 30 个的低速旋钮计数场景。例如 EC11 编码器。适用于对脉冲数不要求 100% 准确的场景。

**备注：**如需精确或快速的脉冲计数，请使用 [硬件 PCNT 功能](#)。硬件 PCNT 支持的芯片 ESP32, ESP32-C6, ESP32-H2, ESP32-S2, ESP32-S3。

### 9.3.2 硬件设计

旋转编码器的参考设计如下：



### 9.3.3 旋钮事件

每个旋钮拥有下表的 5 个事件：

事件	触发条件
KNOB_LEFT	左旋
KNOB_RIGHT	右旋
KNOB_H_LIM	计数达到最高限制
KNOB_L_LIM	计数达到最低限制
KNOB_ZERO	计数变成零

每个旋钮都可以有 [回调](#)的使用方式：

- 回调：一个旋钮的每个事件都可以为其注册一个回调函数，产生事件时回调函数将会被调用。这种方式的效率和实时性高，不会丢失事件。

**注意：** 回调函数中不能有 TaskDelay 等阻塞的操作

### 9.3.4 配置项

- KNOB\_PERIOD\_TIME\_MS : 扫描周期
- KNOB\_DEBOUNCE\_TICKS : 消抖次数
- KNOB\_HIGH\_LIMIT : 旋钮所能计数的最高数字
- KNOB\_LOW\_LIMIT : 旋钮所能计数的最低数字

### 9.3.5 应用示例

#### 创建旋钮

```
// create knob
knob_config_t cfg = {
    .default_direction = 0,
    .gpio_encoder_a = GPIO_KNOB_A,
    .gpio_encoder_b = GPIO_KNOB_B,
};
s_knob = iot_knob_create(&cfg);
if(NULL == s_knob) {
    ESP_LOGE(TAG, "knob create failed");
}
```

#### 注册回调函数

```
static void _knob_left_cb(void *arg, void *data)
{
    ESP_LOGI(TAG, "KNOB: KNOB_LEFT, count_value:%"PRIu32", iot_knob_get_count_
↪value((button_handle_t) arg));
}
iot_knob_register_cb(s_knob, KNOB_LEFT, _knob_left_cb, NULL);
```

#### 低功耗支持

在 light\_sleep 模式下，esp\_timer 定时器会唤醒 CPU，导致功耗居高不下，Knob 组件提供了通过 GPIO 电平唤醒的低功耗方案。

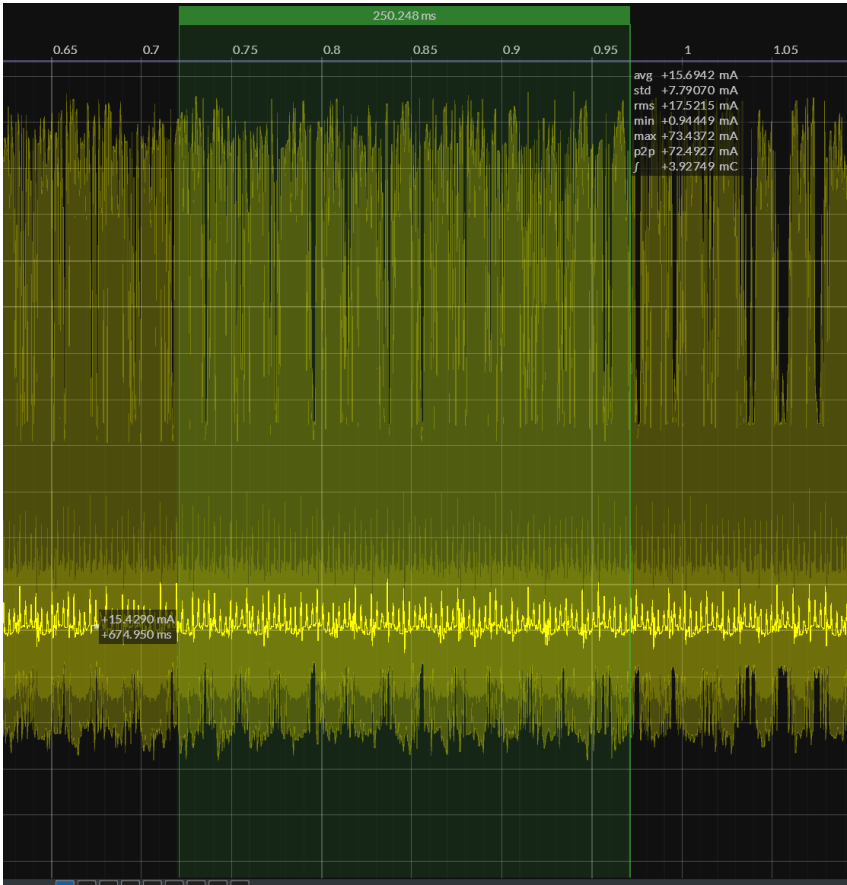
所需配置：

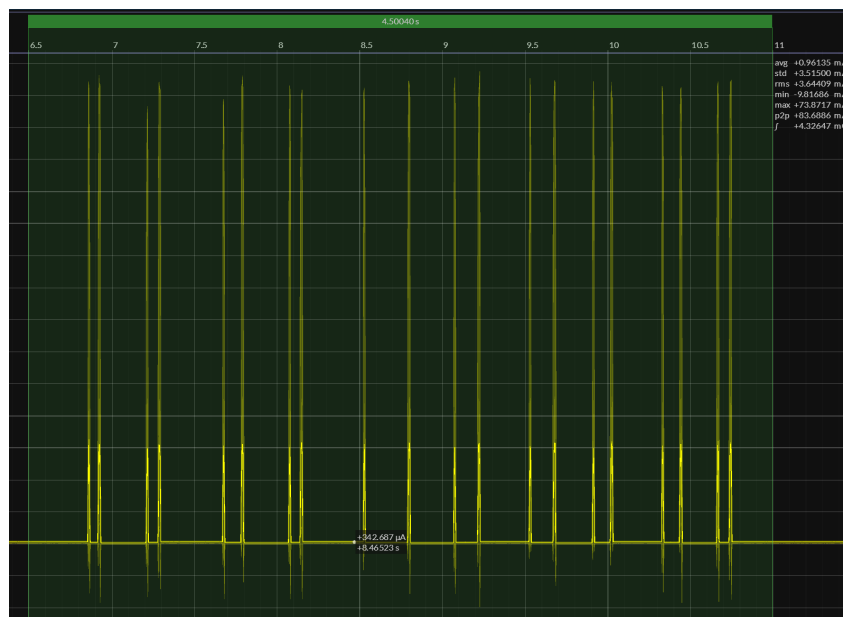
- 在 knob\_config\_t 中打开 enable\_power\_save 选项

功耗对比：

- 未开启低功耗模式，在 250ms 内旋转一次
- 开启低功耗模式，在 250ms 内旋转一次
- 开启低功耗模式，在 4.5s 内旋转十次

低功耗模式下的旋钮响应迅速，且功耗更低





## 开启和关闭

组件支持在任意时刻开启和关闭。

```
// stop knob
iot_knob_stop();
// resume knob
iot_knob_resume();
```

## 9.3.6 API Reference

### Header File

- [components/knob/include/iot\\_knob.h](#)

### Functions

*knob\_handle\_t* **iot\_knob\_create** (const *knob\_config\_t* \*config)

create a knob

**参数 config** –pointer of knob configuration

**返回** A handle to the created knob

esp\_err\_t **iot\_knob\_delete** (*knob\_handle\_t* knob\_handle)

Delete a knob.

**参数 knob\_handle** –A knob handle to delete

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

esp\_err\_t **iot\_knob\_register\_cb** (*knob\_handle\_t* knob\_handle, *knob\_event\_t* event, *knob\_cb\_t* cb, void \*usr\_data)

Register the knob event callback function.

**参数**

- **knob\_handle** –A knob handle to register

- **event** –Knob event
- **cb** –Callback function
- **usr\_data** –user data

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

esp\_err\_t **iot\_knob\_unregister\_cb** (*knob\_handle\_t* knob\_handle, *knob\_event\_t* event)

Unregister the knob event callback function.

**参数**

- **knob\_handle** –A knob handle to register
- **event** –Knob event

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

*knob\_event\_t* **iot\_knob\_get\_event** (*knob\_handle\_t* knob\_handle)

Get knob event.

**参数 knob\_handle** –A knob handle to register

**返回** knob\_event\_t Knob event

int **iot\_knob\_get\_count\_value** (*knob\_handle\_t* knob\_handle)

Get knob count value.

**参数 knob\_handle** –A knob handle to register

**返回** int count\_value

esp\_err\_t **iot\_knob\_clear\_count\_value** (*knob\_handle\_t* knob\_handle)

Clear knob count value to zero.

**参数 knob\_handle** –A knob handle to register

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

esp\_err\_t **iot\_knob\_resume** (void)

resume knob timer, if knob timer is stopped. Make sure iot\_knob\_create() is called before calling this API.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE timer state is invalid.

esp\_err\_t **iot\_knob\_stop** (void)

stop knob timer, if knob timer is running. Make sure iot\_knob\_create() is called before calling this API.

**返回**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE timer state is invalid

## Structures

struct **knob\_config\_t**

Knob config.

## Public Members



`uint8_t default_direction`  
0:positive increase 1:negative increase

`uint8_t gpio_encoder_a`  
Encoder Pin A

`uint8_t gpio_encoder_b`  
Encoder Pin B

`bool enable_power_save`  
Enable power save mode

### Type Definitions

`typedef void (*knob_cb_t)(void*, void*)`

`typedef void *knob_handle_t`

### Enumerations

`enum knob_event_t`

Knob events.

*Values:*

enumerator **KNOB\_LEFT**

EVENT: Rotate to the left

enumerator **KNOB\_RIGHT**

EVENT: Rotate to the right

enumerator **KNOB\_H\_LIM**

EVENT: Count reaches maximum limit

enumerator **KNOB\_L\_LIM**

EVENT: Count reaches the minimum limit

enumerator **KNOB\_ZERO**

EVENT: Count back to 0

enumerator **KNOB\_EVENT\_MAX**

EVENT: Number of events

enumerator **KNOB\_NONE**

EVENT: No event

## 9.4 触摸屏驱动

触摸屏现已是显示屏应用中的标配，ESP-IoT-Solution 提供了常见类型的触摸屏驱动，已支持如下控制芯片：

电阻触摸屏	电容触摸屏
XPT2046	FT5216
NS2016	FT5436
	FT6336
	FT5316

上面列出的电容触摸屏控制芯片通常可使用 FT5x06 驱动。

与屏幕驱动相似，为了方便移植到不同的 GUI 库，将一部分通用的函数封装到了一个 `touch_panel_driver_t` 结构体中。完成初始化后将通过调用结构体里面的函数完成对触摸屏的操作，而无需关心是具体哪一个触摸屏型号。

### 9.4.1 校准触摸屏

在实际应用中，电阻触摸屏必须在使用前进行校准，而电容触摸屏则一般由控制芯片完成该工作，无需额外的校准步骤。驱动中已经集成了电阻触摸屏的校准算法，校准过程使用了三个点来校准，用一个点来验证，当最后验证的误差大于某个阈值将导致校准失败，然后自动重新进行校准，直到校准成功。

调用校准函数 `calibration_run()` 将会在屏幕上开始校准的过程，校准完成后，参数将保存在 NVS 中用于下次启动，避免每次使用前的重复校准。

### 9.4.2 触摸屏的按下

不论是电阻还是电容触摸屏，通常的触摸屏控制芯片会有一个用于通知触摸事件的中断引脚。但是驱动中没有使用该信号，一方面是因为对于有屏幕的应用需要尽量节省出 IO 给其他外设；另一方面是触摸控制器给出的该信号不如程序通过寄存器数据判断的准确性高。

对于电阻触摸屏来说，判断按下的依据是 Z 方向的压力大于配置的阈值；对于电容触摸屏则是判断至少有一个触摸点存在。

### 9.4.3 触摸屏的旋转

触摸屏具有与显示屏一样的 8 个方向，定义在 `touch_panel_dir_t` 中。这里的旋转是通过软件换算来实现的，通常把二者的方向设置为相同。但这并不是一成不变的，例如：在使用电容触摸屏时，有可能触摸屏固有的方向与显示屏原始显示方向不一致，如果简单的将这两个方向设置为相同后，将无法正确的点击屏幕内容，这时需要根据实际情况调整。

触摸屏的分辨率设置也是很重要的，因为触摸屏旋转后的换算依赖于触摸屏的宽和高分辨率大小，设置不当将无法得到正确的旋转效果。

---

**备注：**在使用电阻屏时，由于电阻屏在每个方向上的电阻值不一定均匀分布，这会导致经过旋转换算后触摸位置的不准确，所以建议电阻屏校准后不再进行旋转操作。

---

### 9.4.4 应用示例

## 初始化触摸屏

```
touch_panel_driver_t touch; // a touch panel driver

i2c_config_t i2c_conf = {
    .mode = I2C_MODE_MASTER,
    .sda_io_num = 35,
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_io_num = 36,
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .master.clk_speed = 100000,
};
i2c_bus_handle_t i2c_bus = i2c_bus_create(I2C_NUM_0, &i2c_conf);

touch_panel_config_t touch_cfg = {
    .interface_i2c = {
        .i2c_bus = i2c_bus,
        .clk_freq = 100000,
        .i2c_addr = 0x38,
    },
    .interface_type = TOUCH_PANEL_IFACE_I2C,
    .pin_num_int = -1,
    .direction = TOUCH_DIR_LRTB,
    .width = 800,
    .height = 480,
};

/* Initialize touch panel controller FT5x06 */
touch_panel_find_driver(TOUCH_PANEL_CONTROLLER_FT5X06, &touch);
touch.init(&touch_cfg);

/* start to run calibration */
touch.calibration_run(&lcd, false);
```

### 备注:

- 当使用的是电容触摸屏时，调用校准函数将直接返回 ESP\_OK。
- 默认情况下只打开了 FT5x06 触摸屏的驱动，如果要使用其他的驱动，需要在 menuconfig -> Component config -> Touch Screen Driver -> Choose Touch Screen Driver 中使能对应驱动。

## 获取触摸屏是否按下及其触点坐标

```
touch_panel_points_t points;
touch.read_point_data(&points);
int32_t x = points.curx[0];
int32_t y = points.cury[0];
if(TOUCH_EVT_PRESS == points.event) {
    ESP_LOGI(TAG, "Pressed, Touch point at (%d, %d)", x, y);
}
```

## 9.4.5 API 参考

### Header File

- [components/display/touch\\_panel/touch\\_panel.h](#)

## Functions

esp\_err\_t **touch\_panel\_find\_driver** (*touch\_panel\_controller\_t* controller, *touch\_panel\_driver\_t* \*out\_driver)

Find a touch panel controller driver.

### 参数

- **controller** –Touch panel controller to initialize
- **out\_driver** –Pointer to a touch driver

### 返回

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG Arguments is NULL.
- ESP\_ERR\_NOT\_FOUND: Touch panel controller was not found.

## Structures

struct **touch\_panel\_points\_t**

Information of touch panel.

### Public Members

*touch\_panel\_event\_t* **event**

Event of touch

uint8\_t **point\_num**

Touch point number

uint16\_t **curx**[TOUCH\_MAX\_POINT\_NUMBER]

Current x coordinate

uint16\_t **cury**[TOUCH\_MAX\_POINT\_NUMBER]

Current y coordinate

struct **touch\_panel\_config\_t**

Configuration of touch panel.

### Public Members

*i2c\_bus\_handle\_t* **i2c\_bus**

Handle of i2c bus

int **clk\_freq**

i2c clock frequency

spi clock frequency

uint8\_t **i2c\_addr**

screen i2c slave address

struct *touch\_panel\_config\_t*::[anonymous]::[anonymous] **interface\_i2c**  
I2c interface

*spi\_bus\_handle\_t* **spi\_bus**  
Handle of spi bus

int8\_t **pin\_num\_cs**  
SPI Chip Select Pin

struct *touch\_panel\_config\_t*::[anonymous]::[anonymous] **interface\_spi**  
SPI interface

union *touch\_panel\_config\_t*::[anonymous] **[anonymous]**  
Interface configuration

*touch\_panel\_interface\_type\_t* **interface\_type**  
Interface bus type, see touch\_interface\_type\_t struct

int8\_t **pin\_num\_int**  
Interrupt pin of touch panel. NOTE: You can set to -1 for no connection with hardware. If PENIRQ is connected, set this to pin number.

*touch\_panel\_dir\_t* **direction**  
Rotate direction

uint16\_t **width**  
touch panel width

uint16\_t **height**  
touch panel height

struct **touch\_panel\_driver\_t**  
Define screen common function.

## Public Members

esp\_err\_t (\***init**)(const *touch\_panel\_config\_t* \*config)  
Initial touch panel.

**Attention** If you have been called function touch\_panel\_init() that will call this function automatically, and should not be called it again.

**Param config** Pointer to a structure with touch config arguments.

### Return

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t (\***deinit**)(void)

Deinitial touch panel.

**Return**

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t (\***calibration\_run**)(const scr\_driver\_t \*screen, bool recalibrate)

Start run touch panel calibration.

**Param screen** Screen driver for display prompts

**Param recalibrate** Is mandatory, set true to force calibrate

**Return**

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t (\***set\_direction**)(*touch\_panel\_dir\_t* dir)

Set touch rotate rotation.

**Param dir** rotate direction

**Return**

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t (\***read\_point\_data**)(*touch\_panel\_points\_t* \*point)

Get current touch information, see struct *touch\_panel\_points\_t*.

**Param point** a pointer of *touch\_panel\_points\_t* contained touch information.

**Return**

- ESP\_OK Success
- ESP\_FAIL Fail

## Macros

**TOUCH\_MAX\_POINT\_NUMBER**

max point number on touch panel

## Enumerations

enum **touch\_panel\_event\_t**

Touch events.

*Values:*

enumerator **TOUCH\_EVT\_RELEASE**

Release event

enumerator **TOUCH\_EVT\_PRESS**

Press event

enum **touch\_panel\_dir\_t**

Define all screen direction.

*Values:*

enumerator **TOUCH\_DIR\_LRTB**

From left to right then from top to bottom, this consider as the original direction of the touch panel

enumerator **TOUCH\_DIR\_LRBT**

From left to right then from bottom to top

enumerator **TOUCH\_DIR\_RLTB**

From right to left then from top to bottom

enumerator **TOUCH\_DIR\_RLBT**

From right to left then from bottom to top

enumerator **TOUCH\_DIR\_TBLR**

From top to bottom then from left to right

enumerator **TOUCH\_DIR\_BTLR**

From bottom to top then from left to right

enumerator **TOUCH\_DIR\_TBRL**

From top to bottom then from right to left

enumerator **TOUCH\_DIR\_BTRL**

From bottom to top then from right to left

enumerator **TOUCH\_DIR\_MAX**

enumerator **TOUCH\_MIRROR\_X**

Mirror X-axis

enumerator **TOUCH\_MIRROR\_Y**

Mirror Y-axis

enumerator **TOUCH\_SWAP\_XY**

Swap XY axis

enum **touch\_panel\_interface\_type\_t**

*Values:*

enumerator **TOUCH\_PANEL\_IFACE\_I2C**

I2C interface

enumerator **TOUCH\_PANEL\_IFACE\_SPI**

SPI interface

enum **touch\_panel\_controller\_t**

All supported touch panel controllers.

*Values:*

enumerator **TOUCH\_PANEL\_CONTROLLER\_FT5X06**

enumerator **TOUCH\_PANEL\_CONTROLLER\_XPT2046**

enumerator **TOUCH\_PANEL\_CONTROLLER\_NS2016**





## Chapter 10

# 红外

### 10.1 红外学习

这是基于 RMT 模块的红外学习组件，可以接收学习 38KHz 载波的红外信号。接收的信号以 raw 的形式保存和转发，不支持对红外协议的具体解析。

#### 10.1.1 应用示例

##### 创建红外学习任务

```
const ir_learn_cfg_t config = {
    .learn_count = 4,          /*!< 需要的红外学习次数 */
    .learn_gpio = GPIO_NUM_38, /*!< 红外学习 IO */
    .clk_src = RMT_CLK_SRC_DEFAULT, /*!< RMT 时钟源 */
    .resolution = 1000000,     /*!< RMT 分辨率, 1M, 1 个时钟周期 = 1 微秒 */

    .task_stack = 4096,        /*!< 红外学习任务堆栈大小 */
    .task_priority = 5,        /*!< 红外学习任务优先级 */
    .task_affinity = 1,        /*!< 红外学习任务固定到核 (-1 表示无固定) */
    .callback = cb,            /*!< 红外学习结果回调函数 */
};

ESP_ERROR_CHECK(ir_learn_new(&config, &handle));
```

##### 回调函数处理

```
void ir_learn_auto_learn_cb(ir_learn_state_t state, uint8_t sub_step, struct ir_
↪learn_sub_list_head *data)
{
    switch (state) {
        /**< 红外学习准备就绪，在成功初始化后 */
        case IR_LEARN_STATE_READY:
            ESP_LOGI(TAG, "IR Learn ready");
            break;
        /**< 红外学习退出 */
        case IR_LEARN_STATE_EXIT:
```

(下页继续)

(续上页)

```

    ESP_LOGI(TAG, "IR Learn exit");
    break;
/**< 红外学习成功 */
case IR_LEARN_STATE_END:
    ESP_LOGI(TAG, "IR Learn end");
    ir_learn_save_result(&ir_test_result, data);
    ir_learn_print_raw(data);
    ir_learn_stop(&handle);
    break;
/**< 红外学习失败 */
case IR_LEARN_STATE_FAIL:
    ESP_LOGI(TAG, "IR Learn failed, retry");
    break;
/**< 红外学习步骤, 从1开始 */
case IR_LEARN_STATE_STEP:
default:
    ESP_LOGI(TAG, "IR Learn step:[%d][%d]", state, sub_step);
    break;
}
return;
}
}

```

红外学习支持单包和多包学习, 如果学习成功, 通知 IR\_LEARN\_STATE\_END 事件。用户可自行处理学习结果, 返回学习的数据包格式见 [ir\\_learn\\_sub\\_list\\_t](#)。

学习过程会自动校验数据, 如果校验失败, 通知 IR\_LEARN\_STATE\_FAIL 事件, 校验逻辑如下:

- 每包 symbol 数量, 数量不一致, 则判定为学习失败。
- 每个电平的时间差值, 如果超过阈值, 则判定为学习失败。 (menuconfig 的 RMT\_DECODE\_MARGIN\_US 可调整阈值)

## 学习包发送

```

void ir_learn_test_tx_raw(struct ir_learn_sub_list_head *rmt_out)
{
    /**< 创建 RMT 发送通道 */
    rmt_tx_channel_config_t tx_channel_cfg = {
        .clk_src = RMT_CLK_SRC_DEFAULT,
        .resolution_hz = IR_RESOLUTION_HZ,
        .mem_block_symbols = 128, // 通道可以同时存储的 RMT 符号数量
        .trans_queue_depth = 4, // 允许在后台挂起的传输数量
        .gpio_num = IR_TX_GPIO_NUM,
    };
    rmt_channel_handle_t tx_channel = NULL;
    ESP_ERROR_CHECK(rmt_new_tx_channel(&tx_channel_cfg, &tx_channel));

    /**< 将载波调制到发送通道 */
    rmt_carrier_config_t carrier_cfg = {
        .duty_cycle = 0.33,
        .frequency_hz = 38000, // 38KHz
    };
    ESP_ERROR_CHECK(rmt_apply_carrier(tx_channel, &carrier_cfg));

    rmt_transmit_config_t transmit_cfg = {
        .loop_count = 0, // 禁止循环
    };

    /**< 注册红外编码器, 编码格式为原始格式 */
    ir_encoder_config_t raw_encoder_cfg = {
        .resolution = IR_RESOLUTION_HZ,
    };
}

```

(下页继续)

```

};
rmt_encoder_handle_t raw_encoder = NULL;
ESP_ERROR_CHECK(ir_encoder_new(&raw_encoder_cfg, &raw_encoder));

ESP_ERROR_CHECK(rmt_enable(tx_channel)); // 启用 RMT 发送通道

/**< 遍历并发送命令 */
struct ir_learn_sub_list_t *sub_it;
SLIST_FOREACH(sub_it, rmt_out, next) {
    vTaskDelay(pdMS_TO_TICKS(sub_it->timediff / 1000));

    rmt_symbol_word_t *rmt_symbols = sub_it->symbols.received_symbols;
    size_t symbol_num = sub_it->symbols.num_symbols;

    ESP_ERROR_CHECK(rmt_transmit(tx_channel, raw_encoder, rmt_symbols, symbol_
    ↪num, &transmit_cfg));
    rmt_tx_wait_all_done(tx_channel, -1); // 等待发送完成
}

/**< 停止并删除 RMT 发送通道 */
rmt_disable(tx_channel);
rmt_del_channel(tx_channel);
raw_encoder->del(raw_encoder);
}

```

## 10.1.2 API Reference

### Header File

- [components/ir/ir\\_learn/include/ir\\_learn.h](#)

### Functions

esp\_err\_t **ir\_learn\_new** (const [ir\\_learn\\_cfg\\_t](#) \*cfg, [ir\\_learn\\_handle\\_t](#) \*handle\_out)

Create new IR learn handle.

#### 参数

- **cfg** –[in] Config for IR learn
- **handle\_out** –[out] New IR learn handle

#### 返回

- ESP\_OK Device handle creation success.
- ESP\_ERR\_INVALID\_ARG Invalid device handle or argument.
- ESP\_ERR\_NO\_MEM Memory allocation failed.

esp\_err\_t **ir\_learn\_restart** ([ir\\_learn\\_handle\\_t](#) ir\_learn\_hdl)

Restart IR learn process.

参数 **ir\_learn\_hdl** –[in] IR learn handle

#### 返回

- ESP\_OK Restart process success.
- ESP\_ERR\_INVALID\_ARG Invalid device handle or argument.

esp\_err\_t **ir\_learn\_stop** ([ir\\_learn\\_handle\\_t](#) \*ir\_learn\_hdl)

Stop IR learn process.

---

备注: Delete all

---

参数 **ir\_learn\_hdl** –[in] IR learn handle

返回

- ESP\_OK Stop process success.
- ESP\_ERR\_INVALID\_ARG Invalid device handle or argument.

esp\_err\_t **ir\_learn\_add\_list\_node** (struct ir\_learn\_list\_head \*learn\_head)

Add IR learn list node, every new learn list will create it.

参数 **learn\_head** –[in] IR learn list head

返回

- ESP\_OK Create learn list success.
- ESP\_ERR\_NO\_MEM Memory allocation failed.

esp\_err\_t **ir\_learn\_add\_sub\_list\_node** (struct ir\_learn\_sub\_list\_head \*sub\_head, uint32\_t timediff, const rmt\_rx\_done\_event\_data\_t \*symbol)

Add IR learn sub step list node, every sub step should be added.

参数

- **sub\_head** –[in] IR learn sub step list head
- **timediff** –[in] Time diff between each sub step
- **symbol** –[in] symbols of each sub step

返回

- ESP\_OK Create learn list success.
- ESP\_ERR\_NO\_MEM Memory allocation failed.

esp\_err\_t **ir\_learn\_clean\_data** (struct ir\_learn\_list\_head \*learn\_head)

Delete IR learn list node, will recursively delete sub steps.

参数 **learn\_head** –[in] IR learn list head

- ESP\_OK Stop process success.
- ESP\_ERR\_INVALID\_ARG Invalid device handle or argument.

esp\_err\_t **ir\_learn\_clean\_sub\_data** (struct ir\_learn\_sub\_list\_head \*sub\_head)

Delete sub steps.

参数 **sub\_head** –[in] IR learn sub list head

- ESP\_OK Stop process success.
- ESP\_ERR\_INVALID\_ARG Invalid device handle or argument.

esp\_err\_t **ir\_learn\_check\_valid** (struct ir\_learn\_list\_head \*learn\_head, struct ir\_learn\_sub\_list\_head \*result\_out)

Add IR learn list node, every new learn list will create it.

参数

- **learn\_head** –[in] IR learn list head
- **result\_out** –[out] IR learn result

返回

- ESP\_OK Get learn result process.
- ESP\_ERR\_INVALID\_SIZE Size error.

esp\_err\_t **ir\_learn\_print\_raw** (struct ir\_learn\_sub\_list\_head \*cmd\_list)

Print the RMT symbols.

参数 **cmd\_list** –[in] IR learn list head

- ESP\_OK Stop process success.
- ESP\_ERR\_INVALID\_ARG Invalid device handle or argument.

## Structures

struct **ir\_learn\_sub\_list\_t**

An element in the list of infrared (IR) learn data packets.

### Public Functions

**SLIST\_ENTRY (ir\_learn\_sub\_list\_t) next**

Pointer to the next packet

### Public Members

**uint32\_t timediff**

The interval time from the previous packet (ms)

**rmt\_rx\_done\_event\_data\_t symbols**

Received RMT symbols

struct **ir\_learn\_list\_t**

The head of a list of infrared (IR) learn data packets.

### Public Functions

**SLIST\_ENTRY (ir\_learn\_list\_t) next**

Pointer to the next packet

### Public Members

struct ir\_learn\_sub\_list\_head **cmd\_sub\_node**

Package head of every cmd

struct **ir\_learn\_cfg\_t**

IR learn configuration.

### Public Members

**rmt\_clock\_source\_t clk\_src**

RMT clock source

**uint32\_t resolution**

RMT resolution, in Hz

**int learn\_count**

IR learn count needed

**int learn\_gpio**

IR learn io that consumed by the sensor

*ir\_learn\_result\_cb* **callback**

IR learn result callback for user

int **task\_priority**

IR learn task priority

int **task\_stack**

IR learn task stack size

int **task\_affinity**

IR learn task pinned to core (-1 is no affinity)

### Type Definitions

typedef void **\*ir\_learn\_handle\_t**

Type of IR learn handle.

typedef void (**\*ir\_learn\_result\_cb**)(*ir\_learn\_state\_t* state, uint8\_t sub\_step, struct ir\_learn\_sub\_list\_head \*data)

IR learn result user callback.

**Param state [out]** IR learn step

**Param sub\_step [out]** Interval less than 500 ms, we think it's the same command

**Param data [out]** Command list of this step

### Enumerations

enum **ir\_learn\_state\_t**

Type of IR learn step.

*Values:*

enumerator **IR\_LEARN\_STATE\_STEP**

IR learn step, start from 1

enumerator **IR\_LEARN\_STATE\_READY**

IR learn ready, after successful initialization

enumerator **IR\_LEARN\_STATE\_END**

IR learn successfully

enumerator **IR\_LEARN\_STATE\_FAIL**

IR learn failure

enumerator **IR\_LEARN\_STATE\_EXIT**

IR learn exit

# Chapter 11

## 传感器集

### 11.1 Sensor Hub 简介

Sensor Hub 是一个传感器管理组件，可以实现对传感器设备的硬件抽象、设备管理和数据分发。基于 Sensor Hub 开发应用程序时，用户无需处理复杂的传感器实现，只需要对传感器的工作方式、采集间隔、量程等进行简单的选择，然后向关心的事件消息注册回调函数，即可在传感器状态切换或者数据采集好时收到通知。

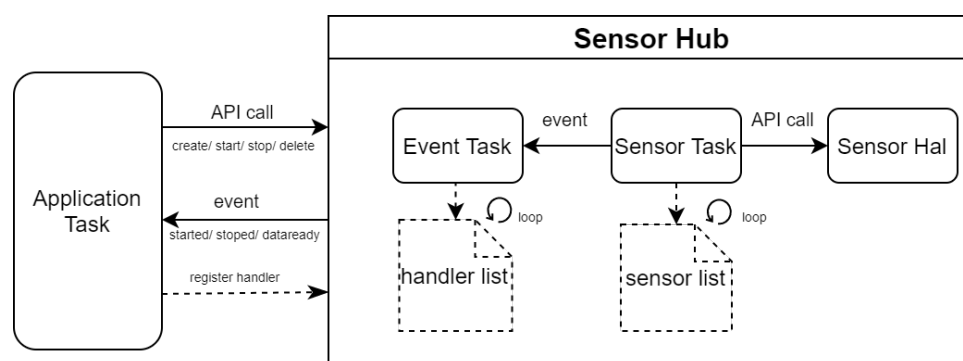


图 1: Sensor Hub 编程模型

Sensor Hub 对常见的传感器类别进行了硬件抽象，用户在切换传感器型号时，无需修改上层应用程序，也可以通过实现硬件抽象层的传感器接口添加新的传感器到 Sensor Hub 中。该组件由于实现了对传感器的集中管理，在简化操作的同时也提高了运行效率，可作为传感器应用的基础组件，应用在环境感知、运动感知、健康管理等更多智能化场景中。

#### 11.1.1 Sensor Hub 使用方法

1. 创建一个传感器实例：使用 `iot_sensor_create()` 创建一个传感器实例，参数包括 `sensor_id_t` 中定义的传感器 ID、传感器配置项和传感器句柄指针。传感器 ID 用于查找和加载对应的驱动，一个 ID 只能对应创建一个实例。配置项中 `bus` 用于指定传感器挂载到的总线位置；`mode` 用于指定传感器的工作模式；`min_delay` 用于指定传感器的采集间隔，其它均为非必须项。创建成功之后，获得该传感器句柄；
2. 注册传感器事件回调函数：在传感器事件发生时，回调函数将会被依次调用，注册回调函数的方法有以下两种，注册成功之后将返回事件回调函数实例句柄：
  - 使用 `iot_sensor_handler_register()` 通过传感器句柄注册回调函数



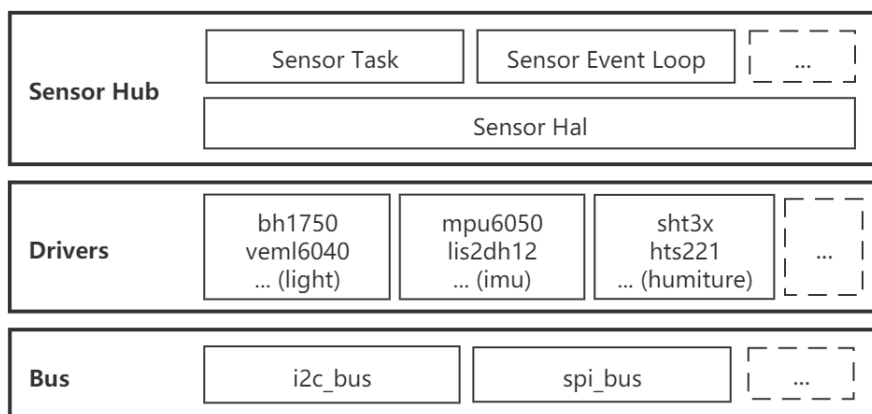


图 2: Sensor Hub 驱动

- 使用 `iot_sensor_handler_register_with_type()` 通过传感器类型注册回调函数
- 3. 启动传感器: 使用 `iot_sensor_start()` 启动指定的传感器, 传感器启动之后将发出 `SENSOR_STARTED` 事件, 之后将以设定的周期持续采集传感器数据, 并发送 `SENSOR_XXXX_DATA_READY` 事件。事件回调函数可通过 `event_data` 参数获取每一个事件的具体数据;
- 4. 停止传感器: 使用 `iot_sensor_stop()` 可临时关闭指定的传感器, 传感器关闭之后将发出 `SENSOR_STOPED` 事件, 之后采集工作将停止。如果该传感器驱动支持电源管理, 传感器将被设置为睡眠模式;
- 5. 取消注册传感器事件回调函数: 用户程序可在任意时刻使用事件回调函数实例句柄取消对事件的注册, 之后该事件发生时, 该回调函数将不再被调用。取消注册的方法对应也有两种:
  - 使用 `iot_sensor_handler_unregister()` 通过传感器句柄取消已注册的回调函数
  - 使用 `iot_sensor_handler_unregister_with_type()` 通过传感器类型取消已经注册的回调函数
- 6. 删除传感器: 使用 `iot_sensor_delete()` 可删除对应的传感器, 释放已分配的内存等资源。

### 11.1.2 示例程序

1. 光照传感器控制 LED 开关示例: [sensors/sensor\\_control\\_led](#)。
2. 传感器监测示例: [sensors/sensor\\_hub\\_monitor](#)。

### 11.1.3 API 参考

#### Header File

- [components/sensors/sensor\\_hub/include/sensor\\_type.h](#)

#### Structures

struct **sensor\_data\_t**  
sensor data type

#### Public Members

int64\_t **timestamp**

timestamp

uint8\_t **sensor\_id**

sensor id

int32\_t **event\_id**

reserved for future use

uint32\_t **min\_delay**

minimum delay between two events, unit: ms

axis3\_t **acce**

Accelerometer. unit: G

axis3\_t **gyro**

Gyroscope. unit: dps

axis3\_t **mag**

Magnetometer. unit: Gauss

float **temperature**

Temperature. unit: dCelsius

float **humidity**

Relative humidity. unit: percentage

float **baro**

Pressure. unit: pascal (Pa)

float **light**

Light. unit: lux

rgbw\_t **rgbw**

Color. unit: lux

uv\_t **uv**

ultraviole unit: lux

float **proximity**

Distance. unit: centimeters

float **hr**

Heat rate. unit: HZ

float **tvoc**

TVOC. unit: permillage

float **noise**

Noise Loudness, unit: HZ

float **step**

Step sensor, unit: 1

float **force**

Force sensor, unit: mN

float **current**

Current sensor unit: mA

float **voltage**

Voltage sensor unit: mV

float **data**[4]

for general use

struct **sensor\_data\_group\_t**

sensor data group type

### Public Members

uint8\_t **number**

effective data number

[\*sensor\\_data\\_t\*](#) **sensor\_data**[SENSOR\_DATA\_GROUP\_MAX\_NUM]

data buffer

### Macros

**SENSOR\_EVENT\_ANY\_ID**

register handler for any event id

### Type Definitions

typedef void \***sensor\_driver\_handle\_t**

hal level sensor driver handle

typedef void \***bus\_handle\_t**

i2c/spi bus handle

### Enumerations

enum **sensor\_type\_t**

sensor type

*Values:*

enumerator **NULL\_ID**

NULL

enumerator **HUMITURE\_ID**

humidity or temperature sensor

enumerator **IMU\_ID**

gyro or acc sensor

enumerator **LIGHT\_SENSOR\_ID**

light illumination or uv or color sensor

enumerator **SENSOR\_TYPE\_MAX**

max sensor type

enum **sensor\_command\_t**

sensor operate command

*Values:*

enumerator **COMMAND\_SET\_MODE**

set measure mode

enumerator **COMMAND\_SET\_RANGE**

set measure range

enumerator **COMMAND\_SET\_ODR**

set output rate

enumerator **COMMAND\_SET\_POWER**

set power mode

enumerator **COMMAND\_SELF\_TEST**

sensor self test

enumerator **COMMAND\_MAX**

max sensor command

enum **sensor\_power\_mode\_t**

sensor power mode

*Values:*

enumerator **POWER\_MODE\_WAKEUP**

wakeup from sleep

enumerator **POWER\_MODE\_SLEEP**

set to sleep

enumerator **POWER\_MAX**

max sensor power mode

enum **sensor\_mode\_t**

sensor acquire mode

*Values:*

enumerator **MODE\_DEFAULT**

default work mode

enumerator **MODE\_POLLING**

polling acquire with a interval time

enumerator **MODE\_INTERRUPT**

interrupt mode, acquire data when interrupt comes

enumerator **MODE\_MAX**

max sensor mode

enum **sensor\_range\_t**

sensor acquire range

*Values:*

enumerator **RANGE\_DEFAULT**

default range

enumerator **RANGE\_MIN**

minimum range for high-speed or high-precision

enumerator **RANGE\_MEDIUM**

medium range for general use

enumerator **RANGE\_MAX**

maximum range for full scale

enum **sensor\_event\_id\_t**

sensor general events

*Values:*

enumerator **SENSOR\_STARTED**

sensor started, data acquire will be started

enumerator **SENSOR\_STOPPED**

sensor stopped, data acquire will be stopped

enumerator **SENSOR\_EVENT\_COMMON\_END**  
max common events id

enum **sensor\_data\_event\_id\_t**  
sensor data ready events

*Values:*

enumerator **SENSOR\_ACCE\_DATA\_READY**  
Accelerometer data ready

enumerator **SENSOR\_GYRO\_DATA\_READY**  
Gyroscope data ready

enumerator **SENSOR\_MAG\_DATA\_READY**  
Magnetometer data ready

enumerator **SENSOR\_TEMP\_DATA\_READY**  
Temperature data ready

enumerator **SENSOR\_HUMI\_DATA\_READY**  
Relative humidity data ready

enumerator **SENSOR\_BARO\_DATA\_READY**  
Pressure data ready

enumerator **SENSOR\_LIGHT\_DATA\_READY**  
Light data ready

enumerator **SENSOR\_RGBW\_DATA\_READY**  
Color data ready

enumerator **SENSOR\_UV\_DATA\_READY**  
ultraviolet data ready

enumerator **SENSOR\_PROXI\_DATA\_READY**  
Distance data ready

enumerator **SENSOR\_HR\_DATA\_READY**  
Heart rate data ready

enumerator **SENSOR\_TVOC\_DATA\_READY**  
TVOC data ready

enumerator **SENSOR\_NOISE\_DATA\_READY**  
Noise Loudness data ready

enumerator **SENSOR\_STEP\_DATA\_READY**  
Step data ready

enumerator **SENSOR\_FORCE\_DATA\_READY**

Force data ready

enumerator **SENSOR\_CURRENT\_DATA\_READY**

Current data ready

enumerator **SENSOR\_VOLTAGE\_DATA\_READY**

Voltage data ready

enumerator **SENSOR\_EVENT\_ID\_END**

max common events id

## Header File

- [components/sensors/sensor\\_hub/include/iot\\_sensor\\_hub.h](#)

## Functions

`esp_err_t iot_sensor_create` (*sensor\_id\_t* sensor\_id, const *sensor\_config\_t* \*config, *sensor\_handle\_t* \*p\_sensor\_handle)

Create a sensor instance with specified sensor\_id and desired configurations.

### 参数

- **sensor\_id** –sensor' s id detailed in *sensor\_id\_t*.
- **config** –sensor' s configurations detailed in *sensor\_config\_t*
- **p\_sensor\_handle** –return sensor handle if succeed, NULL if failed.

### 返回

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t iot_sensor_start` (*sensor\_handle\_t* sensor\_handle)

start sensor acquisition, post data ready events when data acquired. if start succeed, sensor will start to acquire data with desired mode and post events in min\_delay(ms) intervals `SENSOR_STARTED` event will be posted.

**参数** **sensor\_handle** –sensor handle for operation

### 返回

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t iot_sensor_stop` (*sensor\_handle\_t* sensor\_handle)

stop sensor acquisition, and stop post data events. if stop succeed, `SENSOR_STOPED` event will be posted.

**参数** **sensor\_handle** –sensor handle for operation

### 返回

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t iot_sensor_delete` (*sensor\_handle\_t* \*p\_sensor\_handle)

delete and release the sensor resource.

**参数** **p\_sensor\_handle** –point to sensor handle, will set to NULL if delete succeed.

### 返回

- `ESP_OK` Success
- `ESP_FAIL` Fail

```
uint8_t iot_sensor_scan (bus_handle_t bus, sensor_info_t *buf[], uint8_t num)
```

Scan for valid sensors attached on bus.

**参数**

- **bus** –bus handle
- **buf** –Pointer to a buffer to save sensors' information, if NULL no information will be saved.
- **num** –Maximum number of sensor information to save, invalid if buf set to NULL, latter sensors will be discarded if num less-than the total number found on the bus.

**返回** uint8\_t total number of valid sensors found on the bus

```
esp_err_t iot_sensor_handler_register (sensor_handle_t sensor_handle, sensor_event_handler_t  
handler, sensor_event_handler_instance_t *context)
```

Register a event handler to a sensor' s event with sensor\_handle.

**参数**

- **sensor\_handle** –sensor handle for operation
- **handler** –the handler function which gets called when the sensor' s any event is dispatched
- **context** –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\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail

```
esp_err_t iot_sensor_handler_unregister (sensor_handle_t sensor_handle,  
sensor_event_handler_instance_t context)
```

Unregister a event handler from a sensor' s event.

**参数**

- **sensor\_handle** –sensor handle for operation
- **context** –the instance object of the registration to be unregistered

**返回** esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail

```
esp_err_t iot_sensor_handler_register_with_type (sensor_type_t sensor_type, int32_t event_id,  
sensor_event_handler_t handler,  
sensor_event_handler_instance_t *context)
```

Register a event handler with sensor\_type instead of sensor\_handle. the api only care about the event type, don' t care who post it.

**参数**

- **sensor\_type** –sensor type declared in sensor\_type\_t.
- **event\_id** –sensor event declared in sensor\_event\_id\_t and sensor\_data\_event\_id\_t
- **handler** –the handler function which gets called when the event is dispatched
- **context** –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\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail



esp\_err\_t **iot\_sensor\_handler\_unregister\_with\_type** ([sensor\\_type\\_t](#) sensor\_type, int32\_t event\_id, [sensor\\_event\\_handler\\_instance\\_t](#) context)

Unregister a event handler from a event. the api only care about the event type, don' t care who post it.

#### 参数

- **sensor\_type** –sensor type declared in sensor\_type\_t.
- **event\_id** –sensor event declared in sensor\_event\_id\_t and sensor\_data\_event\_id\_t
- **context** –the instance object of the registration to be unregistered

#### 返回

- esp\_err\_t
- ESP\_OK Success
  - ESP\_FAIL Fail

## Structures

struct **sensor\_info\_t**

sensor information type

### Public Members

const char \***name**

sensor name

const char \***desc**

sensor descriptive message

[sensor\\_id\\_t](#) **sensor\_id**

sensor id

const uint8\_t \***addrs**

sensor address list

struct **sensor\_config\_t**

sensor initialization parameter

### Public Members

[bus\\_handle\\_t](#) **bus**

i2c/spi bus handle

[sensor\\_mode\\_t](#) **mode**

set acquire mode detiled in sensor\_mode\_t

[sensor\\_range\\_t](#) **range**

set measuring range

uint32\_t **min\_delay**

set minimum acquisition interval

```
int intr_pin
    set interrupt pin

int intr_type
    set interrupt type
```

### Type Definitions

```
typedef void *sensor_handle_t
    sensor handle

typedef void *sensor_event_handler_instance_t
    sensor event handler handle

typedef void (*sensor_event_handler_t)(void *event_handler_arg, sensor_event_base_t event_base,
int32_t event_id, void *event_data)
    function called when an event is posted to the queue
```

### Enumerations

```
enum sensor_id_t
    sensor id, used for iot_sensor_create

    Values:
```

## 11.2 温湿度传感器

温湿度传感器包括温度传感器、湿度传感器和兼具两种功能的传感器，可用于智慧人居、智慧农场、智能工厂等场景下的环境温湿度检测。

### 11.2.1 已适配列表

名称	功能	总线	供应商	规格书	硬件抽象层
HDC2010	温度、湿度	I2C	TI	<a href="#">规格书</a>	
HTS221	温度、湿度	I2C	ST	<a href="#">规格书</a>	√
SHT3X	温度、湿度	I2C	Sensirion	<a href="#">规格书</a>	√
MVH3004D	温度、湿度	I2C	—		

### 11.2.2 API 参考

以下 API 实现了对温湿度传感器的硬件抽象，用户可直接调用该层代码编写传感器应用程序，也可以使用 [sensor\\_hub](#) 中的传感器接口实现更简单的调用。

#### Header File

- [components/sensors/sensor\\_hub/include/hal/humiture\\_hal.h](#)

## Functions

*sensor\_humiture\_handle\_t* **humiture\_create** (*bus\_handle\_t* bus, int id)

Create a humiture/temperature/humidity sensor instance. Same series' sensor or sensor with same address can only be created once.

**参数**

- **bus** –i2c bus handle the sensor attached to
- **id** –id declared in humiture\_id\_t

**返回** *sensor\_humiture\_handle\_t* return humiture sensor handle if succeed, return NULL if create failed.

esp\_err\_t **humiture\_delete** (*sensor\_humiture\_handle\_t* \*sensor)

Delete and release the sensor resource.

**参数** **sensor** –point to humiture sensor handle, will set to NULL if delete succeed.

**返回** esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **humiture\_test** (*sensor\_humiture\_handle\_t* sensor)

Test if sensor is active.

**参数** **sensor** –humiture sensor handle to operate

**返回** esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **humiture\_acquire\_humidity** (*sensor\_humiture\_handle\_t* sensor, float \*humidity)

Acquire humiture sensor relative humidity result one time.

**参数**

- **sensor** –humiture sensor handle to operate.
- **humidity** –result data (unit:percentage)

**返回** esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail
- ESP\_ERR\_NOT\_SUPPORTED Function not supported on this sensor

esp\_err\_t **humiture\_acquire\_temperature** (*sensor\_humiture\_handle\_t* sensor, float \*sensor\_data)

Acquire humiture sensor temperature result one time.

**参数**

- **sensor** –humiture sensor handle to operate.
- **sensor\_data** –result data (unit:dCelsius)

**返回** esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail
- ESP\_ERR\_NOT\_SUPPORTED Function not supported on this sensor

esp\_err\_t **humiture\_sleep** (*sensor\_humiture\_handle\_t* sensor)

Set sensor to sleep mode.

**参数** **sensor** –humiture sensor handle to operate

**返回** esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail
- ESP\_ERR\_NOT\_SUPPORTED Function not supported on this sensor

esp\_err\_t **humiture\_wakeup** (*sensor\_humiture\_handle\_t* sensor)

Wakeup sensor from sleep mode.

**参数** **sensor** –humiture sensor handle to operate

返回 `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail
- `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t humiture_acquire` (*`sensor_humiture_handle_t`* sensor, *`sensor_data_group_t`* \*data\_group)

acquire a group of sensor data

参数

- **sensor** –humiture sensor handle to operate
- **data\_group** –acquired data

返回 `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t humiture_control` (*`sensor_humiture_handle_t`* sensor, *`sensor_command_t`* cmd, void \*args)

control sensor mode with control commands and args

参数

- **sensor** –humiture sensor handle to operate
- **cmd** –control commands detailed in `sensor_command_t`
- **args** –control commands args
  - `ESP_OK` Success
  - `ESP_FAIL` Fail
  - `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

## Type Definitions

typedef void \***sensor\_humiture\_handle\_t**

humiture sensor handle

## Enumerations

enum **humiture\_id\_t**

humiture sensor id, used for `humiture_create`

*Values:*

enumerator **SHT3X\_ID**

sht3x humiture sensor id

enumerator **HTS221\_ID**

hts221 humiture sensor id

enumerator **HUMITURE\_MAX\_ID**

max humiture sensor id

## 11.3 惯性传感器 (IMU)

惯性传感器包含陀螺仪传感器、加速度传感器和兼具多种功能的传感器等。主要用于测量物体的加速度、角速度等，进而解算出物体的运动姿态。

### 11.3.1 已适配列表

名称	功能	总线	供应商	规格书	硬件抽象层
LIS2DH12	3-axis acceler	I2C	ST	<a href="#">规格书</a>	√
MPU6050	3-axis acceler + 3-axis gyro	I2C	InvenSense	<a href="#">规格书</a>	√

### 11.3.2 API 参考

以下 API 实现了对惯性传感器的硬件抽象，用户可直接调用该层代码编写传感器应用程序，也可以使用 [sensor\\_hub](#) 中的传感器接口实现更简单的调用。

#### Header File

- [components/sensors/sensor\\_hub/include/hal/imu\\_hal.h](#)

#### Functions

[sensor\\_imu\\_handle\\_t](#) **imu\_create** ([bus\\_handle\\_t](#) bus, int imu\_id)

Create a Inertial Measurement Unit sensor instance. Same series' sensor or sensor with same address can only be created once.

##### 参数

- **bus** -i2c bus handle the sensor attached to
- **imu\_id** -id declared in imu\_id\_t

返回 [sensor\\_imu\\_handle\\_t](#) return imu sensor handle if succeed, NULL is failed.

esp\_err\_t **imu\_delete** ([sensor\\_imu\\_handle\\_t](#) \*sensor)

Delete and release the sensor resource.

##### 参数

**sensor** -point to imu sensor handle, will set to NULL if delete succeed.

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **imu\_test** ([sensor\\_imu\\_handle\\_t](#) sensor)

Test if sensor is active.

##### 参数

**sensor** -imu sensor handle to operate

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **imu\_acquire\_acce** ([sensor\\_imu\\_handle\\_t](#) sensor, axis3\_t \*acce)

Acquire imu sensor accelerometer result one time.

##### 参数

- **sensor** -imu sensor handle to operate
- **acce** -result data (unit:g)

返回 esp\_err\_t

- ESP\_OK Success
- ESP\_FAIL Fail
- ESP\_ERR\_NOT\_SUPPORTED Function not supported on this sensor

esp\_err\_t **imu\_acquire\_gyro** ([sensor\\_imu\\_handle\\_t](#) sensor, axis3\_t \*gyro)

Acquire imu sensor gyroscope result one time.

##### 参数

- **sensor** -imu sensor handle to operate

- **gyro** –result data (unit:dps)
- 返回 `esp_err_t`
- `ESP_OK` Success
  - `ESP_FAIL` Fail
  - `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t imu_sleep` (*`sensor_imu_handle_t`* sensor)

Set sensor to sleep mode.

参数 **sensor** –imu sensor handle to operate

- 返回 `esp_err_t`
- `ESP_OK` Success
  - `ESP_FAIL` Fail
  - `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t imu_wakeup` (*`sensor_imu_handle_t`* sensor)

Wakeup sensor from sleep mode.

参数 **sensor** –imu sensor handle to operate

- 返回 `esp_err_t`
- `ESP_OK` Success
  - `ESP_FAIL` Fail
  - `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t imu_acquire` (*`sensor_imu_handle_t`* sensor, *`sensor_data_group_t`* \*data\_group)

acquire a group of sensor data

参数

- **sensor** –imu sensor handle to operate
- **data\_group** –acquired data

- 返回 `esp_err_t`
- `ESP_OK` Success
  - `ESP_FAIL` Fail

`esp_err_t imu_control` (*`sensor_imu_handle_t`* sensor, *`sensor_command_t`* cmd, void \*args)

control sensor mode with control commands and args

参数

- **sensor** –imu sensor handle to operate
- **cmd** –control commands detailed in `sensor_command_t`
- **args** –control commands args
  - `ESP_OK` Success
  - `ESP_FAIL` Fail
  - `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

## Type Definitions

typedef void \***sensor\_imu\_handle\_t**  
imu sensor handle

## Enumerations

enum **imu\_id\_t**  
imu sensor id, used for `imu_create`  
*Values:*

```

enumerator MPU6050_ID
    MPU6050 imu sensor id

enumerator LIS2DH12_ID
    LIS2DH12 imu sensor id

enumerator IMU_MAX_ID
    max imu sensor id

```

## 11.4 环境光传感器

环境光传感器包含光照强度传感器、颜色传感器、紫外线传感器和兼具多种功能的传感器。

### 11.4.1 已适配列表

名称	功能	总线	供应商	规格书	硬件抽象层
BH1750	Light	I2C	rohm	<a href="#">规格书</a>	√
VEML6040	Light RGBW	I2C	Vishay	<a href="#">规格书</a>	√
VEML6075	Light UVA UVB	I2C	Vishay	<a href="#">规格书</a>	√

### 11.4.2 API 参考

以下 API 实现了对环境光传感器的硬件抽象，用户可直接调用该层代码编写传感器应用程序，也可以使用 [sensor\\_hub](#) 中的传感器接口，实现更简单的调用。

#### Header File

- [components/sensors/sensor\\_hub/include/hal/light\\_sensor\\_hal.h](#)

#### Functions

[sensor\\_light\\_handle\\_t](#) **light\_sensor\_create** ([bus\\_handle\\_t](#) bus, int id)

Create a light sensor instance. same series' sensor or sensor with same address can only be created once.

#### 参数

- **bus** -i2c bus handle the sensor attached to
- **id** -id declared in [light\\_sensor\\_id\\_t](#)

**返回** [sensor\\_light\\_handle\\_t](#) return light sensor handle if succeed, return NULL if failed.

[esp\\_err\\_t](#) **light\_sensor\_delete** ([sensor\\_light\\_handle\\_t](#) \*sensor)

Delete and release the sensor resource.

**参数** **sensor** -point to light sensor handle, will set to NULL if delete succeed.

**返回** [esp\\_err\\_t](#)

- ESP\_OK Success
- ESP\_FAIL Fail

`esp_err_t light_sensor_test (sensor_light_handle_t sensor)`

Test if sensor is active.

**参数** **sensor** –light sensor handle to operate.

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t light_sensor_acquire_light (sensor_light_handle_t sensor, float *lux)`

Acquire light sensor illuminance result one time.

**参数**

- **sensor** –light sensor handle to operate.
- **lux** –result data (unit:lux)

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail
- `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t light_sensor_acquire_rgbw (sensor_light_handle_t sensor, rgbw_t *rgbw)`

Acquire light sensor color result one time. light color includes red green blue and white.

**参数**

- **sensor** –light sensor handle to operate.
- **rgbw** –result data (unit:lux)

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail
- `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t light_sensor_acquire_uv (sensor_light_handle_t sensor, uv_t *uv)`

Acquire light sensor ultra violet result one time. light Ultraviolet includes UVA UVB and UV.

**参数**

- **sensor** –light sensor handle to operate.
- **uv** –result data (unit:lux)

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail
- `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t light_sensor_sleep (sensor_light_handle_t sensor)`

Set sensor to sleep mode.

**参数** **sensor** –light sensor handle to operate.

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail
- `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t light_sensor_wakeup (sensor_light_handle_t sensor)`

Wakeup sensor from sleep mode.

**参数** **sensor** –light sensor handle to operate.

**返回** `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail
- `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

`esp_err_t light_sensor_acquire (sensor_light_handle_t sensor, sensor_data_group_t *data_group)`

acquire a group of sensor data

**参数**



- **sensor** –light sensor handle to operate
- **data\_group** –acquired data

返回 `esp_err_t`

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t light_sensor_control` (*`sensor_light_handle_t`* sensor, *`sensor_command_t`* cmd, void \*args)

control sensor mode with control commands and args

参数

- **sensor** –light sensor handle to operate
- **cmd** –control commands detailed in `sensor_command_t`
- **args** –control commands args
  - `ESP_OK` Success
  - `ESP_FAIL` Fail
  - `ESP_ERR_NOT_SUPPORTED` Function not supported on this sensor

## Type Definitions

typedef void \***sensor\_light\_handle\_t**

light sensor handle

## Enumerations

enum **light\_sensor\_id\_t**

light sensor id, used for `light_sensor_create`

*Values:*

enumerator **BH1750\_ID**

BH1750 light sensor id

enumerator **VEML6040\_ID**

VEML6040 light sensor id

enumerator **VEML6075\_ID**

VEML6075 light sensor id

enumerator **LIGHT\_MAX\_ID**

max light sensor id

## 11.5 气压传感器

气压传感器用于检测气体绝对压强，也可以用于计算海拔高度等，常用于环境监测设备，高度测量和空间定位设备等。

### 11.5.1 已适配列表

名称	功能	总线	供应商	规格书	硬件抽象层
BME280	Pressure	I2C/SPI	BOSCH	<a href="#">规格书</a>	

## 11.6 手势传感器

手势传感器一般是指将反射红外光的测量结果转换为有关物理运动的传感器，可用于实现非接触式人机交互等。

### 11.6.1 已适配列表

名称	功能	总线	供应商	规格书	硬件抽象层
APDS9960	Light, RGB and Gesture Sensor	I2C	Avago	<a href="#">规格书</a>	

## 11.7 热敏电阻传感器

NTC 是 Negative Temperature Coefficient 的缩写，NTC 热敏电阻是一种具有负温度系数的热敏电阻，其特点是在工作温度范围内电阻随温度的升高而降低。利用其基本的电阻温度特性、电压电流特性，NTC 系列热敏电阻已广泛应用于家用电器产品中，以达到自动增益调整、过负荷保护、温度控制、温度补偿、稳压稳幅等作用。

NTC 常见的封装/形态如下：

1. 探针式 NTC 热敏电阻：适合多种应用场景，适合极高温或极低温探测
2. 贴片式 NTC 热敏电阻：一次性贴装在 PCB 上，适合温度补偿电路，LED 照明温度监测，电池组温度监测

NTC 关键参数：

1. 阻值规格 (25°C 阻值 1K,5K,10K,50K,100K 等)
2. 阻值精度 (0.5%,1%,2%,3%,5%)
3. 使用温度范围
4. B 值 (材料常数，B 值越高电阻变化率越高)
5. 探头外形、探头材料、导线材料、导线长度

**NTC Driver 适用电路：**单 ADC 通道测量：当 NTC 在上面时，随着温度升高，NTC 的电阻值下降，导致分压电路中 NTC 端的电压下降，固定电阻端的电压上升，最终输出电压会随之变化。而当 NTC 在下面时，情况正好相反，NTC 的电阻值下降会导致分压电路中 NTC 端的电压上升，固定电阻端的电压下降，输出电压也会相应变化。因此，NTC 在上面和下面会得到相反的结果，这一点在设计和使用 NTC 测量电路时需要特别注意，确保正确理解和处理 NTC 的温度特性。

当使用以下电路时：Vcc —→ Rt —→ Rref —→ GND 需要通过 `ntc\_config\_t` 中的 `circuit\_mode` 字段选择相应的电路模式。当使用这种电路模式时，应该使用 `CIRCUIT\_MODE\_NTC\_VCC`。

当使用以下电路时：Vcc —→ Rref —→ Rt —→ GND 需要通过 `ntc\_config\_t` 中的 `circuit\_mode` 字段选择相应的电路模式。当使用这种电路模式时，应该使用 `CIRCUIT\_MODE\_NTC\_GND`。

关于分压电阻 Rref 取值：Rref 的阻值一般取 Rt 在 25 摄氏度的阻值

**NTC 数字温度转换参数，公式为：** $R_t = R \cdot \exp(B \cdot (1/T_1 - 1/T_2))$  -Rt 是热敏电阻在 T1 温度下的电阻 -R 是室温下热敏电阻在 T2 时的标称电阻 -B 值是热敏电阻的一个重要参数 -EXP 是 e 的 n 次方 -T1 和 T2 是指 K 度，即开尔文温度，K 度 = 273.15 (绝对温度) + 摄氏度

## 11.7.1 应用示例

### 创建 ntc 热敏电阻驱动

```
// Create a ntc driver and register call-back
ntc_config_t ntc_config = {
    .b_value = 3950,
    .r25_ohm = 10000,
    .fixed_ohm = 10000,
    .vdd_mv = 3300,
    .circuit_mode = CIRCUIT_MODE_NTC_GND,
    .atten = ADC_ATTEN_DB_11,
    .channel = ADC_CHANNEL_3,
    .unit = ADC_UNIT_1
};

ntc_device_handle_t ntc = NULL;
adc_oneshot_unit_handle_t adc_handle = NULL;
ESP_ERROR_CHECK(ntc_dev_create(&ntc_config, &ntc, &adc_handle));
ESP_ERROR_CHECK(ntc_dev_get_adc_handle(ntc, &adc_handle));

//get ntc temperature
float temp = 0.0;
if (ntc_dev_get_temperature(ntc, &temp) == ESP_OK) {
    ESP_LOGI(TAG, "NTC temperature = %.2f °C", temp);
}

//delete handle
TEST_ASSERT_EQUAL(ESP_OK, ntc_dev_delete(ntc));
```

## 11.7.2 API 参考

以下 API 实现了对热敏电阻传感器的硬件抽象，用户可直接调用该层代码编写传感器应用程序，也可以使用 [sensor\\_hub](#) 中的传感器接口实现更简单的调用。

### Header File

- [components/sensors/ntc\\_driver/include/ntc\\_driver.h](#)

### Functions

`esp_err_t ntc_dev_create(ntc\_config\_t *config, ntc\_device\_handle\_t *ntc_handle, adc\_oneshot\_unit\_handle\_t *adc_handle)`

Initialize the NTC and ADC channel config.

#### 参数

- **config** –
- **ntc\_handle** –A ntc driver handle
- **adc\_handle** –A adc handle

#### 返回

- ESP\_OK Success
- ESP\_FAIL Failure

`esp_err_t ntc_dev_get_adc_handle(ntc\_device\_handle\_t ntc_handle, adc\_oneshot\_unit\_handle\_t *adc_handle)`

Get the adc handle.

#### 参数

- **ntc\_handle** –A ntc driver handle
- **adc\_handle** –A adc handle

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

esp\_err\_t **ntc\_dev\_delete** (*ntc\_device\_handle\_t* ntc\_handle)

Delete ntc driver device and ntc detect device.

**参数** **ntc\_handle** –A ntc driver handle

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

esp\_err\_t **ntc\_dev\_get\_temperature** (*ntc\_device\_handle\_t* ntc\_handle, float \*temperature)

Get the ntc temperature.

**参数**

- **ntc\_handle** –A ntc driver handle
- **temperature** –NTC temperature

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

## Structures

struct **ntc\_config\_t**

NTC config data type.

## Public Members

*ntc\_circuit\_mode\_t* **circuit\_mode**

ntc circuit mode

adc\_unit\_t **unit**

adc unit

adc\_atten\_t **atten**

adc atten

adc\_channel\_t **channel**

adc channel

uint32\_t **b\_value**

beta value of NTC (K)

uint32\_t **r25\_ohm**

25°C resistor value of NTC (K)

uint32\_t **fixed\_ohm**

fixed resistor value ( $\Omega$ )

```
uint32_t vdd_mv  
    vdd voltage (mv)
```

### Type Definitions

```
typedef void *ntc_device_handle_t
```

### Enumerations

```
enum ntc_circuit_mode_t  
    Supported circuit mode.  
  
    Values:  
  
    enumerator CIRCUIT_MODE_NTC_VCC  
  
    enumerator CIRCUIT_MODE_NTC_GND
```

## 11.8 功率监视器

功率监视器是一种用于监测和管理电源的集成电路。它可以实时监测电源的电压、电流和功率等参数以供系统使用。Power Monitor ICs 可以应用在包括计算机、电源管理系统、消费电子设备、工业控制系统和通信设备等。

### 11.8.1 适配列表

名称	功能	总线	供应商	规格书	硬件抽象层
INA236	16 位数字功率监视器	I2C	TI	<a href="#">规格书</a>	x

### 11.8.2 API 参考

The following API implements hardware abstraction for power monitor sensors. Users can directly call this layer of code to write sensor applications.

#### Header File

- [components/sensors/power\\_monitor/ina236/include/ina236.h](#)

#### Functions

```
esp_err_t ina236_create(ina236\_handle\_t *handle, ina236\_config\_t *config)  
    Create and init object and return a handle.
```

##### 参数

- **handle** –Pointer to handle
- **config** –Pointer to configuration

**返回**

- ESP\_OK Success
- Others Fail

esp\_err\_t **ina236\_delete** ([\*ina236\\_handle\\_t\*](#) handle)

Deinit object and free memory.

**参数** **handle** –ina236 handle Handle

**返回**

- ESP\_OK Success
- Others Fail

esp\_err\_t **ina236\_get\_voltage** ([\*ina236\\_handle\\_t\*](#) handle, float \*volt)

Get the Voltage on the bus.

**参数**

- **handle** –object handle of ina236
- **volt** –Voltage value in volts

**返回**

- ESP\_OK Success
- Others Fail

esp\_err\_t **ina236\_get\_current** ([\*ina236\\_handle\\_t\*](#) handle, float \*curr)

Get the Current on the bus.

**参数**

- **handle** –object handle of ina236
- **curr** –Current value in A

**返回**

- ESP\_OK Success
- Others Fail

esp\_err\_t **ina236\_clear\_mask** ([\*ina236\\_handle\\_t\*](#) handle)

Clear the mask of the alert pin.

**参数** **handle** –object handle of ina236

**返回**

- ESP\_OK Success
- Others Fail

## Structures

struct **ina236\_config\_t**

ina236 configuration structure

### Public Members

[\*i2c\\_bus\\_handle\\_t\*](#) **bus**

I2C bus object

bool **alert\_en**

Enable alert callback

uint8\_t **dev\_addr**

I2C device address

`uint8_t alert_pin`

Alert pin number

`int236_alert_cb_t alert_cb`

Alert callback function

## Macros

`INA236_I2C_ADDRESS_DEFAULT`

## Type Definitions

`typedef struct ina236_t *ina236_handle_t`

INA236 handle.

`typedef void (*int236_alert_cb_t)(void *arg)`

ina236 alert callback function

## Chapter 12

# 触摸传感器

### 12.1 触摸接近感应传感器

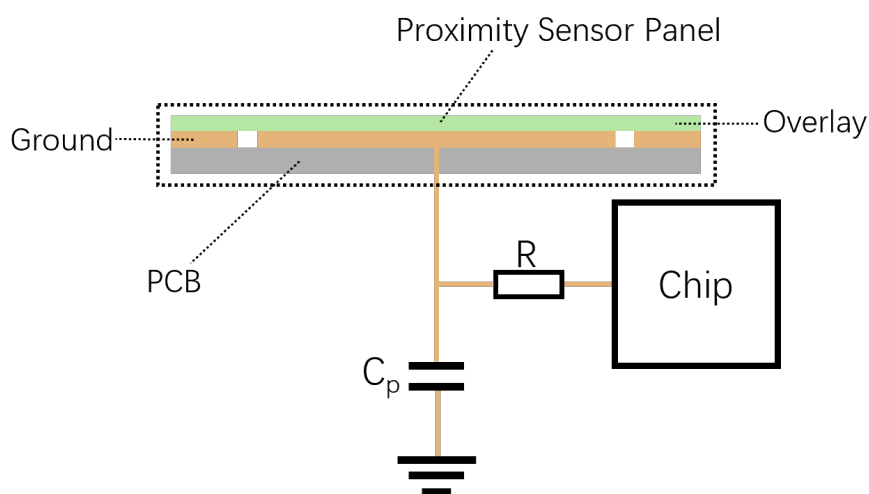
`touch_proximity_sensor` 组件基于 ESP32-S3 内置的触摸传感器进行开发，使用该组件可以轻松实现触摸接近感应功能。

#### 备注:

- ESP32/ESP32-S2/ESP32-S3 触摸相关组件仅用于测试或演示目的。由于触摸功能的抗干扰能力差，可能无法通过 EMS 测试，因此不建议用于量产产品。
- 该组件目前仅适用于 ESP32-S3，并且需要 IDF 版本大于等于 v5.0。

#### 12.1.1 实现原理

触摸接近感应传感器是以 ESP32-S3 的触摸传感器接近感应功能为基础实现的，其硬件原理结构图如下：



当目标物体靠近传感器时，其等效电容会发生变化。目标物体可以是人的手指、手或任何导电物体。当触摸传感器被配置为接近感应模式时，传感器输出的值是累加值，当目标物体靠近传感器面板时，传感器输出的累计值会变大。基于这一特点，本方案将触摸传感器输出的原始数据（累加值）定义为 `raw_value`



，并从中衍生出 *baseline* 和 *smooth\_value* 两个数据变量，再结合合理的阈值检测算法，最终实现接近感应功能。

具体的软件实现有以下三个步骤：

1. 判断新数据的有效性。
2. 依据 *smooth\_value* 和 *baseline* 的更新逻辑，以 *raw\_value* 为源数据更新 *smooth\_value* 和 *baseline*。
3. 判断 *smooth\_value* - *baseline* 的值是否大于 0，若大于 0 则再判断是否大于 **触发阈值**，若大于则判定为有效的感应触发动作；若 *smooth\_value* - *baseline* 的值小于 0，先判断当前是否处于触发状态，若处于触发状态，则再判断其绝对值是否大于 **解除触发阈值**，若大于则判定为有效的触发解除动作。

### 12.1.2 测试硬件参考

- 开发板
  - 可以使用的 [ESP-S2S3-Touch-DevKit-1](#) 开发套件进行验证测试，主板为 MainBoard v1.1，接近感应子板为 Proximity Board V1.0。

### 12.1.3 配置参考

#### 创建接近感应传感器

使用 `touch_proximity_sensor` 组件，可通过 `proxi_config_t` 结构体来配置接近感应传感器。

```
// Configuration structure for touch proximity sensor
typedef struct {
    uint32_t channel_num;
    uint32_t channel_list[TOUCH_PROXIMITY_NUM_MAX];
    uint32_t meas_count;
    float smooth_coef;
    float baseline_coef;
    float max_p;
    float min_n;
    float threshold_p[TOUCH_PROXIMITY_NUM_MAX];
    float threshold_n[TOUCH_PROXIMITY_NUM_MAX];
    float hysteresis_p;
    float noise_p;
    float noise_n;
    uint32_t debounce_p;
    uint32_t debounce_n;
    uint32_t reset_p;
    uint32_t reset_n;
    uint32_t gold_value[TOUCH_PROXIMITY_NUM_MAX];
} proxi_config_t;
```

具体参数说明如下：

配置参数	说明
channel_num	触摸接近感应通道数量，最多支持 3 个
channel_list	触摸接近感应通道列表，即触摸通道
meas_count	接近感应通道的累计测量次数，值越大，数据更新越慢
smooth_coef	数据平滑处理系数，降低数据波动
baseline_coef	基线系数，确定基线调整的速率，用于消除环境变化的影响
max_p	最大有效正变化率
min_n	最小有效负变化率
threshold_p	正向触发阈值
threshold_n	负向触发阈值
hysteresis_p	正阈值迟滞系数，在触发和解除触发之间提供缓冲区，以防止连续误触发
noise_p	正噪声阈值，基线更新与该值有关
noise_n	负噪声阈值，基线更新与该值有关
debounce_p	正阈值的去抖动次数，以减少误触发
debounce_n	负阈值的去抖动次数，以减少误解除触发
reset_p	触发基线重置的正向阈值
reset_n	触发基线重置的负向阈值
gold_value	金标准值，用于在特殊情况下恢复正常值

然后使用 `touch_proximity_sensor_create()` 配置并创建接近感应传感器对象。

```
proxi_config_t config = (proxi_config_t)DEFAULTS_PROX_CONFIGS();
esp_err_t ret = touch_proximity_sensor_create(&config, &s_touch_proximity_sensor, &
↪example_proxi_callback, NULL);
if (ret != ESP_OK) {
    ESP_LOGE(TAG, "touch proximity sense create failed");
}
```

其中，`s_touch_proximity_sensor` 为触摸接近感应传感器句柄，`example_proxi_callback` 为接近感应传感器事件回调函数。

### 启动和停止接近感应传感器

使用 `touch_proximity_sensor_start()` 启动接近感应传感器：

```
// Start the touch proximity sensor
touch_proximity_sensor_start(s_touch_proximity_sensor);
```

使用 `touch_proximity_sensor_stop()` 停止接近感应传感器：

```
// Stop the touch proximity sensor
touch_proximity_sensor_stop(s_touch_proximity_sensor);
```

**备注：**接近感应传感器的启动和停止过程需要一定时间才能完成，因此，在调用启动和停止 API 之后，添加等待时间是有必要的，通常，启动时间为 300 ms，停止过程需 200 ms，详情请参考示例程序。

### 删除接近感应传感器

使用 `touch_proximity_sensor_delete()` 删除接近感应传感器对象，并释放资源：

```
// Delete the touch proximity sensor
touch_proximity_sensor_delete(s_touch_proximity_sensor);
```

### 12.1.4 参数调节参考

- `channel_num` 最大为 3。
- `channel_list` 数组必须赋值为 `touch_pad_t` 枚举变量中的值。
- `meas_count` 数值越大，触摸传感器新数据的更新速率越慢。
- `smooth_coef` 是数据平滑处理系数，平滑后的 `smooth` 值等于  $smooth * (1.0 - smooth\_coef) + raw * smooth\_coef$ ，`smooth_coef` 数值越大，`raw` 的权重就越大，平滑效果越差，`smooth` 波形越抖，`smooth` 跟随 `raw` 值速度越快，触发响应越快，抗干扰能力越弱；`smooth_coef` 数值越小，`raw` 的权重就越小，平滑效果越好，`smooth` 波形越平滑，`smooth` 跟随 `raw` 值速度越慢，触发响应越慢，抗干扰能力越强。
- `baseline_coef` 是基线更新系数，基线新值等于  $baseline * (1.0 - baseline\_coef) + smooth * baseline\_coef$ ，该值越大，基线跟随 `smooth` 速度越快，触发响应越慢，抗干扰能力越强。
- `max_p` 当 `raw - baseline` 的值大于  $baseline * max\_p$  时，`raw` 值为异常值，忽略掉。
- `min_n` 当 `baseline - raw` 的值大于  $baseline * min\_n$  时，`raw` 值为异常值，忽略掉。
- `threshold_p` 值越大，接近感应触发的距离越近，抗干扰能力越强，反之相反。
- `threshold_n` 值越大，接近感应触发的距离越近，抗干扰能力越强，反之相反。
- `noise_p` 和 `noise_n` 的值越大，基线更容易跟随 `smooth`，接近感应距离会相应变小，抗干扰能力越好。
- `debounce_p` 和 `debounce_n` 的值需要参考 `meas_count` 的值进行调整，`meas_count` 越小，`debounce_p` 和 `debounce_n` 应相应增大，以提高抗干扰能力。

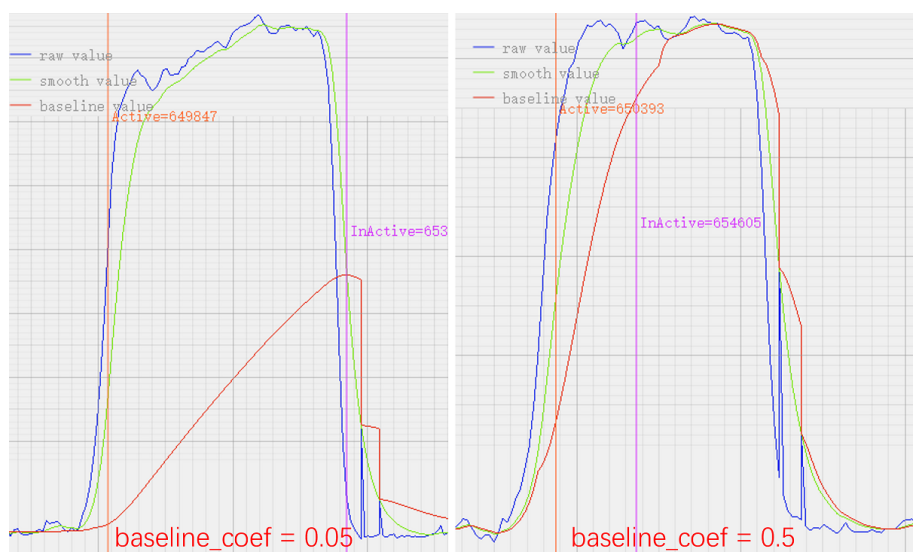
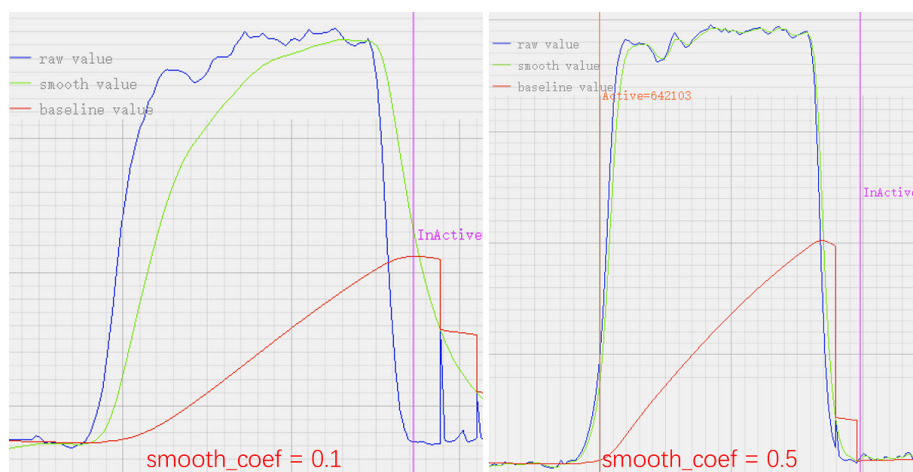
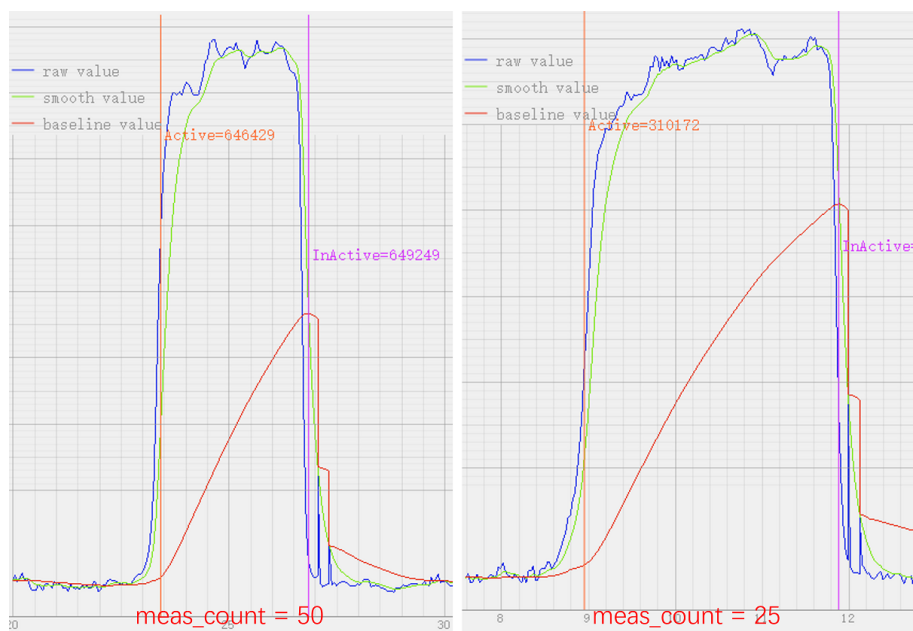
#### 调参波形对比

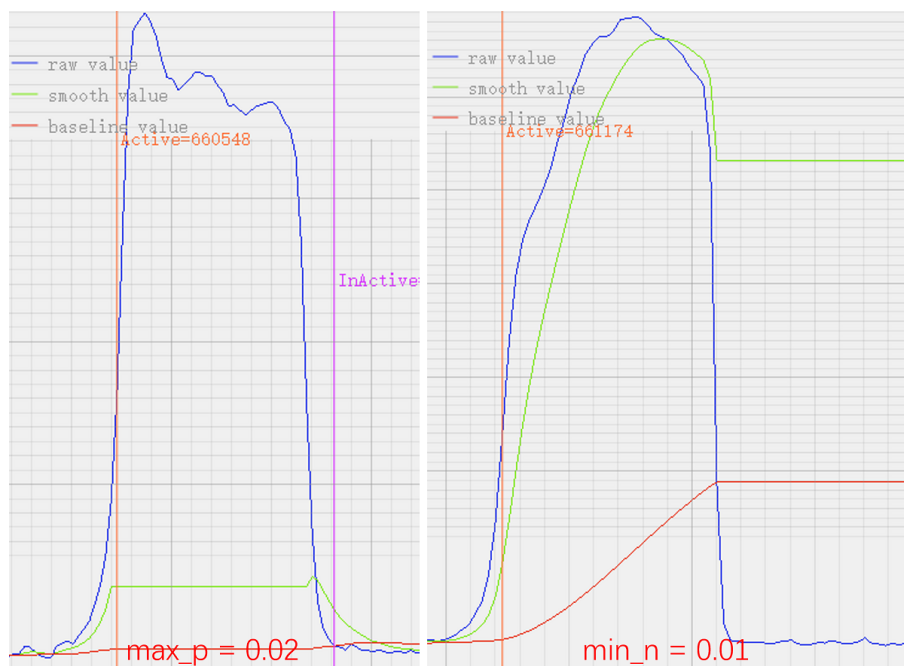
默认的触摸接近感应传感器配置参数如下：

参数	默认值
<code>channel_num</code>	1
<code>channel_list</code>	TOUCH_PAD_NUM8
<code>meas_count</code>	50
<code>smooth_coef</code>	0.2
<code>baseline_coef</code>	0.1
<code>max_p</code>	0.2
<code>min_n</code>	0.08
<code>threshold_p</code>	0.002
<code>threshold_n</code>	0.002
<code>hysteresis_p</code>	0.2
<code>noise_p</code>	0.001
<code>noise_n</code>	0.001
<code>debounce_p</code>	2
<code>debounce_n</code>	1
<code>reset_p</code>	1000
<code>reset_n</code>	3

以下调参对比都将在以上参数基础上 **仅修改一个参数** 进行对比。

1. 修改 `meas_count` 的值，将改变传感器数据更新速率，其值越大，传感器数据更新速率越慢。测试现象：将手放在感应面板上方 10cm 处保持 3 秒时间，较小的 `meas_count` 数值，感应时的波形宽度将更宽，波形对比图如下：
2. 修改 `smooth_coef` 的值，将改变 `smooth` 波形的平滑效果。`smooth_coef` 值越小，平滑效果越好，抗干扰能力越强，`smooth` 跟随 `raw` 越慢，触发响应越慢，反之相反。不同 `smooth_coef` 下的波形对比图如下：
3. 修改 `baseline_coef` 的值，将改变 `baseline` 的更新效果。`baseline_coef` 值越小，`baseline` 跟随 `smooth` 越慢，触发响应越慢，触发的持续时间越长，反之相反。不同 `baseline_coef` 下的波形对比图如下：
4. 修改 `max_p` 和 `min_n` 的值，将改变 `smooth` 和 `baseline` 的更新逻辑。`max_p` 值太小，会导致手接近感应面板时 `smooth` 和 `baseline` 太小时的波形图如下：





5. 修改 `threshold_p` 的值，将改变接近感应的距离，其值越小，能够感应的距离越远，但抗干扰能力越差，易引发误触发。不同 `threshold_p` 下的波形对比图如下：
6. 修改 `hysteresis_p` 的值，将改变触发和解除触发的时间点，即触发迟滞和解除触发迟滞。`hysteresis_p` 的值越小，触发响应越快，反之相反。不同 `threshold_p` 下的波形对比图如下：
7. 修改 `noise_p` 和 `noise_n` 的值，将改变 `baseline` 的更新效果。`noise_p` 的值越小，`baseline` 跟随 `smooth` 越慢，触发响应越慢，触发的持续时间越长，反之相反。不同 `noise_p` 和 `noise_n` 下的波形对比图如下：
8. 修改 `debounce_p` 和 `debounce_n` 的值，将改变触发和解除触发的时间点和抗干扰能力。`debounce_p` 的值越大，触发响应越慢，抗干扰能力越强，反之相反；`debounce_n` 的值越大，解除触发响应越慢，抗干扰能力越强，反之相反。`debounce_p` 和 `debounce_n` 的值需要结合 `meas_count` 来调节，`meas_count` 的值减小，`debounce_p` 和 `debounce_n` 的值应适当增大。不同 `noise_p` 和 `noise_n` 下的波形对比图如下：

**备注：**要达到理想的接近感应效果，仅对一两个参数进行简单调节是不够的，需要综合调整多个参数。

### 12.1.5 示例程序

- [touch/touch\\_proximity](#)

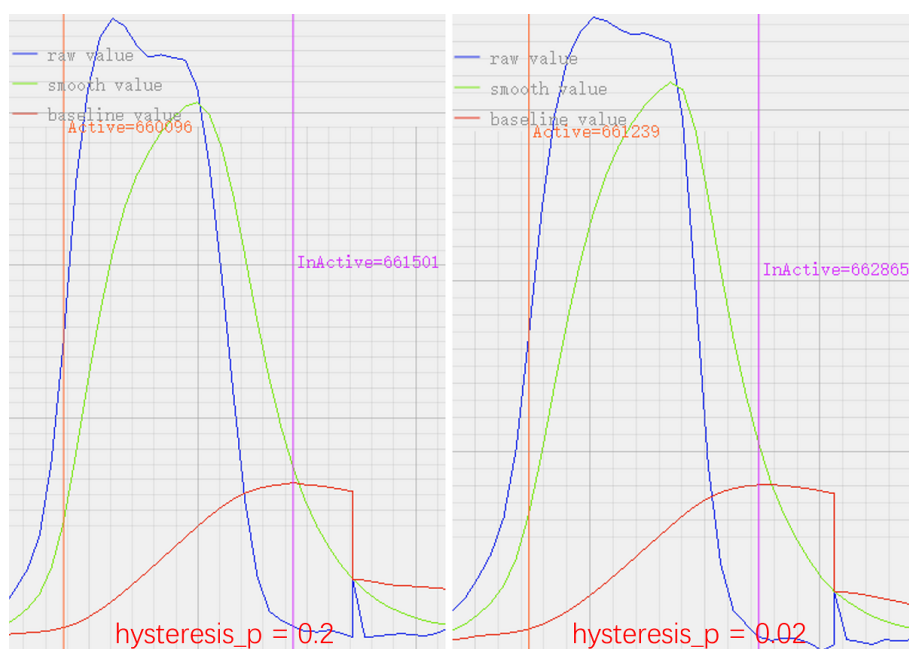
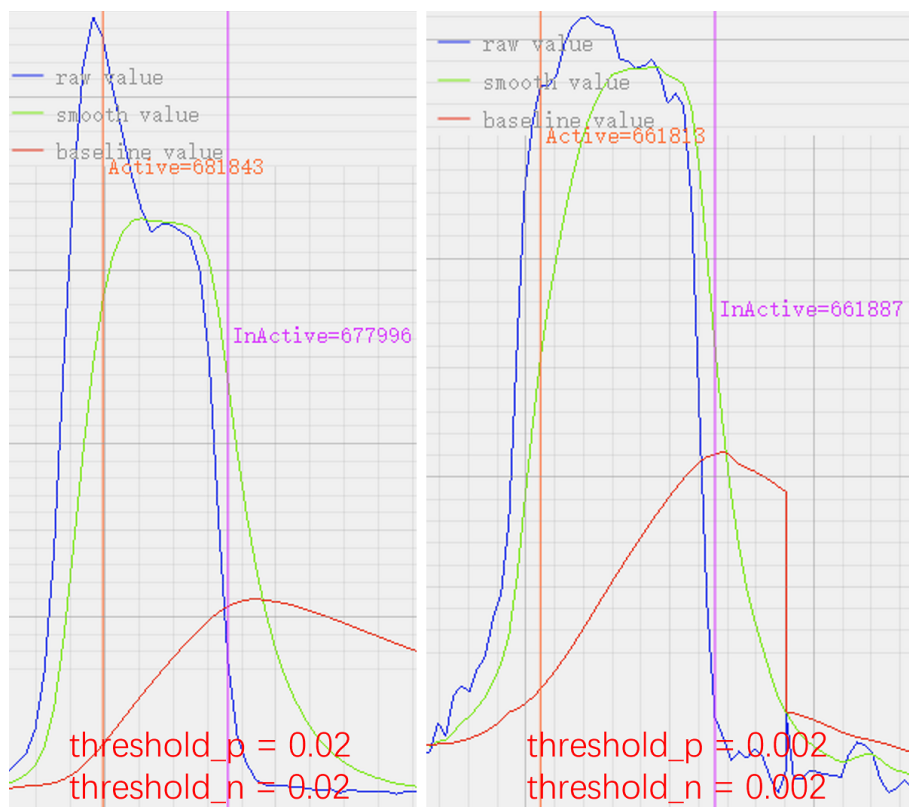
### 12.1.6 API Reference

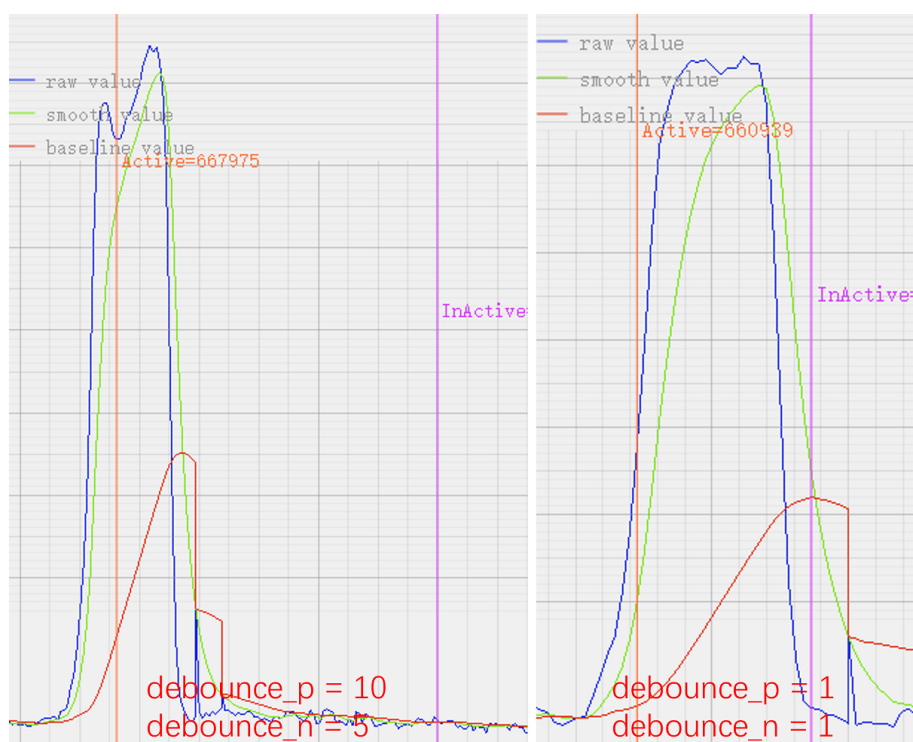
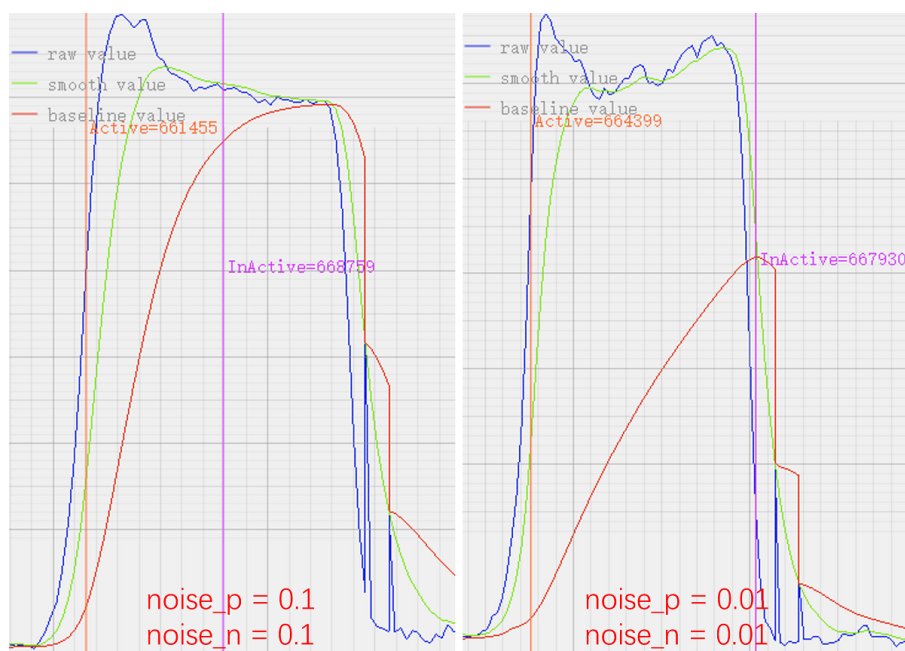
#### Header File

- [components/touch/touch\\_proximity\\_sensor/include/touch\\_proximity\\_sensor.h](#)

#### Functions

`esp_err_t touch_proximity_sensor_create` (`proxi_config_t` \*config, `touch_proximity_handle_t` \*sensor\_handle, `proxi_cb_t` cb, void \*cb\_arg)







Create a touch proximity sensor instance.

**参数**

- **config** –The touch pad channel configuration.
- **sensor\_handle** –The handle of the successfully created touch proximity sensor.
- **cb** –Callback function to handle proximity events.
- **cb\_arg** –The callback function argument.

**返回**

- ESP\_OK: Create the touch proximity sensor successfully.
- ESP\_ERR\_NO\_MEM: Failed to create the touch proximity sensor (memory allocation failed).

esp\_err\_t **touch\_proximity\_sensor\_start** (*touch\_proximity\_handle\_t* proxi\_sensor)

Start the touch proximity sensor.

This function starts the touch proximity sensor operation.

**参数** **proxi\_sensor** –Pointer to the handle of the touch proximity sensor instance.

**返回**

- ESP\_OK: Start the touch proximity sensor successfully
- ESP\_ERR\_INVALID\_ARG: The touch proximity sensor failed to start (*touch\_proximity\_handle\_t* is NULL, or *channel\_num* is zero).
- ESP\_FAIL: The touch proximity sensor failed to start (failed to create queue for touch pad).

esp\_err\_t **touch\_proximity\_sensor\_stop** (*touch\_proximity\_handle\_t* proxi\_sensor)

Stop the touch proximity sensor.

This function stops the operation of the touch proximity sensor associated with the provided sensor handle.

**参数** **proxi\_sensor** –Pointer to the handle of the touch proximity sensor instance.

**返回**

- ESP\_OK: Stop the touch proximity sensor successfully

esp\_err\_t **touch\_proximity\_sensor\_delete** (*touch\_proximity\_handle\_t* proxi\_sensor)

Delete the touch proximity sensor instance.

This function deletes the touch proximity sensor instance associated with the provided sensor handle.

**参数** **proxi\_sensor** –Pointer to the handle of the touch proximity sensor instance to be deleted.

**返回**

- ESP\_OK: Delete the touch proximity sensor instance successfully

## Structures

struct **proxi\_config\_t**

Configuration structure for touch proximity sensor.

This structure defines the configuration parameters for a touch proximity sensor.

## Public Members

uint32\_t **channel\_num**

Number of touch proximity sensor channels

uint32\_t **channel\_list**[TOUCH\_PROXIMITY\_NUM\_MAX]

Touch proximity sensor channel list



`uint32_t meas_count`  
Accumulated measurement count

`float smooth_coef`  
Smoothing coefficient

`float baseline_coef`  
Baseline coefficient

`float max_p`  
Maximum effective positive change rate

`float min_n`  
Minimum effective negative change rate

`float threshold_p[TOUCH_PROXIMITY_NUM_MAX]`  
Positive threshold

`float threshold_n[TOUCH_PROXIMITY_NUM_MAX]`  
Negative threshold

`float hysteresis_p`  
Hysteresis for positive threshold

`float noise_p`  
Positive noise threshold

`float noise_n`  
Negative noise threshold

`uint32_t debounce_p`  
Debounce times for positive threshold

`uint32_t debounce_n`  
Debounce times for negative threshold

`uint32_t reset_p`  
Baseline reset positive threshold

`uint32_t reset_n`  
Baseline reset negative threshold

`uint32_t gold_value[TOUCH_PROXIMITY_NUM_MAX]`  
Gold value

## Macros

`TOUCH_PROXIMITY_NUM_MAX`

**DEFAULTS\_PROX\_CONFIGS** ()

### Type Definitions

typedef struct touch\_proximity\_sensor\_t \***touch\_proximity\_handle\_t**

typedef void (\***proxi\_cb\_t**)(uint32\_t channel, *proxi\_evt\_t* event, void \*cb\_arg)  
proximity sensor user callback type

### Enumerations

enum **proxi\_evt\_t**

*Values:*

enumerator **PROXI\_EVT\_INACTIVE**

enumerator **PROXI\_EVT\_ACTIVE**



# Chapter 13

# 存储方案

## 13.1 存储媒介

已支持存储媒介列表：

名称	关键特性	应用场景	容量	传输模式	速度	驱动	备注
SPI Flash	可与代码共用，无附加成本	参数存储、文本、图像存储	MB	SPI	40/80 MHz 4 线	SPI Flash Driver	
SD Card	大容量、可插拔	声音、视频文件存储	GB	SDIO/SPI	20/40 MHz 1 线/4 线	SD/SDIO/MMC Driver	*1
eMMC	大容量、高速读写	声音、视频文件存储	GB	SDIO	20/40 MHz 1 线/4 线/8 线	SD/SDIO/MMC Driver	*2
EEP-ROM	可按字节寻址，低成本	参数存储	MB	I2C	100 ~ 400 KHz	eeptom	

备注：

- \*1. ESP32-S2 仅支持 SPI 模式
- \*2. ESP32-S2 不支持

### 13.1.1 SPI Flash

ESP32/ESP32-S/ESP32-C 系列芯片默认使用 NOR Flash 存取用户代码和数据，Flash 可集成在模组或芯片中，容量一般为 4 MB、8 MB 或 16 MB。对于 ESP-IDF v4.0 以上版本，SPI Flash 组件除了支持对主 Flash 进行读写操作以外，同时可支持外接第二块 Flash 芯片进行数据的存储。

用户可以使用 [分区表](#) 对 Flash 进行分区，根据分区表的功能划分，Flash 除了存放用户编译生成的二进制代码之外，也可以作为非易失性存储 (NVS) 空间存放应用程序参数，还可以将指定 Flash 空间挂载到文件系统 (FatFS 等) 存放文本、图像等文件。

Flash 芯片支持双线 (DOUT/DIO) 和四线 (QOUT/QIO) 操作模式，可通过配置工作在 40 MHz 或 80 MHz 模式。由于可以直接使用主 Flash 芯片进行数据存储，无需额外添加存储芯片，因此特别适合于容量需求较小 (MB)、集成度要求高、成本敏感的应用场景。

参考文档：

- [SPI Flash API](#)

### 13.1.2 SD Card

ESP32 支持使用 SDIO 接口或 SPI 接口访问 SD 卡。SDIO 接口支持 1 线、4 线或 8 线模式，支持默认速率 20 MHz 和高速 40 MHz 两种速率。需要注意的是该接口至少占用 6 个 GPIO，并且只能使用 [固定的引脚](#)。SPI 接口支持通过 GPIO matrix 为 SD 卡指定任意的 IO，通过 CS 引脚可以支持对多个 SD 卡的访问，SPI 接口硬件设计上更加灵活，但是相比 SDIO 接口访问速率较低。

ESP-IDF 中的 SD/SDIO/MMC Driver 基于 SD 卡的两种访问模式进行了协议层的封装，提供了 SD 卡的初始化接口和协议层 API。SD 卡具有大容量、可插拔的特点，广泛适用于智能音响、电子相册等具有大容量存储需求的应用场景。

**参考文档：**

- [SD/SDIO/MMC 驱动](#)：支持 SDIO 和 SPI 两种传输模式；
- [SDMMC 主机驱动](#)：SDIO 模式；
- [SD SPI 主机驱动](#)：SPI 模式；
- 使用 SPI 或 1-bit 模式，请注意 [引脚上拉需求](#)。

**示例程序：**

- [storage/sd\\_card](#)：访问使用 FAT 文件系统的 SD 卡。

### 13.1.3 eMMC

eMMC (embedded MMC) 内存芯片与 SD 卡具有相似的协议，可以使用与 SD 卡相同的驱动程序 [SD/SDIO/MMC 驱动](#)。但是需要注意的是，eMMC 芯片仅能使用 SDIO 模式，不支持 SPI 模式。eMMC 目前在 8 线模式下支持默认速率 20 MHz 和高速 40 MHz，在 4 线模式下支持高速 40 MHz DDR 模式。

eMMC 一般以芯片的形式焊接到主板上，相比 SD 卡集成度更高，适用于可穿戴设备等，具有大容量存储需求同时对系统集成度有一定要求的场景。

**参考文档：**

- [SD/SDIO/MMC 驱动](#)；
- [已支持的 eMMC 速度模式](#)。

### 13.1.4 EEPROM

EEPROM (如 [AT24C0X 系列](#)) 是 1024-16384 位的串行电可擦写存储器（通过控制引脚电平也可运行在只读模式），它的存储空间一般按照 word 进行分布，每个 word 包含 8-bit 空间。EEPROM 可按字节寻址，读写操作简单，特别适合于保存配置参数等，经过优化也可应用于对功耗和可靠性等有一定要求的工业和商业场景。

**已适配的 EEPROM 芯片：**

名称	功能	总线	供应商	规格书	驱动
AT24C01/02	1024/2048 bits EEPROM	I2C	Atmel	<a href="#">规格书</a>	<a href="#">eeprom</a>

## 13.2 文件系统

已支持的文件系统列表：

表 1: 文件系统特性比较

关键特性	NVS 库	FAT 文件系统	SPIFFS 文件系统	LittleFS 文件系统
特点	键值对保存, 接口安全	操作系统支持, 兼容性强	针对嵌入式开发, 资源占用低	资源占用低, 读、写、擦除速度快
应用场景	参数保存	音视频、文件保存	音视频、文件保存	文件保存
容量	KB-MB	GB	< 128 MB	< 128 MB
目录支持	X	√	X	√
磨损均衡	√	可选	√	√
读写效率	0	0	0	高
资源占用	0	0	1	1
掉电保护	√	X	X	√
加密	√	√	X	X

**备注:**

- 0: 暂无数据或不参与比较。
- 1: 低 RAM 占用。

### 13.2.1 NVS 库

非易失性存储 (NVS) 库主要用于读写在 flash NVS 分区中存储的数据。NVS 中的数据以键值对的方式保存, 其中键是 ASCII 字符串, 值可以是整数、字符串、二进制数据 (BLOB) 类型。NVS 支持掉电保护和数据加密, 适合存储一些较小的数据, 如应用程序参数等。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。

**参考文档:**

- [非易失性存储库](#)。
- 批量生产时, 可以使用 [NVS 分区生成工具](#)。

**示例程序:**

- 写入单个整数值: [storage/nvs\\_rw\\_value](#)。
- 写入二进制大对象: [storage/nvs\\_rw\\_blob](#)。

### 13.2.2 FAT 文件系统

ESP-IDF 使用 FatFs 库实现了对 FAT 文件系统的支持, FatFs 是独立于平台和存储介质的文件系统层, 通过统一接口实现对物理设备 (如 flash、SD 卡) 的访问。用户可以直接调用 FatFs 的接口操作, 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。

FAT 文件系统操作系统兼容性强, 广泛应用于 USB 存储盘或 SD 卡等移动存储设备上。ESP32 系列芯片通过支持 FAT 文件系统, 可以实现对这些常见存储设备的访问。

**参考文档:**

- [FatFs 与 VFS 配合使用](#)。
- [FatFs 与 VFS 和 SD 卡配合使用](#)。

**示例程序:**

- [storage/sd\\_card](#): 访问使用 FAT 文件系统的 SD 卡。
- [storage/ext\\_flash\\_fatfs](#): 访问使用 FAT 文件系统的外部 Flash 芯片。

### 13.2.3 SPIFFS 文件系统

SPIFFS 是一个专用于 SPI NOR flash 的嵌入式文件系统，原生支持磨损均衡、文件系统一致性检查等功能。用户可以直接调用 SPIFFS 提供的 Posix 样式接口，也可以通过 VFS 操作 SPIFFS 的大多数功能。

SPIFFS 作为嵌入式平台 SPI NOR Flash 的专用文件系统，相比 FAT 文件系统占用 RAM 资源更少，但是仅用于支持容量小于 128 MB 的 Flash 芯片。

**参考文档：**

- [SPIFFS 文件系统](#)。
- [两种生成 SPIFFS 镜像的工具](#)。

**示例程序：**

- [storage/spiffs](#)：SPIFFS 使用示例。

### 13.2.4 LittleFS 文件系统

LittleFS 是一个专用于 SPI NOR flash 的嵌入式文件系统，原生支持磨损均衡、文件系统一致性检查、断电保护等功能。

LittleFS 作为高完整性的嵌入式平台 SPI NOR Flash 文件系统，支持高效的读写速度且占用的 RAM 资源更少。

LittleFS 目前由第三方维护，可通过包管理器轻松获取。

**参考文档：**

- [LittleFS 文件系统组件仓库](#)。
- [LittleFS 文件系统组件使用说明](#)。

**示例程序：**

- [storage/littlefs](#)：LittleFS 使用示例。

### 13.2.5 虚拟文件系统 (VFS)

ESP-IDF 虚拟文件系统 (VFS) 组件可以为不同文件系统 (FAT, SPIFFS) 提供统一的接口，也可以为设备驱动程序提供类似文件读写的操作接口。

**参考文档：**

- [虚拟文件系统组件](#)。

## Chapter 14

# 电机

### 14.1 无刷电机

#### 14.1.1 无刷电机控制概述

本指南包含以下内容：

- [无刷电机概述](#)：无刷电机优势
- [驱动方式](#)：无刷电机的驱动硬件方式
- [控制方式](#)：控制无刷电机的几种方式

#### 无刷电机概述

无刷直流（Brushless Direct Current, BLDC）电机属于同步电机的一种，可配置为单相，两相，三相。此文讨论的都是三相无刷电机。

无刷电机不使用电刷进行换向，而是使用电子换向，具有以下优点：

- 更好的转速-转矩特性
- 快速的动态响应
- 高效率
- 使用寿命长
- 运转无噪音
- 较高的转速范围

无刷电机的组成为定子和转子两部分：

- 定子是线圈绕组电枢，具有三个星型连接的定子绕组，沿着定子圆周分布这些绕组，以构成均匀分布的磁极。
- 转子用永磁体制成，永磁体的磁极数目大多为 2 到 8 磁极。南磁极和北磁极交替。

如果只给电机通固定的直流电流，电机只会产生不变的磁场。无法转动起来。只有通过适当的顺序来为定子相位供电，在定子上产生一个旋转磁场。转子的固有磁极跟随定子的旋转磁场有序旋转，才能达到转动的目的。





图 1: BLDC 无刷电机

**备注：**理想状态下，转矩峰值出现在两个磁场正交时候，而在两磁场平行时最弱。

#### 重要参数：

- $KV$  值 ( $rpm/V$ ): 可以直观表示无刷电机在具体工作电压下的具体转速。

$$n(rpm) = KV * U$$

- 转矩 ( $Nm$ ): 电机中转子产生的可以用来带动机械负载的驱动力矩。
- 转速 ( $rpm$ ): 电机每分钟的转速。
- 最大电流 ( $A$ ): 可以承受并安全工作的最大电流。
- 极对数  $Pp$ : 转子上磁钢的数量除以 2, 可以通过给任意两相通过小电压, 手动旋转电机一周, 感受阻力的次数就是极对数。如感到 6 次阻力, 极对数就是 6。
- 相电感  $LS(H)$ : 电机静止时的定子绕组两端的电感为  $LL$ , 相电感为其一半:

$$LS = LL/2$$

- 相电阻  $RS(\Omega)$ : 万用表测电机两项电阻  $RL$ , 相电阻为其一半:

$$RS = RL/2$$

#### 驱动方式

无刷电机一般通过 6 MOS 管组成的逆变电路进行驱动, 通过上臂和下臂开关器件的组合, 可以在定子上产生一个旋转磁场。

通过图上的逆变电路, 按照顺序依次导通, 转子磁铁就能循环转动, 每经过 6 次切换电流, 转子转动一圈。这里展示的是导通两个桥臂的方式。

**备注：**上下桥臂不能同时导通, 否则会短路, 因此需要引入死区控制, 来规避掉同一相的上下桥臂同时导通的情况。

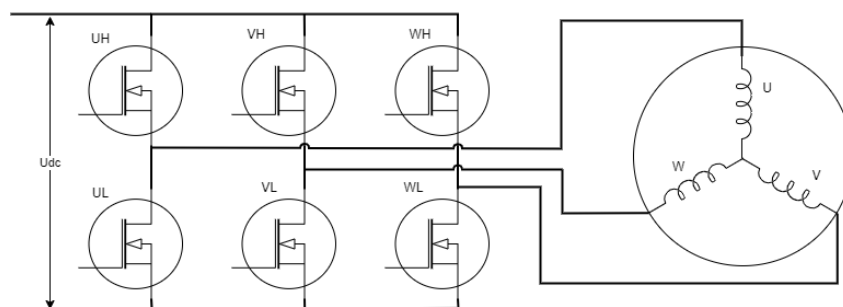


图 2: BLDC 三相逆变电路

表 1: 桥臂导通与电流流向

导通上臂	导通下臂	相电流 A	相电流 B	相电流 C
UH	WL	DC+	悬空	DC-
UH	VL	DC+	DC-	悬空
WH	VL	悬空	DC-	DC+
WH	UL	DC-	悬空	DC+
VH	UL	DC-	DC+	悬空
VH	WL	悬空	DC+	DC-

为了让电机旋转的速度可控，可以将施加在上臂的控制信号设置为 PWM 信号，并通过控制 PWM 的占空比来达到控制转速的作用。

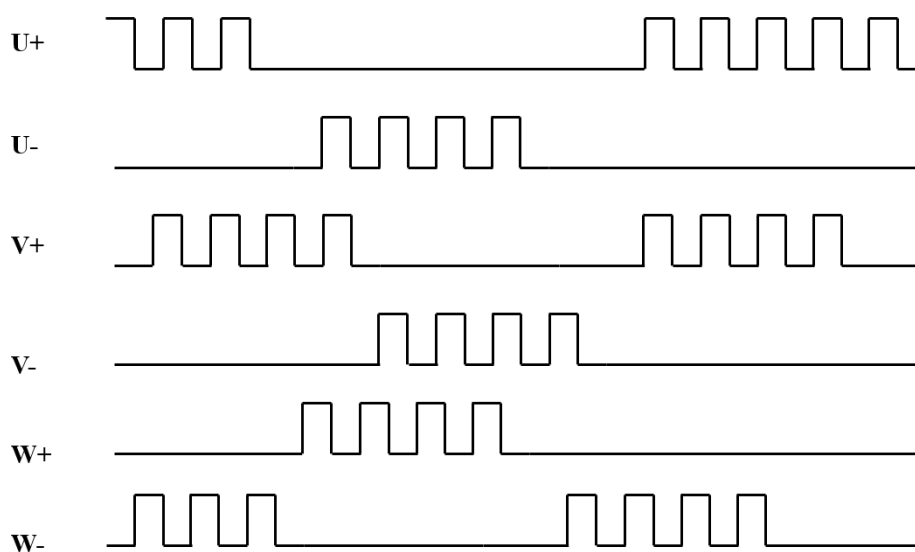


图 3: BLDC PWM 速度控制

### 控制方式

在实际的电机控制中，需要获取转子位置，并计算出下一步导通的桥臂，这样才能让电机旋转起来。获取转子位置一般有两种方式，有感检测和无感检测。

**有感霍尔** 在无刷电机中，一般用 3 个开关型霍尔器件检测转子位置，安装位置一般相隔 120°。如下图所示

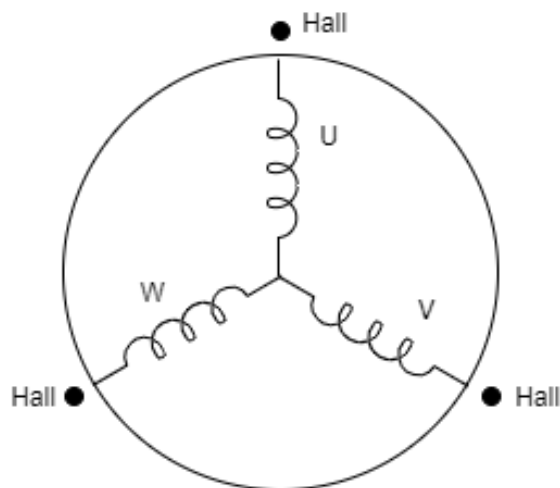


图 4: BLDC 霍尔传感器安装位置

当 N 极靠近霍尔 a 时，a 输出高电平 1，当 N 极远离 a 时，a 输出低电平。其他同理。那么当转子转动一圈，会产生下面的波形。

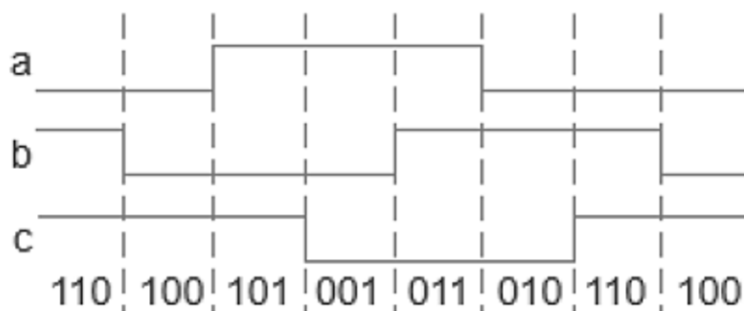


图 5: BLDC 霍尔传感器波形

通过解析霍尔传感器的输出，确定转子的当前位置。并使用“二二导通”法让电机旋转起来，但其存在如下缺点：

- 传感器价格昂贵，并且需要再制造期间将传感器安装在电机上，增加了安装与接线费用。
- 若传感器发生故障，则电机无法继续增长工作。

因此，基于无感检测的无刷电机控制方案成为了主流。

**无感检测** 在一些微小电机系统中，安装位置传感器对电机的体积和成本有不利影响。因此无传感器的位置检测也非常重要。无感控制策略主要包括反电势法、电感法、续流二极管法等。其中，反电动势法是应用最广、最成熟的方案之一。

**反电动势** 反电动势根据楞次定律，方向与提供绕组的主电压相反。反电动势的极性与励磁电压相反。反电动势主要取决于三个方向。

- 转子角速度

- 转子磁铁产生的磁场
- 定子绕组的匝数

$$BEMF = NlrB\omega$$

对于电机来说，转子磁场和定子绕组的匝数都是固定的，那么在实际运转中，唯一决定反电动势的因素就是角速度，或者说转子转速。在每次换向时，都有一个绕组得正电，第二个得负电，第三个保持开路状态。

通过检测各相绕组的反电动势过零点，就能在一个电周期内得到转子的六个位置。非导通相的反电动势过零点延迟  $30^\circ$  电角度就是换相点。

**备注：**在电机转速极慢的时候，反电动势的幅值很低，很难检测到过零点。

基于反电动势检测过零点有两种方式

- 基于 ADC 采样的无感方波电机控制 ADC 采样检测过零点
- 基于比较器检测的无感方波电机控制 比较器检测过零点

此外还有基于相电流采集的无感 FOC 方案

- 双电阻无感 FOC 方案（待更新）

### 14.1.2 基于 ADC 采样的无感方波电机控制

本指南包含以下内容：

#### 目录

- 初始位置检测
- 基于 ADC 方案的 BLDC 无感控制
  - 反电动势定义
  - ADC 方案的过零点采样原理
  - ADC 方案的过零点采样硬件

#### 初始位置检测

由无刷电机的一相电压平衡方程  $u_a = Ri_a + L_a \frac{di_a}{dt} + e_a$  可知，当电机静止时，反电动势为零，则电枢电流为：

$$i = \frac{U}{R} \left( 1 - \frac{1}{e^\tau} \right)$$

通过分析上述公式可知，通过施加高频电压以产生对应的脉冲电流，并对比脉冲电流大小，可准确确定转子的位置范围。

为获取初始状态下的转子位置，`esp_sensorless_bldc_control` 组件在启动时按照顺序施加电压脉冲，获取采样电阻上的电流脉冲，比较 6 个矢量的大小以确定最大矢量所在区间。

表 2: 脉冲注入顺序

注入顺序	U 相	V 相	W 相
1	Udc	Udc	GND
2	GND	GND	Udc
3	Udc	GND	Udc
4	GND	Udc	GND
5	GND	Udc	Udc
6	Udc	GND	GND

**备注:**

1. 在静止状态下，对 BLDC 电机分别注入特定电压矢量，每个电压矢量作用固定时间  $T_s$ ，保障注入电流大小。
2. 电压矢量注入结束时刻，对母线电流采样。
3. 依次注入剩余电压矢量，比较各电压矢量作用下的电流值大小，确定最大电流标识，得到转子初始位置。

**基于 ADC 方案的 BLDC 无感控制**

**反电势定义** 当无刷电机转动时，每个绕组都会产生反电动势电压，根据楞次定律，反电势极性与主电压相反。反电势计算公式：

$$BEMF = NlrB\omega$$

其中，N 为绕组匝数，l 为转子长度，r 为转子内半径，B 为转子磁场， $\omega$  为角速度。

当电机做定后，电机绕组与转子参数固定。电机反电势只与角速度成正比。

下图为电机旋转一个电周期中电流与反电势波形。

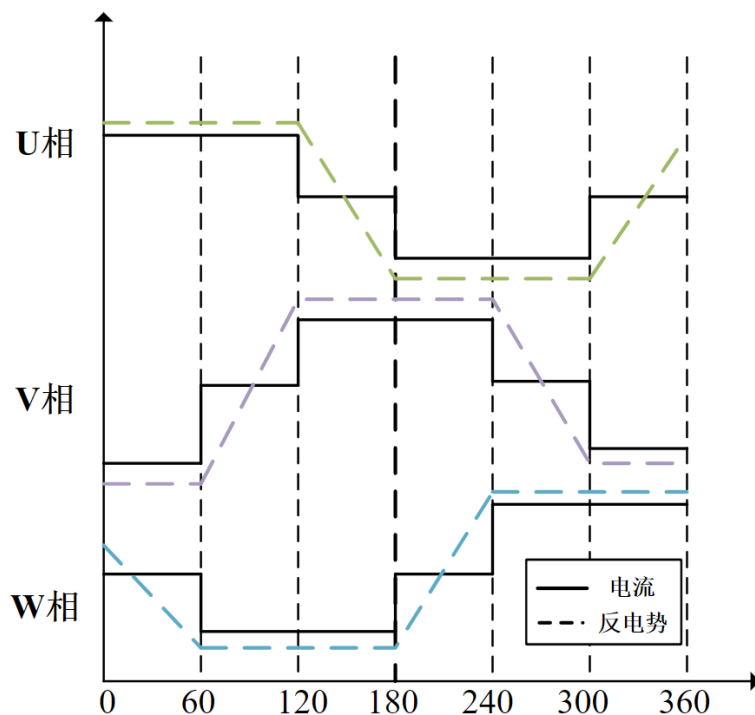


图 6: 电流与反电势波形

**ADC 方案的过零点采样原理** 当 BLDC 电机转动时，反电势过零点发生在浮空相。通过检测各相各相对地电压，并与直流母线电压对比。当端电压等于直流母线电压一半时，即发生过零事件。在基于 ADC 的过零点检测方案中，同时测量端电压与直流母线电压并进行对比，获得过零信号。

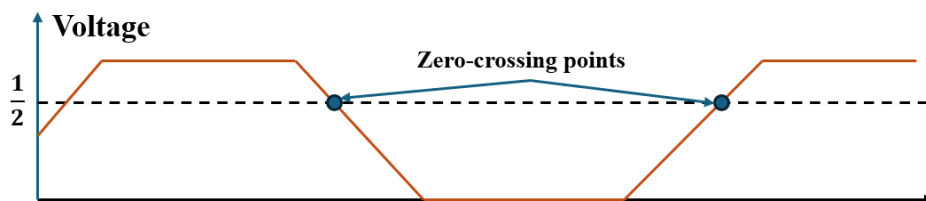


图 7: ADC 过零点检测实现方式

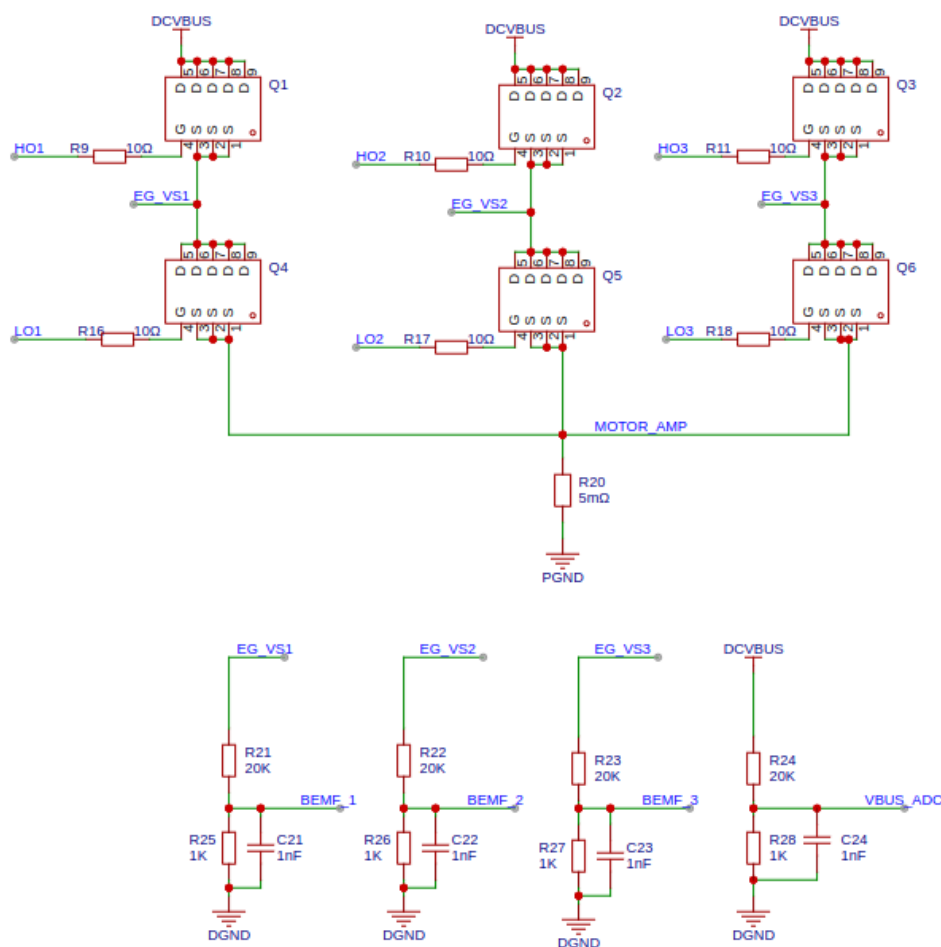


图 8: ADC 过零点检测硬件

**ADC 方案的过零点采样硬件** 为简化计算流程，端电压与直流母线电压采用相同的分压系数。在 12V 电机控制方案中，采用 1/21 的分压方案，控制直流母线电压与端电压范围在 ESP32 系列芯片的  $V_{ref}$  范围内。

**备注：** 注意，电压需要转化到 ESP32 ADC 能够采集的范围。请参考：[ESP32 ADC](#)

### 14.1.3 基于比较器检测的无感方波电机控制

本指南包含以下内容：

#### 目录

- 基于比较器方案的 *BLDC* 无感控制
  - 反电势定义
  - 比较器方案的过零点采样原理
  - 比较器方案的过零点采样硬件

#### 基于比较器方案的 BLDC 无感控制

**反电势定义** 当无刷电机转动时，每个绕组都会产生反电动势电压，根据楞次定律，反电势极性与主电压相反。反电势计算公式：

$$BEMF = NlrB\omega$$

其中， $N$  为绕组匝数， $l$  为转子长度， $r$  为转子内半径， $B$  为转子磁场， $\omega$  为角速度。

当电机做定后，电机绕组与转子参数固定。电机反电势只与角速度成正比。

下图为电机旋转一个电周期中电流与反电势波形。

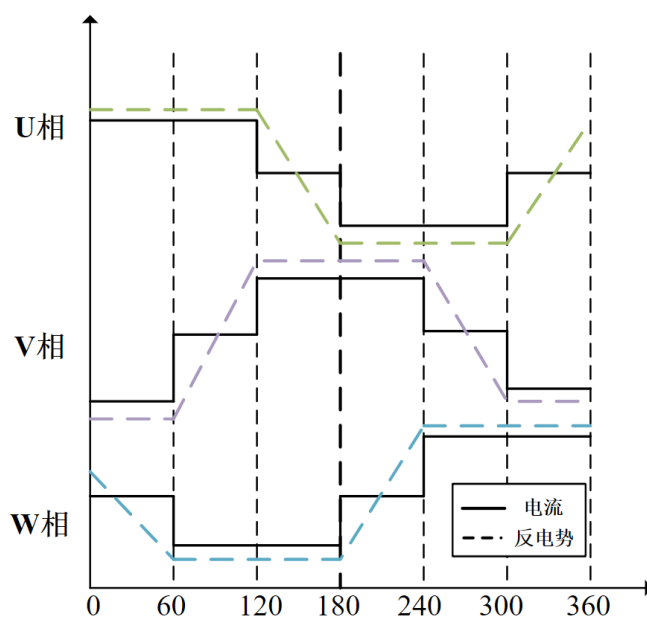


图 9: 电流与反电势波形

**比较器方案的过零点采样原理** 当 BLDC 电机转动时，反电势过零点发生在浮空项。通过检测各相对地电压，并与中性点电压比较。当端电压从大于中性点电压变成小于中性点电压，或端电压从小于中性点电压变成大于中性点电压时，即为过零点。但一般 BLDC 电机并未引出中性点，导致无法直接测量中性点电压。在基于比较器的过零点检测方案中，将三相绕组通过等阻值电阻连接到公共点，以此重构中性点，并将中性点与端电压通过比较器获得过零信号。

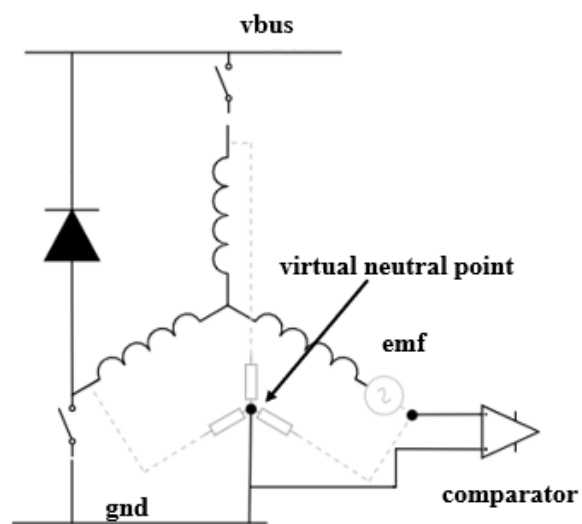


图 10: 比较器过零点原理

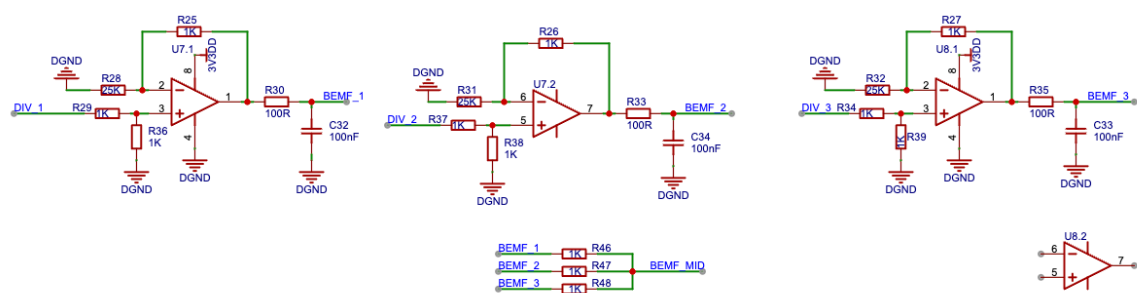


图 11: 比较器过零点硬件



**比较器方案的过零点采样硬件** 使用相同阻值的电阻连接各相构建虚拟中性点。以 U 相为例，U 相反电势与中性点经过比较器输出过零信号。

每一相反电势均存在由正到负以及由负到正的情况，因此三相共存在六种过零状态。为便于程序处理，`esp_sensorless_bldc_control` 将检测到的六种状态与下一次的换相动作映射：

表 3: 正转映射表

顺序	ZERO U	ZERO U	ZERO U	U 相状态	V 相状态	W 相状态
↑	0	0	1	上开下关	上关下开	上关下关
↑	0	1	1	上开下关	上关下关	上关下开
↑	0	1	0	上关下关	上开下关	上关下开
↑	1	1	0	上关下开	上开下关	上关下关
↑	1	0	0	上关下开	上关下关	上开下关
↑	1	0	1	上关下关	上关下开	上开下关

表 4: 反转映射表

顺序	ZERO U	ZERO U	ZERO U	U 相状态	V 相状态	W 相状态
↓	0	1	0	上关下开	上开下关	上关下关
↓	1	1	0	上关下开	上关下关	上开下关
↓	1	0	0	上关下关	上关下开	上开下关
↓	1	0	1	上开下关	上关下开	上关下关
↓	0	0	1	上开下关	上关下关	上关下开
↓	0	1	1	上关下关	上开下关	上关下开

#### 14.1.4 ESP Sensorless BLDC Control Components

本指南包含以下内容：

##### 目录

- *INJECT*
- *ALIGNMENT*
- *DRAG*
- *CLOSED\_LOOP*
  - ADC 采样检测过零点
  - 比较器检测过零点
  - 提前换向
- 堵转保护
- 速度控制
- API 参考
  - *Header File*
  - *Functions*
  - *Structures*
  - *Type Definitions*
  - *Enumerations*
  - *Header File*
  - *Macros*

`esp_sensorless_bldc_control` 组件是基于 ESP32 系列芯片的 BLDC 无感方波控制库。目前以及支持以下功能：

- 基于 ADC 采样检测过零点
- 基于支持比较器检测过零点

- 基于脉冲法实现转子初始相位检测
- 堵转保护

本文主要讲解如何使用 `esp_sensorless_bldc_control` 组件进行无刷电机开发，不涉及原理讲解，如需了解更多原理请参考

- [无刷电机控制概述](#) 无刷电机控制概述
- [基于 ADC 采样的无感方波电机控制](#) ADC 采样检测过零点
- [基于比较器检测的无感方波电机控制](#) 比较器检测过零点

无感方波控制流程主要可以分为以下部分

- INJECT: 注入阶段，通过脉振高频电压注入得到初始相位 INJECT
- ALIGNMENT: 对齐阶段，将转子固定到初始相位 ALIGNMENT
- DRAG: 强托阶段，通过六步换向将转子转动起来 DRAG
- CLOSED\_LOOP: 无感闭环控制，通过检测反电动势过零点进行换向 CLOSED\_LOOP
- BLOCKED: 电机堵转 BLOCKED
- STOP: 电机停止 STOP
- FAULT: 电机故障 FAULT

接下来会分别介绍各个部分的具体流程以及需要注意的参数

## INJECT

通过脉振注入法获取电机初始相位，需要在逆变电路的低端采集母线电流。如下图所示：

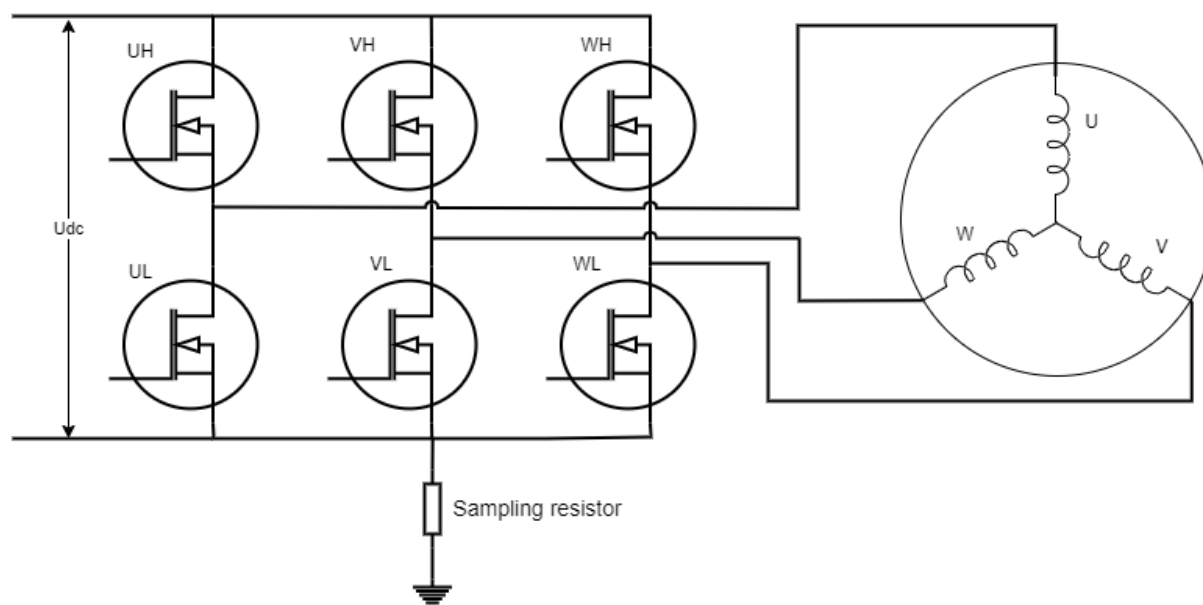


图 12: BLDC 母线电流采集

**备注：** 由于电流不能被直接采集到，因此通过一个采样电阻，可以将电流转化为电压。注意，电压需要转化到 ESP32 ADC 能够采集的范围。请参考：[ESP32 ADC](#)

由于电流只存在于上下管均导通的情况，因此需要在上管导通的时候进行 ADC 采样。将 MCPWM 配置为上升下降模式，并在计数器达到顶峰的时候进行采样，可以采集到准确的母线电压。

**备注：** LEDC 驱动不支持在高电平时触发回调，因此使用 LEDC 方式驱动的方案无法使用 'INJECT' 模式。

`INJECT_ENABLE` 为 1 时，开启 INJECT 模式，否则关闭。默认为 0。PWM 的生成模式必须为 MCPWM

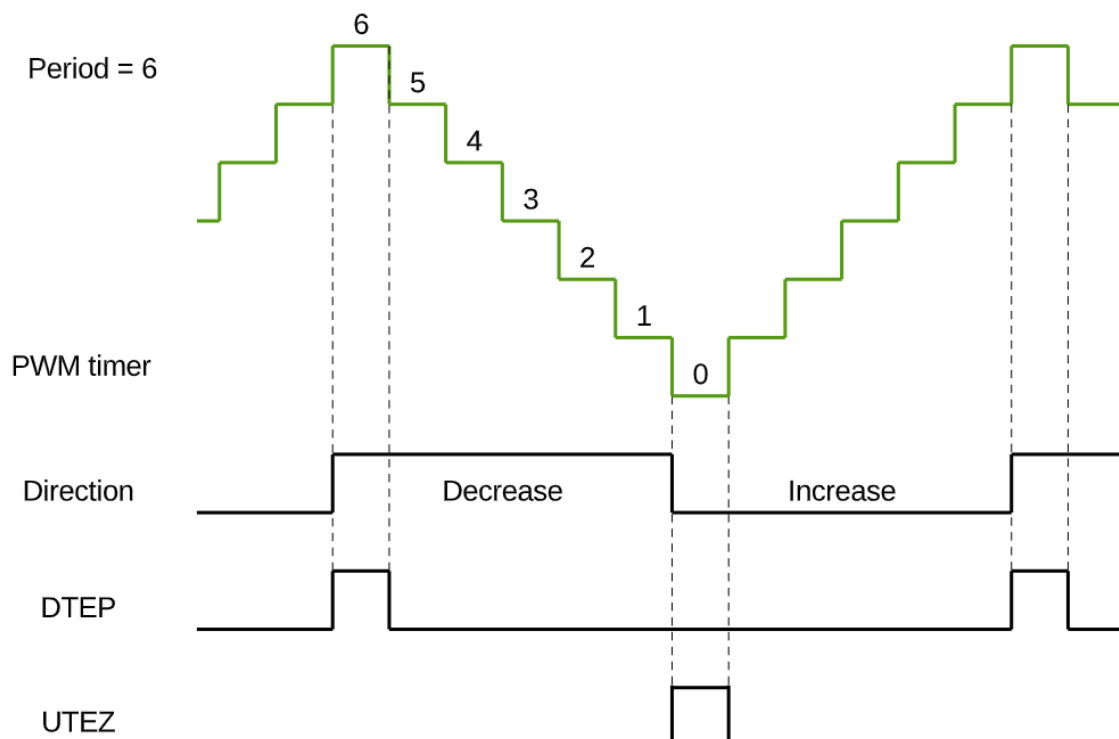


图 13: MCPWM 上升下降模式

*INJECT\_DUTY* 注入的电压大小，一般都是采用高占空比注入

*CHARGE\_TIME* 电感充电时间和脉冲注入时间，该值影响到初始相位检测的精准性。这个值太小会导致采集到的 ADC 值为 0，太大会导致 ADC 值过大。以手动旋转电机，在一圈中可以获得稳定的相位 1-6，不出现错误相位 0 和 7 为佳。

## ALIGNMENT

为保障无刷电机能够正常启动，需要确定转子在静止时的位置。在实际的应用中，通过在任意一组绕组上通电一定时间，将转子固定到固定相位，为后面的强拖做准备。

*ALIGNMENTNMS* 对齐时间，时间太长会过流。时间太短可能会导致转子没有对齐到正确的相位。

*ALIGNMENTDUTY* 对齐的力度。

## DRAG

通过六步换向将转子拖动起来，转子拖动采用升压升频的方式。逐渐的加大电压和换向频率，使电机具有初始速度，有明显的反电动势。以电机拖动过程中无异响，丝滑，无卡顿为佳。拖动时间无需太长。

*RAMP\_TIM\_STA* 拖动的初始延迟时间

*RAMP\_TIM\_END* 拖动的最小延迟时间

*RAMP\_TIM\_STEP* 拖动时间的步进

*RAMP\_DUTY\_STA* 拖动的初始占空比

*RAMP\_DUTY\_END* 拖动的最大占空比

*RAMP\_DUTY\_INC* 拖动占空比的步进

---

**备注：**强拖需要在电机工作环境下进行调参，电机空载参数不一定适用于带载情况

---

## CLOSED\_LOOP

**ADC 采样检测过零点** ADC 采样检测过零点需要采集悬空相电压和电机电源电压，且需要在上管导通的时候进行采集。

---

**备注：**采用 ADC 检测过零点，必须使用 MCPWM 作为驱动

---

`ENTER_CLOSE_TIME` 设置进入闭环的时间，默认强拖一段时间后即可进行闭环控制。

`ZERO_REPEAT_TIME` 连续 N 次检测到过零点才认为是过零点。

`AVOID_CONTINUE_CURRENT_TIME` 在换向后，会存在续电流影响，通过延迟检测规避掉续电流

**比较器检测过零点** 比较器检测过零点是通过硬件比较器比较悬空相反电动势和母线电压，通过 GPIO 检测比较器信号翻转来判断过零点。由于在实际过程中会有很多噪点，需要多次检测来确认过零点。

`ZERO_STABLE_FLAG_CNT` 多次检测到稳定过零点信号后，进入无感控制

`ZERO_CROSS_DETECTION_ACCURACY` 连续 N 次检测到相同信号视为稳定信号 0xFF 为 8 次，0xFFFF 为 16 次。当前支持的最大滤波次数为 0xFFFFFFFF，若依旧无法进入闭环状态，需要排查硬件问题。

---

**备注：**硬件排查方向主要包括采集三相端电压与比较器输出的滤波电容是否设置合理。

---

**提前换向** 过零点信号一般在换向前 30° 到来，当检测到过零点信号后，只需要延迟 30° 的时间即可。但在电机旋转过程中，电气周期不固定以及存在软件滤波和时延等原因，需要稍微补偿一下换向时间。

`ZERO_CROSS_ADVANCE` 提前换向时间，提前角度为  $180 / \text{ZERO\_CROSS\_ADVANCE}$ ，默认为 6

---

**备注：**换向角度并不是越提前约好，可以搭配示波器观测计算的换向角度与实际的换向角度是否一致。

---

## 堵转保护

电机长时间不换相即可视为堵转，此时会停止电机运行，进入堵转保护状态。

## 速度控制

通过 PID 控制速度，使电机达到设定的速度。

`SPEED_KP` 速度控制的 P 值

`SPEED_KI` 速度控制的 I 值

`SPEED_KD` 速度控制的 D 值

`SPEED_MIN_INTEGRAL` 速度控制的积分最小值

`SPEED_MAX_INTEGRAL` 速度控制的积分最大值

`SPEED_MIN_OUTPUT` 速度控制的输出最小值

`SPEED_MAX_OUTPUT` 速度控制的输出最大值，不超过最大占空比

`SPEED_CAL_TYPE` 位置式 PID 还是增量式 PID

*SPEED\_MAX\_RPM* 最大转速 RPM

*SPEED\_MIN\_RPM* 最小转速 RPM

*MAX\_SPEED\_MEASUREMENT\_FACTOR* 为了避免错误的速度检测，如果检测到的速度大于此设定系数，则视为错误速度检测。

## API 参考

### Header File

- [components/motor/esp\\_sensorless\\_bldc\\_control/include/bldc\\_control.h](#)

### Functions

**ESP\_EVENT\_DECLARE\_BASE** (BLDC\_CONTROL\_EVENT)

using `esp_event_handler_register()` to register BLDC\_CONTROL\_EVENT

esp event name

`esp_err_t bldc_control_init` (*bldc\_control\_handle\_t* \*handle, *bldc\_control\_config\_t* \*config)

init bldc control

#### 参数

- **handle** –pointer to bldc control handle
- **config** –pointer to bldc control config

**返回** ESP\_ERR\_INVALID\_ARG if handle or config is NULL ESP\_ERR\_NO\_MEM if memory allocation failed ESP\_OK on success ESP\_FAIL on other errors

`esp_err_t bldc_control_deinit` (*bldc\_control\_handle\_t* \*handle)

deinit bldc control

**参数** **handle** –pointer to bldc control handle ESP\_ERR\_INVALID\_ARG if handle or config is NULL ESP\_OK on success ESP\_FAIL on other errors

`esp_err_t bldc_control_start` (*bldc\_control\_handle\_t* \*handle, `uint32_t` expect\_speed\_rpm)

motor start

#### 参数

- **handle** –pointer to bldc control handle
- **expect\_speed\_rpm** –expect speed in rpm. This parameter does not work in case of open-loop control

**返回** ESP\_OK on success

`esp_err_t bldc_control_stop` (*bldc\_control\_handle\_t* \*handle)

motor stop

**参数** **handle** –pointer to bldc control handle

**返回** ESP\_FAIL if motor stop failed ESP\_OK on success

`dir_enum_t bldc_control_get_dir` (*bldc\_control\_handle\_t* \*handle)

get current motor direction

**参数** **handle** –pointer to bldc control handle

**返回** `dir_enum_t` current motor direction

`esp_err_t bldc_control_set_dir` (*bldc\_control\_handle\_t* \*handle, `dir_enum_t` dir)

set motor direction

#### 参数

- **handle** –pointer to bldc control handle
- **dir** –motor direction

**返回** ESP\_OK on success

`int bldc_control_get_duty (bldc_control_handle_t *handle)`

get current motor pwm duty

参数 **handle** –pointer to bldc control handle

返回 int current motor pwm duty

`esp_err_t bldc_control_set_duty (bldc_control_handle_t *handle, uint16_t duty)`

set motor pwm duty, Closed-loop speed control without calls

参数

- **handle** –pointer to bldc control handle
- **duty** –motor pwm duty

返回 ESP\_OK on success

`int bldc_control_get_speed_rpm (bldc_control_handle_t *handle)`

get current RPM

参数 **handle** –pointer to bldc control handle

返回 int current RPM

`esp_err_t bldc_control_set_speed_rpm (bldc_control_handle_t *handle, int speed_rpm)`

set motor RPM

参数

- **handle** –pointer to bldc control handle
- **speed\_rpm** –motor RPM

返回 ESP\_OK on success

## Structures

`struct bldc_debug_config_t`

Debug configuration, when activated, will periodically invoke the debug\_operation.

## Public Members

`uint8_t if_debug`

set 1 to open debug mode

`esp_err_t (*debug_operation)(void *handle)`

debug operation

`struct bldc_control_config_t`

BLDC Control Configuration.

## Public Members

`speed_mode_t speed_mode`

Speed Mode

`control_mode_t control_mode`

Control Mode

`alignment_mode_t alignment_mode`

Alignment Mode

`bldc_six_step_config_t` **six\_step\_config**

six-step phase change config

`bldc_zero_cross_comparer_config_t` **zero\_cross\_comparer\_config**

Comparator detects zero crossing config

*`bldc_debug_config_t`* **debug\_config**

debug config

### Type Definitions

`typedef void *bldc_control_handle_t`

blDC control handle

### Enumerations

`enum bldc_control_event_t`

*Values:*

enumerator **BLDC\_CONTROL\_START**

BLDC control start event

enumerator **BLDC\_CONTROL\_ALIGNMENT**

BLDC control alignment event

enumerator **BLDC\_CONTROL\_DRAG**

BLDC control drag event

enumerator **BLDC\_CONTROL\_STOP**

BLDC control stop event

enumerator **BLDC\_CONTROL\_CLOSED\_LOOP**

BLDC control closed loop event

enumerator **BLDC\_CONTROL\_BLOCKED**

BLDC control blocked event

`enum speed_mode_t`

*Values:*

enumerator **SPEED\_OPEN\_LOOP**

Open-loop speed control, speed control by setting pwm duty, poor load carrying capacity

enumerator **SPEED\_CLOSED\_LOOP**

Closed-loop speed control, rotational speed control via PID, high load carrying capacity

`enum control_mode_t`

*Values:*

enumerator **BLDC\_SIX\_STEP**

six-step phase change

enumerator **BLDC\_FOC**

foc phase change, not supported yet

### Header File

- [components/motor/esp\\_sensorless\\_bldc\\_control/user\\_cfg/bldc\\_user\\_cfg.h](#)

### Macros

**BLDC\_LEDC**

**BLDC\_MCPWM**

**PWM\_MODE**

Configure the generation of PWM.

Configure the generation of PWM.

**MCPWM\_CLK\_SRC**

MCPWM Settings.

Number of count ticks within a period  $\text{time\_us} = 1,000,000 / \text{MCPWM\_CLK\_SRC}$

**MCPWM\_PERIOD**

$\text{pwm\_cycle\_us} = 1,000,000 / \text{MCPWM\_CLK\_SRC} * \text{MCPWM\_PERIOD}$

**FREQ\_HZ**

LEDC Settings.

**DUTY\_RES**

Set duty resolution to 11 bits

**ALARM\_COUNT\_US**

No changes should be made here.

**DUTY\_MAX**

**PWM\_DUTYCYCLE\_05**

**PWM\_DUTYCYCLE\_10**

**PWM\_DUTYCYCLE\_15**

**PWM\_DUTYCYCLE\_20**

**PWM\_DUTYCYCLE\_25**



**PWM\_DUTYCYCLE\_30**

**PWM\_DUTYCYCLE\_40**

**PWM\_DUTYCYCLE\_50**

**PWM\_DUTYCYCLE\_60**

**PWM\_DUTYCYCLE\_80**

**PWM\_DUTYCYCLE\_90**

**PWM\_DUTYCYCLE\_100**

**INJECT\_ENABLE**

Pulse injection-related parameters.

---

**备注:** Used to detect the initial phase of the motor; MCPWM peripheral support is necessary. Whether to enable pulse injection.

---

**INJECT\_DUTY**

Injected torque.

**CHARGE\_TIME**

Capacitor charging and injection time.

**ALIGNMENTNMS**

Parameters related to motor alignment. Used to lock the motor in a specific phase before strong dragging.

Duration of alignment, too short may not reach the position, too long may cause the motor to overheat.

**ALIGNMENTDUTY**

alignment torque.

**RAMP\_TIM\_STA**

Setting parameters for strong dragging. The principle of strong dragging is to increase the control frequency and intensity.

---

**备注:** If the control cycle speeds up, corresponding reductions should be made to the RAMP\_TIM\_STA, RAMP\_TIM\_END, RAMP\_TIM\_STEP. The start step time for climbing. A smaller value results in faster startup but may lead to overcurrent issues.

---

**RAMP\_TIM\_END**

The end step time for climbing, adjusted based on the load. If loaded, this value should be relatively larger.

**RAMP\_TIM\_STEP**

Decremental increment for climbing step time—adjusted in accordance with RAMP\_TIM\_STA.

**RAMP\_DUTY\_STA**

The starting torque for climbing.

**RAMP\_DUTY\_END**

The ending torque for climbing.

**RAMP\_DUTY\_INC**

The incremental torque step for climbing—too small a value may result in failure to start, while too large a value may lead to overcurrent issues.

**ENTER\_CLOSE\_TIME**

ADC parameters for zero-crossing detection; please do not delete if not in use.

Enter the closed-loop state delay times.

**ZERO\_REPEAT\_TIME**

Change phase after detecting zero-crossing signals continuously for several times.

**AVOID\_CONTINUE\_CURRENT\_TIME**

Avoiding Continuous Current

**ZERO\_STABLE\_FLAG\_CNT**

Comparator parameters for zero-crossing detection; please do not delete if not in use.

After stable detection for multiple revolutions, it is considered to enter a sensorless state.

**ZERO\_CROSS\_DETECTION\_ACCURACY**

Count a valid comparator value every consecutive detection for how many times.

**ZERO\_CROSS\_ADVANCE**

Common parameter for compensated commutation time calculation.

Advance switching at zero-crossing, switching angle =  $180^\circ / \text{ZERO\_CROSS\_ADVANCE}$ . angle compensation should be provided.  $\geq 6$

**POLE\_PAIR**

Motor parameter settings.

Number of pole pairs in the motor.

**BASE\_VOLTAGE**

Rated voltage.

**BASE\_SPEED**

Rated speed unit: rpm.

**SPEED\_KP**

Closed-loop PID parameters for speed.

P

**SPEED\_KI**

I

**SPEED\_KD**

D

**SPEED\_MIN\_INTEGRAL**

Minimum integral saturation limit.

**SPEED\_MAX\_INTEGRAL**

Maximum integral saturation limit.

**SPEED\_MIN\_OUTPUT**

Minimum PWM duty cycle output.

**SPEED\_MAX\_OUTPUT**

Maximum PWM duty cycle output.

**SPEED\_CAL\_TYPE**

0 Incremental 1 Positional

**SPEED\_MAX\_RPM**

Speed parameter settings.

Maximum speed.

**SPEED\_MIN\_RPM**

Minimum speed.

**MAX\_SPEED\_MEASUREMENT\_FACTOR**

Supports a measured speed range from 0 to 1.2 times SpeedMAX. Large values could prevent proper filtering of incorrect data.

## 14.2 舵机

该组件使用 LEDC 外设产生 PWM 信号，可以实现对最多 16 路舵机的独立控制（ESP32 支持 16 路通道，ESP32-S2 支持 8 路通道），控制频率可选择为 50 ~ 400 Hz。使用该层 API，用户只需要指定舵机所在组、通道和目标角度，即可实现对舵机的角度操作。

舵机内部一般存在一个产生固定周期和脉宽的基准信号，通过与输入 PWM 信号进行比较，获得电压差输出，进而控制电机的转动方向和转动角度。常见的 180 度角旋转舵机一般以 20 ms (50 Hz) 为时钟周期，通过 0.5 ~ 2.5 ms 高电平脉冲，对应控制舵机在 0 ~ 180 度之间转动。

该组件可用于对控制精度要求较低的场景，例如玩具小车、遥控机器人、家庭自动化等。

### 14.2.1 使用方法

1. 舵机初始化：使用 `servo_init()` 对一组通道进行初始化，ESP32 包含 LEDC\_LOW\_SPEED\_MODE 和 LEDC\_HIGH\_SPEED\_MODE 两组通道，有些芯片可能只支持一组。初始化配置项主要包括最大角度、信号频率、最小输入脉宽和最大输入脉宽，用于计算角度和占空比的对应关系；引脚和通道用于分别指定芯片引脚和 LEDC 通道的对应关系；
2. 设置目标角度：使用 `servo_write_angle()` 通过指定舵机所在组、所在通道和目标角度，对舵机角度进行控制；

3. 读取当前角度：可使用 `servo_read_angle()` 获取舵机当前角度，需要注意的是该角度是根据输入信号进行推算的理论角度；
4. 舵机去初始化：当一组多个舵机都不再使用时，可使用 `servo_deinit()` 对一组通道进行去初始化。

### 14.2.2 应用示例

```
servo_config_t servo_cfg = {
    .max_angle = 180,
    .min_width_us = 500,
    .max_width_us = 2500,
    .freq = 50,
    .timer_number = LEDC_TIMER_0,
    .channels = {
        .servo_pin = {
            SERVO_CH0_PIN,
            SERVO_CH1_PIN,
            SERVO_CH2_PIN,
            SERVO_CH3_PIN,
            SERVO_CH4_PIN,
            SERVO_CH5_PIN,
            SERVO_CH6_PIN,
            SERVO_CH7_PIN,
        },
        .ch = {
            LEDC_CHANNEL_0,
            LEDC_CHANNEL_1,
            LEDC_CHANNEL_2,
            LEDC_CHANNEL_3,
            LEDC_CHANNEL_4,
            LEDC_CHANNEL_5,
            LEDC_CHANNEL_6,
            LEDC_CHANNEL_7,
        },
    },
    .channel_number = 8,
};

iot_servo_init(LEDC_LOW_SPEED_MODE, &servo_cfg);

float angle = 100.0f;

// Set angle to 100 degree
iot_servo_write_angle(LEDC_LOW_SPEED_MODE, 0, angle);

// Get current angle of servo
iot_servo_read_angle(LEDC_LOW_SPEED_MODE, 0, &angle);

//deinit servo
iot_servo_deinit(LEDC_LOW_SPEED_MODE);
```

### 14.2.3 API 参考

#### Header File

- [components/motor/servo/include/iot\\_servo.h](#)

#### Functions

`esp_err_t iot_servo_init` (ledc\_mode\_t speed\_mode, const *servo\_config\_t* \*config)

Initialize ledc to control the servo.

#### 参数

- **speed\_mode** –Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **config** –Pointer of servo configure struct

#### 返回

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_FAIL Configure ledc failed

`esp_err_t iot_servo_deinit` (ledc\_mode\_t speed\_mode)

Deinitialize ledc for servo.

**参数** **speed\_mode** –Select the LEDC channel group with specified speed mode.

#### 返回

- ESP\_OK Success

`esp_err_t iot_servo_write_angle` (ledc\_mode\_t speed\_mode, uint8\_t channel, float angle)

Set the servo motor to a certain angle.

---

**备注:** This API is not thread-safe

---

#### 参数

- **speed\_mode** –Select the LEDC channel group with specified speed mode.
- **channel** –LEDC channel, select from ledc\_channel\_t
- **angle** –The angle to go

#### 返回

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

`esp_err_t iot_servo_read_angle` (ledc\_mode\_t speed\_mode, uint8\_t channel, float \*angle)

Read current angle of one channel.

#### 参数

- **speed\_mode** –Select the LEDC channel group with specified speed mode.
- **channel** –LEDC channel, select from ledc\_channel\_t
- **angle** –Current angle of the channel

#### 返回

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

## Structures

struct **servo\_channel\_t**

Configuration of servo motor channel.

## Public Members

gpio\_num\_t **servo\_pin**[LEDC\_CHANNEL\_MAX]

Pin number of pwm output

ledc\_channel\_t **ch**[LEDC\_CHANNEL\_MAX]

The ledc channel which used

struct **servo\_config\_t**

Configuration of servo motor.

### Public Members

uint16\_t **max\_angle**

Servo max angle

uint16\_t **min\_width\_us**

Pulse width corresponding to minimum angle, which is usually 500us

uint16\_t **max\_width\_us**

Pulse width corresponding to maximum angle, which is usually 2500us

uint32\_t **freq**

PWM frequency

ledc\_timer\_t **timer\_number**

Timer number of ledc

*servo\_channel\_t* **channels**

Channels to use

uint8\_t **channel\_number**

Total channel number



## Chapter 15

# 安全 & 加密

## 15.1 Flash 加密

### 15.1.1 概述

- 使能 flash encryption 后，使用物理手段（如串口）从 SPI flash 中读取的数据都是经过加密的，大部分数据无法恢复出真实数据。
- flash encryption 使用 256-bit AES key 加密 flash 数据，key 保存在芯片的 efuse 中，生成之后变成软件读写保护。
- 用户烧写 flash 时烧写的是数据明文，第一次 boot 时，软件 bootloader 会对 flash 中的数据在原处加密。
- 一般使用情况下一共有 4 次机会通过串口烧写 flash，通过 OTA 更新 flash 数据没有次数限制。开发阶段可以在 menuconfig 中设置无烧写次数限制，但不要在产品中这么做。

### 15.1.2 使用步骤

1. make menuconfig 中选择 “Security features” ->” Enable flash encryption on boot”
2. 按通常操作编译出 bootloader, partition table 和 app image 并烧写到 flash 中
3. 第一次 boot 时 flash 中被指定加密的数据被加密（大的 partition 加密过程可能需要花费超过 1 分钟），之后就可以正常使用被加密的 flash 数据。

### 15.1.3 加密过程（第一次 boot 时进行）

1. bootloader 读取到 efuse 中的 FLASH\_CRYPT\_CNT 为 0，于是利用硬件随机数生成器产生加密用的 key，此 key 被保存在 efuse 中，对于软件是读写保护的。
2. bootloader 对所有需要被加密的 partition 在 flash 中原处加密
3. 默认情况下 efuse 中的 DISABLE\_DL\_ENCRYPT, DISABLE\_DL\_DECRYPT 和 DISABLE\_DL\_CACHE 会被烧写为 1，这样 UART bootloader 时就不能读取到解密后的 flash 数据
4. efuse 中的 FLASH\_CRYPT\_CONFIG 被烧写成 0xf，此标志用于决定加密 key 的多少位被用于计算每一个 flash 块（32 字节）对应的密钥，设置为 0xf 时使用所有 256 位
5. efuse 中的 FLASH\_CRYPT\_CNT 被烧写成 0x01，此标志用于 flash 烧写次数限制以及加密控制，详见 “FLASH\_CRYPT\_CNT” 一节
6. bootloader 将自己重启，从加密的 flash 执行软件 bootloader



### 15.1.4 串口重烧 flash (3 次重烧机会)

- 串口重烧 flash 过程
  1. make menuconfig 中选择 “Security features” -> “Enable flash encryption on boot”
  2. 编译工程，将所有之前加密的 images（包括 bootloader）烧写到 flash 中。
  3. 在 esp-idf 的 components/esptool\_py/esptool 路径下使用命令 `espefuse.py burn\_efuse FLASH\_CRYPT\_CNT` 烧写 efuse 中的 FLASH\_CRYPT\_CNT
  4. 重启设备，bootloader 根据 FLASH\_CRYPT\_CNT 的值重新加密 flash 数据。
- 若用户确定不再需要通过串口重烧 flash，可以在 esp-idf 的 components/esptool\\_py/esptool 路径下使用命令 `espefuse.py --port PORT write\_protect\_efuse FLASH\_CRYPT\_CNT` 将 FLASH\\_CRYPT\\_CNT 设置为读写保护（注意此步骤必须在 bootloader 已经完成对 flash 加密后进行）

### 15.1.5 FLASH\_CRYPT\_CNT

- FLASH\_CRYPT\_CNT 是 flash 加密方案中非常重要的控制标志，它是 8-bit 的值，它的值一方面决定 flash 中的值是否马上需要加密，另一方面控制 flash 烧写次数限制。
- 当 FLASH\_CRYPT\_CNT 有 (0,2,4,6,8) 位被烧写为 1 时，bootloader 会对 flash 中的内容进行加密。
- 当 FLASH\_CRYPT\_CNT 有 (1,3,5,7) 位被烧写为 1 时，bootloader 知道 flash 的内容已经过加密，直接读取 flash 中的数据解密后使用。
- FLASH\_CRYPT\_CNT 的变化过程：
  1. 没有使能 flash 加密时，永远是 0
  2. 使能了 flash 加密，在第一次 boot 时 bootloader 发现它的值是 0x00，于是知道 flash 中的数据还未加密，利用硬件随机数生成器产生 key，然后加密 flash，最后将它的最低位置 1（取值为 0x01）
  3. 后续 boot 时，bootloader 发现它的值是 0x01，知道 flash 中的数据已加密，可以解密后直接使用
  4. 用户需要串口重烧 flash，于是使用命令行手动烧写 FLASH\_CRYPT\_CNT，此时 2 个 bit 被置为 1（取值为 0x03）
  5. 重启设备，bootloader 发现 FLASH\_CRYPT\_CNT 的值是 0x03（2 bit 1），于是重新加密 flash 数据，加密完成后 bootloader 将 FLASH\_CRYPT\_CNT 烧写为 0x07（3 bit 1），flash 加密正常使用
  6. 用户需要串口重烧 flash，于是使用命令行手动烧写 FLASH\_CRYPT\_CNT，此时 4 个 bit 被置为 1（取值为 0x0f）
  7. 重启设备，bootloader 发现 FLASH\_CRYPT\_CNT 的值是 0x0f（4 bit 1），于是重新加密 flash 数据，加密完成后 bootloader 将 FLASH\_CRYPT\_CNT 烧写为 0x1f（5 bit 1），flash 加密正常使用
  8. 用户需要串口重烧 flash，于是使用命令行手动烧写 FLASH\_CRYPT\_CNT，此时 6 个 bit 被置为 1（取值为 0x3f）
  9. 重启设备，bootloader 发现 FLASH\_CRYPT\_CNT 的值是 0x4f（6 bit 1），于是重新加密 flash 数据，加密完成后 bootloader 将 FLASH\_CRYPT\_CNT 烧写为 0x7f（7 bit 1），flash 加密正常使用
  10. 注意！此时不能再使用命令行烧写 FLASH\_CRYPT\_CNT，bootloader 读到 FLASH\_CRYPT\_CNT 为 0xff（8 bit 1）时，会停止后续的 boot。

### 15.1.6 被加密的数据

- Bootloader
- Secure boot bootloader digest（若 Secure Boot 被使能，flash 中会多出这一项，具体查看 “Secure Boot” 中 “执行过程” 的步骤 3）
- Partition table
- Partition table 中指向的所有 Type 域标记为 “app” 的部分
- Partition table 中指向的所有 Flags 域标记为 “encrypted” 的部分（用于非易失性存储（NVS）部分的 flash 在任何情况下都不会被加密）

### 15.1.7 哪些方式读到解密后的数据（真实数据）

- 通过内存管理单元的 flash 缓存读取的 flash 数据都是经过解密后的数据，包括：
  - flash 中的可执行应用程序代码

- 存储在 flash 中的只读数据
- 任何通过 API `esp_spi_flash_mmap()` 读取的数据
- 由 ROM bootloader 读取的软件 bootloader image 数据
- 如果调用 API `esp_partition_read()` 读取被加密区域的数据，则读取的 flash 数据是经过解密后的数据

### 15.1.8 哪些方式读到不解密的数据（无法使用的脏数据）

- 通过 API `esp_spi_flash_read()` 读取的数据
- ROM 中的函数 `SPIRead()` 读取的数据

### 15.1.9 软件写入加密数据

- 调用 API `esp_partition_write()` 时，只有写到被加密的 partition 的数据才会被加密
- 函数 `esp_spi_flash_write()` 根据参数 `write_encrypted` 是否被设为 `true` 决定是否对数据加密
- ROM 函数 `esp_rom_spiflash_write_encrypted()` 将加密后的数据写入 flash 中，而 `SPIWrite()` 将不加密的数据写入到 flash 中

## 15.2 安全启动

### 15.2.1 概述

- Secure Boot 的目的是保证芯片只运行用户指定的程序，芯片每次启动时都会验证从 flash 中加载的 partition table 和 app images 是否是用户指定的
- Secure Boot 中采用 ECDSA 签名算法对 partition table 和 app images 进行签名和验证，ECDSA 签名算法使用公钥/私钥对，私钥用于对指定的二进制文件签名，公钥用于验证签名
- 由于 partition table 和 app images 是在软件 bootloader 中被验证的，所以为了防止攻击者篡改软件 bootloader 从而跳过签名验证，Secure Boot 过程中会在 ROM bootloader 时检查软件 bootloader image 是否被篡改，检查用到的 secure boot key 由硬件随机数生成器产生，保存在 efuse 中，对于软件是读写保护的

### 15.2.2 所用资源

- ECDSA 算法公钥/私钥对
  - 烧写 flash 前在 PC 端生成
  - 公钥会被编译到 bootloader image 中，软件 bootloader 在执行时会读取公钥，使用公钥验证 flash 中 partition table 和 app images 是否是经过相应的私钥签名的
  - 私钥在编译时被用于对 partition table 和 app images 签名，私钥必须被保密好，一旦泄露任何使用此私钥签名的 image 都能通过 boot 时的签名验证
- secure bootloader key
  - 这是一个 256-bit AES key，在第一次 Secure Boot 时由硬件随机数生成，保存在 efuse 中，软件无法读取
  - 使用此 key 验证软件 bootloader image 是否被修改

### 15.2.3 执行过程

1. 编译 bootloader image 时发现 menuconfig 中使能了 secure boot，于是根据 menuconfig 中指定的公钥/私钥文件路径将公钥编译到 bootloader image 中，bootloader 被编译成支持 secure boot
2. 编译 partition table 和 app images 时使用私钥计算出签名，将签名编译到相应的二进制文件中
3. 芯片第一次 boot 时，软件 bootloader 根据以下步骤使能 secure boot：
  - 硬件产生一个 secure boot key，将这个 key 保存在 efuse 中，利用这个 key、一个随机数 IV 和 bootloader image 计算出 secure digest

- secure digest 与随机数 IV 保存在 flash 的 0x0 地址，用于在后续 boot 时验证 bootloader image 是否被篡改
  - 若 menuconfig 中选择了禁止 JTAG 中断和 ROM BASIC 中断，bootloader 会将 efuse 中的一些标志位设置为禁止这些中断（强烈建议禁止这些中断）
  - bootloader 通过烧写 efuse 中的 ABS\_DONE\_0 永久使能 secure boot
4. 芯片在后面的 boot 中，ROM bootloader 发现 efuse 中的 ABS\_DONE\_0 被烧写，于是从 flash 的地址 0x0 读取第一次 boot 时保存的 secure digest 和随机数 IV，硬件使用 efuse 中的 secure boot key、随机数 IV 与当前的 bootloader image 计算当前的 secure digest，若与 flash 中的 secure digest 不同，则 boot 不会继续，否则就执行软件 bootloader。
  5. 软件 bootloader 使用 bootloader image 中保存的公钥对 flash 中的 partition table 和 app images 签字进行验证，验证成功之后才会 boot 到 app 代码中

## 15.2.4 使用步骤

1. make menuconfig 选择 “enable secure boot in bootloader”
2. make menuconfig 设置保存公钥/秘钥对的文件
3. 生成公钥和秘钥，先执行 “make” 命令，此时由于还没有公钥/秘钥对，所以命令行中会提示生成公钥/秘钥对的命令，按提示执行命令即可。但在产品级使用中，建议使用 openssl 或者其他工业级加密程序生成公钥/秘钥对。例如使用 openssl: “openssl ecparam -name prime256v1 -genkey -noout -out my\_secure\_boot\_signing\_key.pem”（若使用现有的公钥/秘钥对文件，可以跳过此步）
4. 运行命令 “make bootloader” 产生一个使能 secure boot 的 bootloader image
5. 执行完 4 后命令行会提示下一步烧写 bootloader image 的命令，按提示烧写即可
6. 运行命令 “make flash” 编译并烧写 partition table 和 app images
7. 重启芯片，软件 bootloader 会使能 secure boot，查看串口打印确保 secure boot 成功启用。

## 15.2.5 注意事项

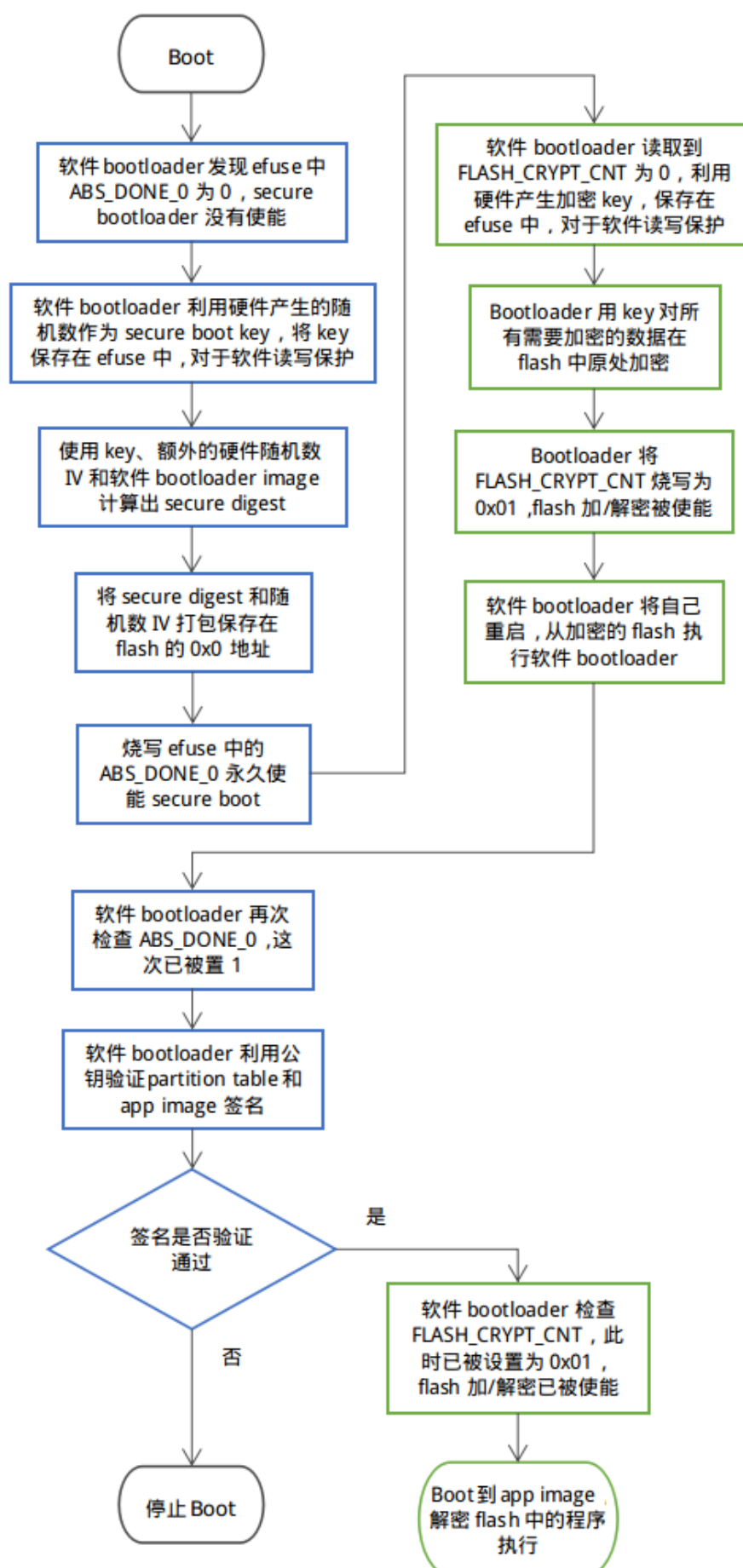
- 正常使用情况下，bootloader image 只能烧写一次，partition table 和 app images 可以重复烧写
- 秘钥必须保密，一旦泄露 secure boot 将失去作用
- 用于 OTA 的 image 必须进行秘钥签名，OTA 时会使用公钥进行验证
- 在默认设置下，bootloader 的从 0x1000 地址开始，最大长度为 28KB (bootloader)。如果发现 Secure Boot 发送错误，请先检查是否因为 Bootloader 地址过大。通过在 menuconfig 中调整 Bootloader 的 log 等级，可以有效降低编译后的 Bootloader 大小。

## 15.2.6 可重复烧写 bootloader

- 默认情况下 bootloader image 只能烧写一次，在产品中强烈建议这样做，因为 bootloader image 可以重新烧写的情况下可以通过修改 bootloader 跳过后续 image 的验证过程，这样 secure boot 就失去作用
- 可重复烧写 bootloader 模式下，secure bootloader key 是在 PC 端产生的，此 key 必须保密，一旦 key 被泄露，其它使用此 key 生成 digest 的 bootloader image 也能通过硬件检查
- 使用步骤：
  1. make menuconfig 中选择 “secure bootloader mode” -> “Reflashable”
  2. 按 “使用步骤” 一节步骤 2 和 3 生成公钥与秘钥
  3. 运行指令 “make bootloader”，一个 256-bit secure boot key 会根据用于签名的私钥计算出，命令行会打印两个后续步骤，按循序执行：
    - 将 PC 端生成的 secure boot key 烧入 efuse 中的命令
    - 将编译好的带有预计算出的 secure digest 的 bootloader image 烧写到 flash 中
  4. 从 “使用步骤” 一节的步骤 6 继续执行

## 15.2.7 Secure Boot 与 Flash Encryption 流程图

- 第一次 boot 时 secure boot 与 flash encrypt 的生效过程如下图所示，图中蓝色框是 secure boot 的步骤，绿色框是 flash encrypt 的步骤



- 后续 boot 时流程图如下，图中绿色框中的步骤会执行解密，解密是由硬件自动完成的

### 15.2.8 开发阶段使用可重复烧写 flash 的 Secure Boot 与 Flash encryption

1. make menuconfig 中使能 secure boot 和 flash encrypt, “Secure bootloader mode” 选择 “Reflashable”, 并设置你的公钥/私钥.pem 文件路径
2. 编译 bootloader 并生成 secure boot key:

```
make bootloader
```

3. 使用 key 和 bootloader 计算带 digest 的 bootloader

```
python $IDF_PATH/components/esptool_py/esptool/espsecure.py digest_secure_
↳bootloader --keyfile ./build/bootloader/secure_boot_key.bin -o ./build/
↳bootloader/bootloader_with_digest.bin ./build/bootloader/bootloader.bin
```

4. 编译 partition\_table 与 app

```
make partition_table
make app
```

5. 加密三个 bin 文件

```
python $IDF_PATH/components/esptool_py/esptool/espsecure.py encrypt_flash_data_
↳--keyfile flash_encrypt_key.bin --address 0x0 -o build/bootloader/bootloader_
↳digest_encrypt.bin build/bootloader/bootloader_with_digest.bi
python $IDF_PATH/components/esptool_py/esptool/espsecure.py encrypt_flash_data_
↳--keyfile flash_encrypt_key.bin --address 0x8000 -o build/partitions_
↳singleapp_encrypt.bin build/partitions_singleapp.bin
python $IDF_PATH/components/esptool_py/esptool/espsecure.py encrypt_flash_data_
↳--keyfile flash_encrypt_key.bin --address 0x10000 -o build/iot_encrypt.bin_
↳build/iot.bin
```

6. 烧写三个加密后的 bin 文件

```
python $IDF_PATH/components/esptool_py/esptool/esptool.py --baud 1152000 write_
↳flash 0x0 build/bootloader/bootloader_digest_encrypt.bin
python $IDF_PATH/components/esptool_py/esptool/esptool.py --baud 1152000 write_
↳flash 0x8000 build/partitions_singleapp_encrypt.bin
python $IDF_PATH/components/esptool_py/esptool/esptool.py --baud 1152000 write_
↳flash 0x10000 build/iot_encrypt.bin
```

7. 将 flash\_encryption\_key 烧入 efuse (仅在第一次 boot 前烧写):

```
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_key flash_
↳encryption flash_encrypt_key.bin
```

8. 将 secure boot key 烧入 efuse (仅在第一次 boot 前烧写) :

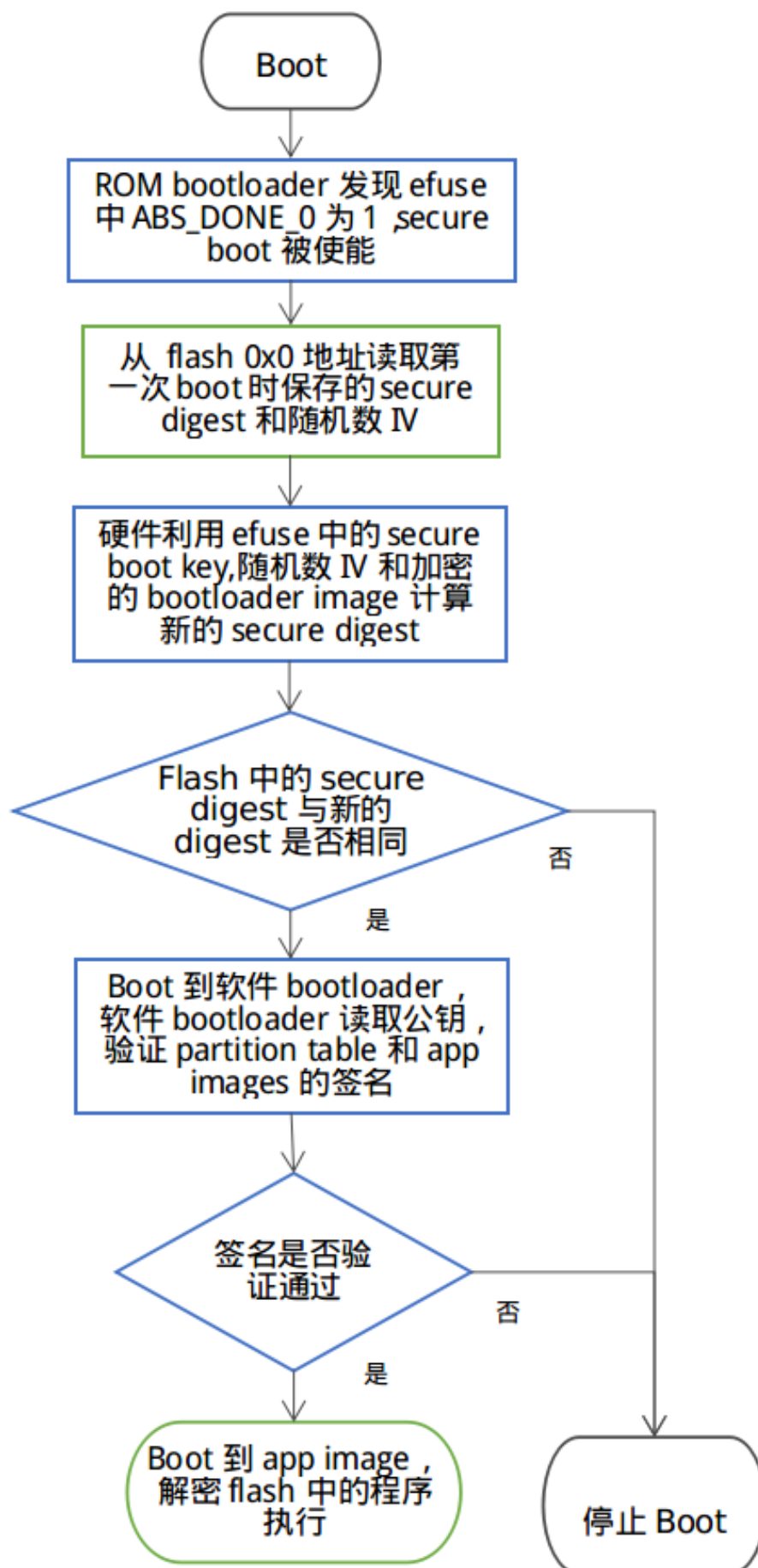
```
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_key secure_
↳boot ./build/bootloader/secure_boot_key.bin
```

9. 烧写 efuse 中的控制标志 (仅在第一次 boot 前烧写)

```
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse ABS_DONE_
↳0
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse FLASH_
↳CRYPT_CNT
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse FLASH_
↳CRYPT_CONFIG 0xf
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse DISABLE_
↳DL_ENCRYPT
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse DISABLE_
↳DL_DECRYPT
python $IDF_PATH/components/esptool_py/esptool/espefuse.py burn_efuse DISABLE_
↳DL_CACHE
```

(下页继续)





## 15.3 启用安全加密的生产方案

### 15.3.1 Windows 平台的下载工具

- 乐鑫提供 windows 平台的下载工具，能够在工厂生产环境中批量烧写固件
- 生产下载工具的配置文件在 `configure` 文件夹内，涉及安全特性的配置在 `security.conf` 中，目前涉及的配置内容如下表：

ITEM	Function	de-fault
<code>debug_enable</code>	是否开启 debug 模式，在 debug 模式下，工具会根据 pem 文件产生相同密钥，否则随机生成密钥	True
<code>debug_pem_path</code>	设置证书地址，用于生成可重复烧写的密钥，尽在 debug 模式下有效	
SECURE BOOT		
<code>secure_boot_en</code>	开启 secure boot 功能	False
<code>burn_secure_boot_key</code>	使能 secure boot key 烧写	False
<code>secure_boot_force_write</code>	是否不检查 secure boot key block，强制烧写 key	False
<code>secure_boot_rw_protect</code>	开启 secure boot key 区域的读写保护	False
FLASH ENCRYPTION		
<code>flash_encryption_en</code>	开启 flash 加密功能	False
<code>burn_flash_encryption_key</code>	使能 flash encrypt key 烧写	False
<code>flash_encrypt_force_write</code>	是否不检查 flash encrypt key block，强制烧写 key	False
<code>flash_encrypt_rw_protect</code>	开启 flash encrypt key 区域的读写保护	False
AES KEY	Not used yet	
DISABLE FUNC		
<code>jtag_disable</code>	是否关闭 JTAG 调试功能	False
<code>dl_encrypt_disable</code>	是否关闭下载模式下 flash 加密功能	False
<code>dl_decrypt_disable</code>	是否关闭下载模式下 flash 解密功能	False
<code>dl_cache_disable</code>	是否关闭下载模式下的 flash cache 功能	False

- 下载工具的内部逻辑和流程如下：

### 15.3.2 操作步骤

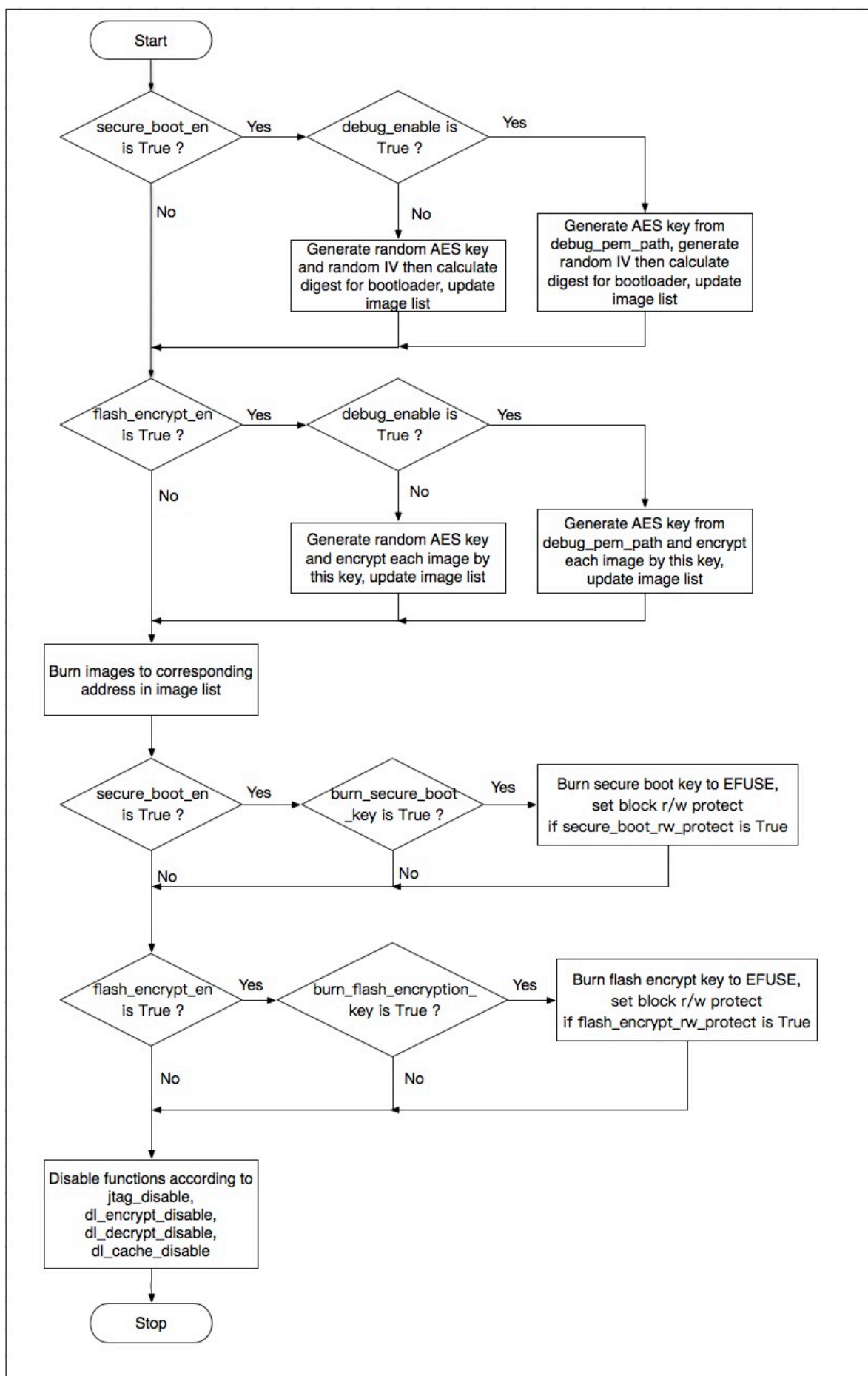
#### 准备工作

- 安装 esptool
  - esptool 默认路径在 `$IDF_PATH/components/esptool_py/esptool/`
  - 也可以通过 python 安装：

```
pip install esptool
或者
pip3 install esptool
```

#### 方案 1: 通过 bootloader 完成 security 特性初始化

- 优势: 可以批量进行 flash 烧录，初始化的固件相同，密钥在第一次上电有在设备内随机生成。





- 缺陷: 设备在首次初始化过程所用时间较长, 如果在首次初始化过程发生掉电等意外情况, 设备可能无法正常启动。
- 由芯片端自动随机生成 secure boot 与 flash encryption 密钥, 并写入芯片 efuse 中, 密钥写入后, 对应的 efuse block 会被设置为读写保护状态, 软件与工具都无法读取密钥。
- 所有编译出的 images 都按正常情况烧写, 芯片会在第一次 boot 时进行配置。
- 通过 make menuconfig 配置 secure boot 和 flash encryption, 按照第一、二节介绍的步骤执行即可, 具体操作步骤如下, 如果了解第一、二节的内容, 可以跳过:

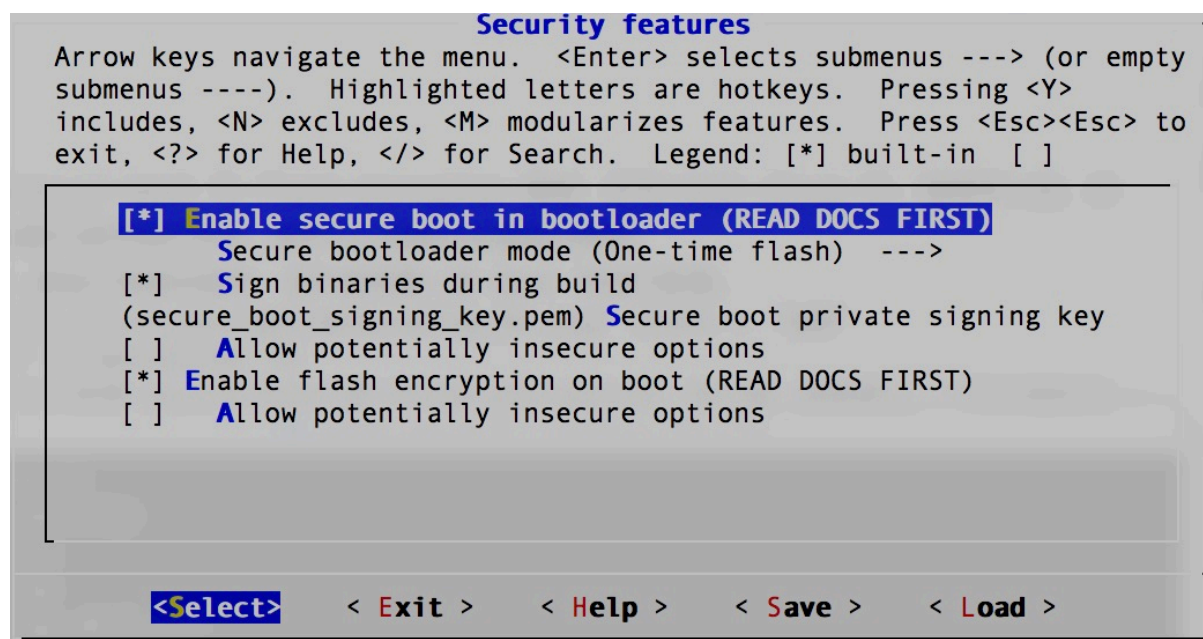
1. 随机生成 RSA 密钥文件:

```

espsecure.py generate_signing_key secure_boot_signing_key.pem
or
openssl ecparam -name prime256v1 -genkey -noout -out secure_boot_signing_key.
↪pem

```

2. 在 menuconfig 中, 选择 Sign binaries during build, 并指定刚才生成的密钥路径, 如下图。



3. 分别编译 bootloader 与应用代码

```

make bootloader
make

```

4. 使用 esptool 将编译生成的 bin 文件写入 flash 对应地址, 以 example 中 hellow-world 工程为例:

```

bootloader.bin --> 0x1000
partition.bin --> 0x8000
app.bin --> 0x10000
python $IDF_PATH/components/esptool_py/esptool/esptool.py --chip esp32 --
↪port /dev/cu.SLAB_USBtoUART --baud 115200 --before default_reset --
↪after no_reset write_flash -z --flash_mode dio --flash_freq 40m --flash_
↪size detect 0x1000 $IDF_PATH/esp-idf/examples/get-started/hello_world/
↪build/bootloader/bootloader.bin 0xf000 $IDF_PATH/esp-idf/examples/get-
↪started/hello_world/build/phy_init_data.bin 0x10000 $IDF_PATH/examples/
↪get-started/hello_world/build/hello-world.bin 0x8000 $IDF_PATH/examples/
↪get-started/hello_world/build/partitions_singleapp.bin

```

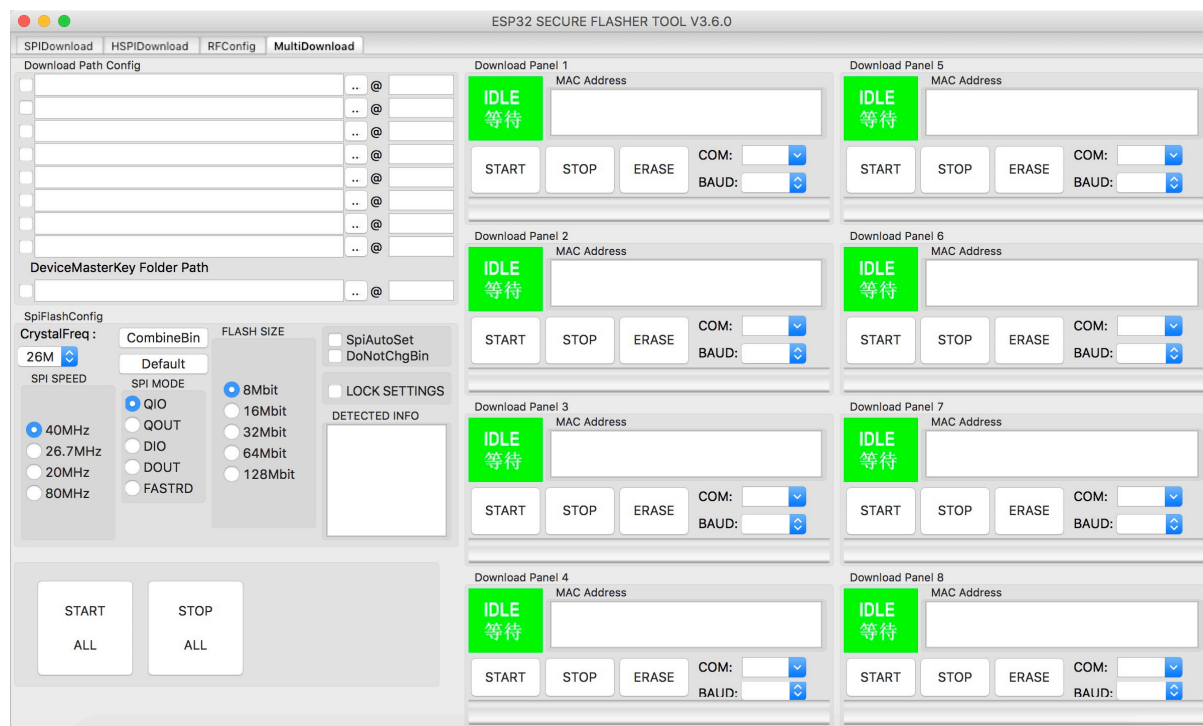
**备注:** 以上命令仅是示例代码, 请在使用时, 替换其中的文件路径以及所选参数, 包括串口、波特率、SPI 模式和频率等。

5. 我们也可以使用 window 平台的下载工具来完成工厂下载。需要在配置文件中, 关闭工具的 security 功能, 这样工具端就不会操作 security 相关特性, 完全由硬件和 bootloader 来完成初始

化：

```
[SECURE BOOT]
secure_boot_en * False
[FLASH ENCRYPTION]
flash_encryption_en * False
```

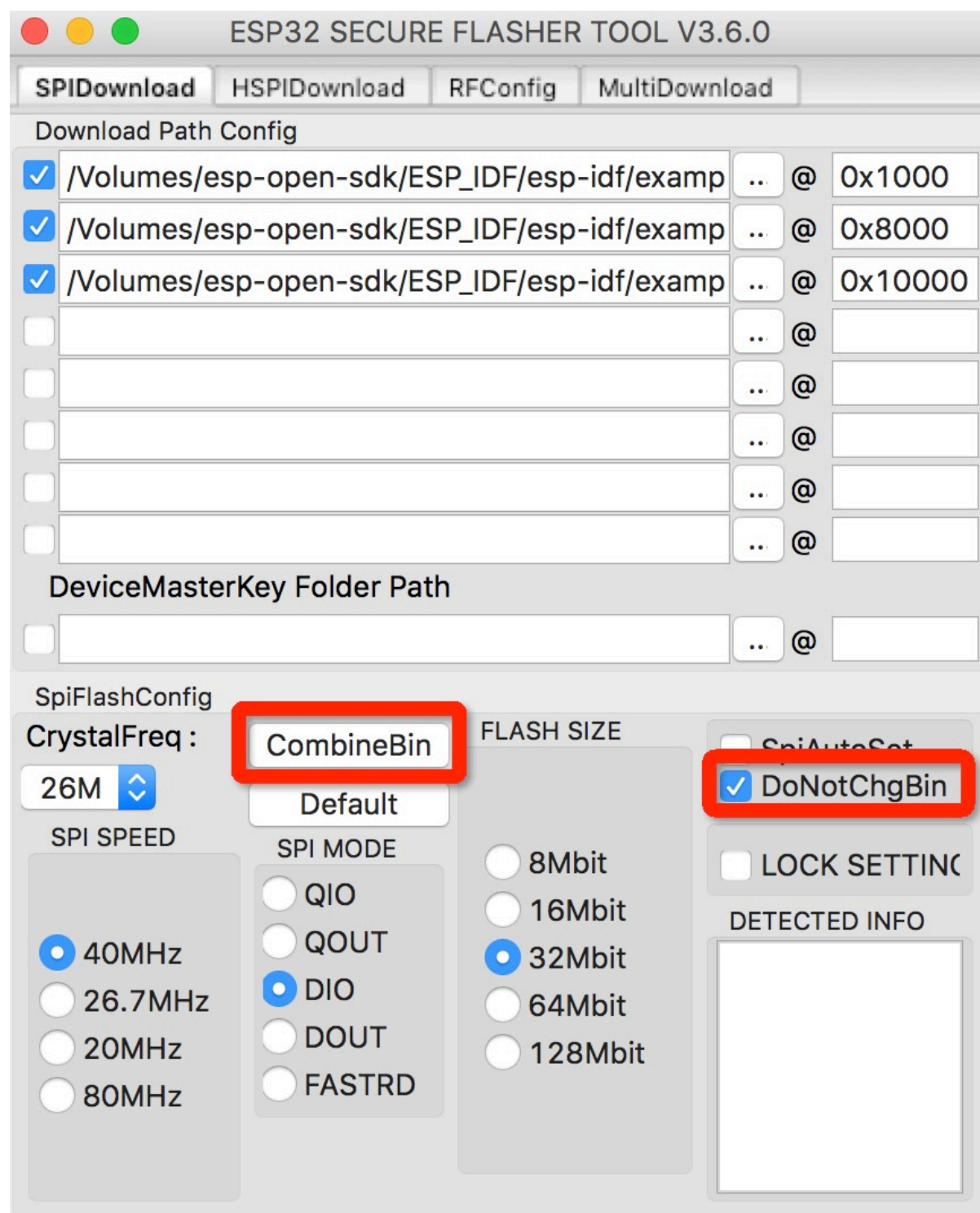
**备注：** 修改并保存参数前，请先关闭下载工具，配置文件修改完成并保存后，再开启运行下载工具。



6. 或者我们可以通过下载工具的 combine 功能，将多个 bin 文件打包为一个文件，再由工厂 flash 烧录器烧进 flash 进行批量生产。
  - 选择 bin 文件并制定 flash 中的地址
  - 选中 ‘DoNotChgBin’ 选项，这样工具不会对 bin 文件的配置 (SPI 模式速率等) 进行任何修改。
  - 点击 ‘CombineBin’ 按键，生产合并后的 bin 文件。
  - 在 ‘combine’ 文件夹下，生成 target.bin，将其烧写到 Flash 的 0x0 地址即可。
  - 工具只会对填写的最大地址范围内的空白区域填充 0xff。并将文件按地址组合。
7. 下载完成后，需要运行一次程序，使 bootloader 完成 security 相关特性的初始化，包括 AES 密钥的随机生成并写入 EFUSE，以及对明文的 flash 进行首次加密。

**备注：** 请误在首次启动完成前，将芯片断电，以免造成芯片无法启动的情况。

- 注意事项：
  - 用于签名的私钥需要保密，如果泄漏，app.bin 有被伪造的可能性。
  - 使用者不能遗失私钥，必须使用私钥用于对 OTA app 签名 (如果有 OTA 功能)。
  - 芯片通过软件 bootloader 对 flash 加密是一个比较缓慢的过程，对于较大的 partition 可能需要花费一分钟左右
  - 若第一次执行 bootloader，flash 加密进行到一半芯片掉电
    - \* 没有使能 secure boot 时，可重新将 images 明文烧写到 flash 中，让芯片下次 boot 时重新加密 flash
    - \* 使能了 secure boot 时，由于无法重新烧写 flash，芯片将永久无法 boot



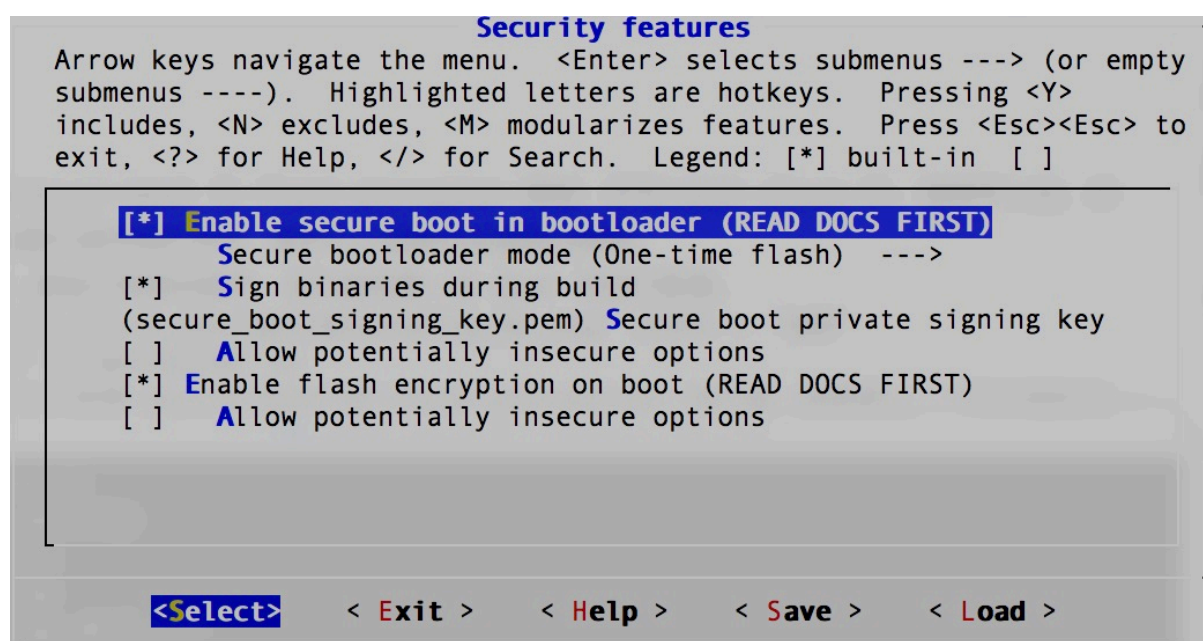
**方案 2: 通过下载工具初始化 security 特性**

- 优势: 工具进行密钥的随机生成, 直接将 image 密文烧写进 flash, 然后配置 efuse. 避免过程中掉电造成无法启动的情况。
- 缺陷: 每个设备必须通过下载工具进行烧写, 因为密钥不同, 无法预先烧写相同的固件到 flash 中。
- 使用下载工具应用 secure boot 和 flash encryption, 这时用户只需要在 make menuconfig 中选择 “enable secure boot in bootloader” 并设置公钥/私钥路径即可
- 下载工具在运行时, 会随机产生 secure boot 与 flash encryption 密钥, 并烧写到对应的 EFUSE 位置中。
- 操作步骤:

1. 随机生成 RSA 密钥文件, 用于签名固件:

```
esptool.py generate_signing_key secure_boot_signing_key.pem
or
openssl ecparam -name prime256v1 -genkey -noout -out secure_boot_signing_
key.pem
```

2. 在 menuconfig 中, 选择 Sign binaries during build, 并指定刚才生成的密钥路径, 如下图。



3. 分别编译 bootloader 与应用代码

```
make bootloader
make
```

4. 设置下载工具的安全配置文件

```
[DEBUG MODE]
debug_enable * False
    ↳ #关闭debug模式, 工具随机生成密钥。否则根据pem文件产生相同密钥
debug_pem_path *
    ↳ #debug模式下, 设置证书地址, 用于生成可重复烧写的密钥
[SECURE BOOT]
secure_boot_en * True                #开启secure boot功能
burn_secure_boot_key * True          #使能secure boot key烧写
secure_boot_force_write * False      #是否不检查secure boot key
    ↳ block, 强制烧写key
secure_boot_rw_protect * True        #开启secure boot key区域的读写保护
[FLASH ENCRYPTION]
flash_encryption_en * True           #开启flash加密功能
burn_flash_encryption_key * True     #使能flash encrypt key烧写
flash_encrypt_force_write * False    #是否不检查flash encrypt key
    ↳ block, 强制烧写key
```

(下页继续)

(续上页)

```

flash_encrypt_rw_protect * True      #开启flash encrypt key区域的读写保护
[AES KEY]
aes_key_en * False                  #目前未实现，仅保留该选项
burn_aes_key * False                #目前未实现，仅保留该选项
[DISABLE FUNC]
jtag_disable * True                 #是否关闭JTAG调试功能
dl_encrypt_disable * True            #是否关闭下载模式下flash加密功能
dl_decrypt_disable * True            #是否关闭下载模式下flash解密功能
dl_cache_disable * True              #是否关闭下载模式下的flash cache功能

```

注意：

修改并保存参数前，请先关闭下载工具，配置文件修改完成并保存后，再开启运行下载工具。

5. 使用下载工具进行下载，若不希望工具修改任何配置参数（比如 flash 频率和模式），请勾选 ‘DoNotChgBin’ 选项。下载工具会更具配置文件的设置，在下载过程中完成固件加密下载和密钥随机生成与烧写。

- 注意事项：

- 用于签名的私钥需要保密，如果泄漏，app.bin 有被伪造的可能性。
- 使用者不能遗失私钥，必须使用私钥用于对 OTA app 签名（如果有 OTA 功能）。
- 用户可以选择不开启 app image 的签名校验，只需要关闭 menuconfig 中的 secure boot 功能即可。下载工具会更具配置文件，通过 efuse 启用 secure boot。禁用 app image 的签名校验会存在安全隐患。



## Chapter 16

# 其它资源

### 16.1 GPIO 扩展

随着 ESP32 系列芯片应用领域的进一步扩展，更多需求多样的应用场景正在被不断导入，包括一些对 GPIO 数量有较大要求的场景。乐鑫后续发布的 ESP32-S2 等产品包含多达 43 个 GPIO，可大大缓解 GPIO 资源紧张的问题。如果依旧无法满足需求，还可通过为 ESP32 添加 GPIO 扩展芯片解决，例如使用基于 I2C 接口控制的 GPIO 扩展模块 MCP23017，每个模块可外扩 16 个 GPIO 口，同时可挂载多达 8 个扩展模块，即可额外扩展 128 个 GPIO 口。

#### 16.1.1 已适配列表

名称	功能	总线	供应商	规格书	驱动
MCP23017	16-bit I/O expander	I2C	Microchip	<a href="#">规格书</a>	<a href="#">mcp23017</a>

### 16.2 ADC 扩展量程方案

#### 16.2.1 ESP32-S3 ADC 扩展量程方案

ESP32-S3 ADC 最大有效量程为 0 ~ 3100 mV，通过外部分压电路，能够满足大部分的 ADC 按键、电池电压检测等功能。但是对于 NTC 等应用场景，可能需要支持满量程（0 ~ 3300 mV）测量。ESP32-S3 可通过配置寄存器调整 ADC 的偏置电压，再结合高电压区域非线性补偿方法，我们可以实现对 ESP32-S3 量程的扩展。

实现量程扩展的过程为：

1. 使用默认偏置，测量一次电压值
2. 当测量电压值小于 2900 mV，直接输出测量电压值作为测量结果
3. 当测量电压值大于 2900 mV，提升偏置电压，进行二次测量，并进行非线性修正计算，输出修正值作为测量结果
4. 测量完成，恢复偏置电压

因此，每次 ADC 测量，期间可能读取 1~2 次实际电压值。对于大部分的应用场景，该方案引入的测量延迟可以忽略不计。

## 16.2.2 Patch 使用方法

当前 Patch 基于 ESP-IDF v4.4.8 开发：

1. 确认 ESP-IDF 已经 checkout 到 v4.4.8
2. 下载 esp32s3\_adc\_range\_to\_3100.patch 文件
3. 使用指令 `git am --signoff < esp32s3_adc_range_to_3100.patch` 将 Patch 应用到 IDF 中
4. 请注意，该方案仅对 `esp_adc_cal_get_voltage` 接口有效，用户可直接调用该接口获取扩展后的读数

## 16.2.3 API 使用说明

1. 如果想要读取量程扩展后的电压值，用户必须使用 `esp_adc_cal_get_voltage` 直接获取 ADC1 或 ADC2 的通道电压。
2. ESP-IDF ADC 其它 API 不受影响，读取的结果和默认结果一致

## 16.3 过零检测

过零信号检测驱动程序是一个设计用于分析过零信号的组件。通过检查过零信号的周期和触发边缘，它可以确定信号的有效性、无效性、是否超出预期的频率范围以及是否存在信号丢失等情况。

过零信号检测组件支持检测两种类型的过零信号。

- 方波形：在信号经过零时翻转当前的电平。
- 脉冲型：在信号经过零时生成的脉冲。

由于响应延迟等因素，该组件支持两种驱动模式，包括 GPIO 中断和 MCPWM 捕获。

用户可以灵活配置组件的驱动模式，并调整诸如有效频率范围和有效信号判断次数等参数。这提供了很高的灵活性。

该程序以事件的形式返回结果和数据，满足用户对信号进行及时处理的需求。

### 16.3.1 过零检测事件

每个零检测事件的触发条件如下表所示：

事件	触发条件
SIGNAL_RISING_EDGE	上升沿
SIGNAL_FALLING_EDGE	下降沿
SIGNAL_VALID	频率在有效范围内次数大于设置阈值
SIGNAL_INVALID	频率在无效范围内次数大于设置阈值
SIGNAL_LOST	在 100ms 内信号没有上升沿、下降沿
SIGNAL_FREQ_OUT_OF_RANGE	计算的频率在设置的频率范围之外

**注意：** 回调函数中不能有 TaskDelay 等阻塞的操作

### 16.3.2 配置项

- USE\_GPTIMER：决定是否使用 GPTimer 驱动默认使用 ESPTimer

### 16.3.3 应用示例

#### 创建过零检测

```
void IRAM_ATTR zero_detection_event_cb(zero_detect_event_t zero_detect_event, zero_
↪detect_cb_param_t *param, void *usr_data) //User's callback API
{
    switch (zero_detect_event) {
        case SIGNAL_FREQ_OUT_OF_RANGE:
            ESP_LOGE(TAG, "SIGNAL_FREQ_OUT_OF_RANGE");
            break;
        case SIGNAL_VALID:
            ESP_LOGE(TAG, "SIGNAL_VALID");
            break;
        case SIGNAL_LOST:
            ESP_LOGE(TAG, "SIGNAL_LOST");
            break;
        default:
            break;
    }
}

// Create a zero detection and register call-back
zero_detect_config_t config = {
    .capture_pin = 2,
    .freq_range_max_hz = 65,
    .freq_range_min_hz = 45, //Hz
    .valid_times = 6,
    .invalid_times = 5,
    .zero_signal_type = SQUARE_WAVE,
    .zero_driver_type = MCPWM_TYPE,
};
zero_detect_handle_t *g_zcds = zero_detect_create(&config);
if(NULL == g_zcds) {
    ESP_LOGE(TAG, "Zero Detection create failed");
}
zero_detect_register_cb(g_zcds, zero_detection_event_cb, NULL);
```

### 16.3.4 API Reference

#### Header File

- [components/zero\\_detection/include/zero\\_detection.h](#)

#### Functions

**[zero\\_detect\\_handle\\_t zero\\_detect\\_create](#)** ([zero\\_detect\\_config\\_t](#) \*config)

Create a zero detect target.

**参数 config** –A zero detect object to config

**返回**

- [zero\\_detect\\_handle\\_t](#) A zero cross detection object

**void [zero\\_show\\_data](#)** ([zero\\_detect\\_handle\\_t](#) zcd\_handle)

Show zero detect test data.

**参数 zcd\_handle** –A zero detect handle



esp\_err\_t **zero\_detect\_delete** ([zero\\_detect\\_handle\\_t](#) zcd\_handle)

Delete a zero detect device.

**参数** **zcd\_handle** –A zero detect handle

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

bool **zero\_detect\_get\_power\_status** ([zero\\_detect\\_handle\\_t](#) zcd\_handle)

Get relay power status.

**参数** **zcd\_handle** –A zero detect handle

**返回**

- true Power on
- false Power off

bool **zero\_detect\_singal\_invaild\_status** ([zero\\_detect\\_handle\\_t](#) zcd\_handle)

Get signal invalid status.

**参数** **zcd\_handle** –A zero detect handle

**返回**

- true Signal is invalid
- false Signal is valid

[zero\\_signal\\_type\\_t](#) **zero\_detect\_get\_signal\_type** ([zero\\_detect\\_handle\\_t](#) zcd\_handle)

Get signal type.

**参数** **zcd\_handle** –A zero detect handle

**返回**

- SQUARE\_WAVE Signal type is square
- PULSE\_WAVE Signal type is pulse

esp\_err\_t **zero\_detect\_register\_cb** ([zero\\_detect\\_handle\\_t](#) zcd\_handle, [zero\\_cross\\_cb\\_t](#) cb, void \*usr\_data)

Register zero cross detection callback.

**参数**

- **zcd\_handle** –A zero detect handle
- **cb** –A callback function
- **usr\_data** –User data

**返回**

- ESP\_OK Success
- ESP\_FAIL Failure

## Unions

union **zero\_detect\_cb\_param\_t**

#include <zero\_detection.h> Event callback parameters union.

## Public Members

struct [zero\\_detect\\_cb\\_param\\_t::signal\\_freq\\_event\\_data\\_t](#) **signal\_freq\_event\_data**

Signal freq event data struct

struct [zero\\_detect\\_cb\\_param\\_t::signal\\_valid\\_event\\_data\\_t](#) **signal\_valid\_event\_data**

Signal valid event data struct

```
struct zero\_detect\_cb\_param\_t::signal\_invalid\_event\_data\_t signal_invalid_event_data
```

Signal invalid event data struct

```
struct zero\_detect\_cb\_param\_t::signal\_rising\_edge\_event\_data\_t signal_rising_edge_event_data
```

Signal rising edge event data struct

```
struct zero\_detect\_cb\_param\_t::signal\_falling\_edge\_event\_data\_t
```

```
signal_falling_edge_event_data
```

Signal falling edge event\_data

```
struct signal_falling_edge_event_data_t
```

*#include <zero\_detection.h>* Signal falling edge data return type.

### Public Members

```
uint16_t valid_count
```

Counting when the signal is valid

```
uint16_t invalid_count
```

Counting when the signal is invalid

```
uint32_t full_cycle_us
```

Current signal cycle

```
struct signal_freq_event_data_t
```

*#include <zero\_detection.h>* Signal exceeds frequency range data return type.

### Public Members

```
mcpwm_capture_edge_t cap_edge
```

Trigger edge of zero cross signal

```
uint32_t full_cycle_us
```

Current signal cycle

```
struct signal_invalid_event_data_t
```

*#include <zero\_detection.h>* Signal invalid data return type.

### Public Members

```
mcpwm_capture_edge_t cap_edge
```

Trigger edge of zero cross signal

```
uint32_t full_cycle_us
```

Current signal cycle

uint16\_t **invalid\_count**

Counting when the signal is invalid

struct **signal\_rising\_edge\_event\_data\_t**

*#include <zero\_detection.h>* Signal rising edge data return type.

### Public Members

uint16\_t **valid\_count**

Counting when the signal is valid

uint16\_t **invalid\_count**

Counting when the signal is invalid

uint32\_t **full\_cycle\_us**

Current signal cycle

struct **signal\_valid\_event\_data\_t**

*#include <zero\_detection.h>* Signal valid data return type.

### Public Members

mcpwm\_capture\_edge\_t **cap\_edge**

Trigger edge of zero cross signal

uint32\_t **full\_cycle\_us**

Current signal cycle

uint16\_t **valid\_count**

Counting when the signal is valid

## Structures

struct **zero\_detect\_config\_t**

User config data type.

### Public Members

int32\_t **capture\_pin**

GPIO number for zero cross detect capture

uint16\_t **valid\_times**

Minimum required number of times for detecting signal validity

uint16\_t **invalid\_times**

Minimum required number of times for detecting signal invalidity

uint64\_t **signal\_lost\_time\_us**

Minimum required duration for detecting signal loss

*zero\_signal\_type\_t* **zero\_signal\_type**

Zero Crossing Signal type

*zero\_driver\_type\_t* **zero\_driver\_type**

Zero crossing driver type

double **freq\_range\_max\_hz**

Maximum value of the frequency range when determining a valid signal

double **freq\_range\_min\_hz**

Minimum value of the frequency range when determining a valid signal

## Macros

**ZERO\_DETECTION\_INIT\_CONFIG\_DEFAULT** ( )

## Type Definitions

typedef void (\***zero\_cross\_cb\_t**)(*zero\_detect\_event\_t* zero\_detect\_event, *zero\_detect\_cb\_param\_t* \*param, void \*usr\_data)

Callback format.

typedef void \***zero\_detect\_handle\_t**

## Enumerations

enum **zero\_detect\_event\_t**

Zero detection events.

*Values:*

enumerator **SIGNAL\_FREQ\_OUT\_OF\_RANGE**

enumerator **SIGNAL\_VALID**

enumerator **SIGNAL\_INVALID**

enumerator **SIGNAL\_LOST**

enumerator **SIGNAL\_RISING\_EDGE**

enumerator **SIGNAL\_FALLING\_EDGE**

enum **zero\_signal\_type\_t**

Zero detection wave type.

*Values:*

enumerator **SQUARE\_WAVE**

enumerator **PULSE\_WAVE**

enum **zero\_driver\_type\_t**

Zero detection driver type.

*Values:*

enumerator **GPIO\_TYPE**

Use GPIO as driver

## Chapter 17

# Contributions Guide

We welcome contributions to the esp-iot-solution project!

### 17.1 How to Contribute

Contributions to esp-iot-solution - fixing bugs, adding features, adding documentation - are welcome. We accept contributions via [Github Pull Requests](#).

### 17.2 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.
- Does any new code conform to the esp-idf : [Style Guide](#) ?
- Have you installed the [pre-commit hook](#) for the esp-iot-solution project?
- Does the code documentation follow requirements in [Documenting-code](#) ?
- Is the code adequately commented for people to understand how it is structured?
- Are comments and documentation written in clear English, with no spelling or grammar errors?
- 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.

### 17.3 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.

### 17.4 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.

## 17.5 Related Documents

### 17.5.1 esp-iot-solution 编码规范

#### 总体原则

- 简洁明了，结构清晰
- 统一风格，易于维护
- 充分注释，易于理解
- [继承 ESP-IDF 已有规范](#)
- 继承第三方代码已有规范

#### 目录结构

- components：总体上按照功能分类，大类下如果存在多级目录，应包含一个 README 做综述和索引；
- docs：rst 格式文档，包括各个组件的使用指南，API 说明；
- examples：总体上按照与组件对应的功能分类；
- tools：CI 脚本、调试工具。

#### 头文件

- 尽量每一 .c 对应一个同名的 .h 文件；
- 单个组件存在多个 .h，主要对外 .h 命名尽量与组件名保持一致；
- 头文件中主要放函数声明，不放函数实现；
- 尽量不在头文件定义任何形式的变量；
- 头文件应按照注释规范，对函数接口进行充分注释；
- 添加宏定义避免重复引用，宏定义为大写的头文件名加下划线填充：

```
#ifndef _IOT_I2C_BUS_H_
#define _IOT_I2C_BUS_H_

#endif
```

- 函数声明添加 extern “C” 修饰，支持 C/C++ 混合编程：

```
#ifdef __cplusplus
extern "C"
{
#endif

//c code

#ifdef __cplusplus
}
#endif
```

#### 注释

- 安装 VSCODE 插件 [Doxygen Documentation Generator](#) 可自动生成注释框架；
- 注释中避免使用单词缩写；
- 函数声明处注释需要描述函数功能、性能或用法，输入和输出参数、函数返回值说明。

自动生成的注释框架：

```
/**
 * @brief
 *
 * @param port
 * @param conf
 * @return i2c_bus_handle_t
 */
i2c_bus_handle_t iot_i2c_bus_create(i2c_port_t port, i2c_config_t* conf);
```

补充信息和参数方向：

```
/**
 * @brief Create an I2C bus instance then return a handle if created successfully.
 * @note Each I2C bus works in a singleton mode, which means for an i2c port only.
 * ↳ one group parameter works. When
 * iot_i2c_bus_create is called more than one time for the same i2c port, following
 * ↳ parameter will override the previous one.
 *
 * @param[in] port I2C port number
 * @param[in] conf Pointer to I2C parameters
 * @return i2c_bus_handle_t Return the I2C bus handle if created successfully,
 * ↳ return NULL if failed.
 */
i2c_bus_handle_t iot_i2c_bus_create(i2c_port_t port, i2c_config_t* conf);
```

- 版权声明注释（第三方代码，请保留版权声明信息）

```
/*
 * SPDX-FileCopyrightText: 2022-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: Apache-2.0
 */
```

## 函数规范

- 多处重复使用的代码尽量设计为函数；
- 作用域仅限于当前文件的函数必须声明为静态 static；
- 设计使用静态全局变量、静态局部变量的函数时，需要考虑重入问题；
- 尽量在一个固定函数中操作静态全局变量；
- 如果函数存在重入或线程安全问题，需在注释中说明；
- 同一组件内的公有函数名，应保持同一前缀；
- 函数名统一使用 snake\_case 格式，只使用小写字母，单词之间加 \_；
- 函数命名指引（应保持与已有代码风格一致，不严格约束）：

函数名格式	函数示例	说明
iot_type_xxx	iot_sensor_xxx;      iot_board_xxx; iot_storage_...	高度抽象的 iot 组件
type_xxx	imu_xxx; light_xxx; eeprom_xxx	对一类外设的抽象
name_xxx	mpu6050_xxx;	底层 driver，由于可能来自第三方，不约束函数名
xxx_creat xxx_delete	/	创建和销毁
xxx_read xxx_write	/	数据操作



## 变量规范

- 避免使用全局变量，可声明为静态全局变量，使用 `get_ set_` 等接口进行变量操作；
- 作用域仅限于当前文件的变量必须声明为静态变量 `static`；
- 静态全局变量请添加 `g_` 前缀，静态局部变量请添加 `s_` 前缀；
- 局部变量设计大小时，应考虑栈溢出的问题；
- 任何变量定义时，必须赋初值；
- 变量功能要明确，避免将单一变量做多个用途；
- 句柄类型变量，在对象销毁后，应重新赋值为 `NULL`；
- 变量统一使用 `snake_case` 格式，只使用小写字母，单词之间加 `_`；
- 避免不必要的缩写，例如 `data` 不必缩写为 `dat`；
- 变量应尽量使用有意义的词语，或者已经达成一致的符号或词语缩写；
- 变量命名指引：

类型	规范	示例
全局变量	避免使用	<code>x</code>
静态全局变量	<code>static</code> 标识， <code>g_</code> 前缀，赋初值	<code>static uint32_t g_connect_num = 0;</code>
静态局部变量	<code>static</code> 标识， <code>s_</code> 前缀，赋初值	<code>static uint32_t s_connect_num = 0;</code>
迭代计数变量	使用通用的 <code>i j k</code>	
常用缩写	<a href="#">abbreviations-in-code</a>	<code>addr, buf, cfg, cmd, , ctrl,</code>

- 常用缩写列表

缩写	全称	缩写	全称	缩写	全称	缩写	全称
<code>addr</code>	address	<code>id</code>	identifier	<code>len</code>	length	<code>ptr</code>	pointer
<code>buf</code>	buffer	<code>info</code>	information	<code>obj</code>	object	<code>ret</code>	return
<code>cfg</code>	config	<code>hdr</code>	header	<code>param</code>	parameter	<code>temp</code>	temporary、temperature
<code>cmd</code>	command	<code>init</code>	initialize	<code>pos</code>	position	<code>ts</code>	timestamp

## 类型定义

- 使用加 `snake_case` 格式加 `_t` 后缀

```
typedef int signed_32_bit_t;
```

- 枚举应通过 `typedef` 通过以下方式定义

```
typedef enum {
    MODULE_FOO_ONE,
    MODULE_FOO_TWO,
    MODULE_FOO_THREE
} module_foo_t;
```

## 格式和排版规范

该部分继承 [ESP-IDF 规范](#)

- 1. 缩进** 每个缩进层使用 **4 个空格**，不要使用制表符进行缩进，将编辑器配置为每次按 `tab` 键时发出 4 个空格。
- 2. 垂直间隔** 在函数之间放置一个空行，不要以空行开始或结束函数。

```

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);
    }
}

```

只要不严重影响可读性，最大行长度为 120 个字符。

### 3. 水平间隔 总是在条件和循环关键字之后添加单个空格

```

if (condition) {    // correct
    // ...
}

switch (n) {        // correct
    case 0:
        // ...
}

for(int i = 0; i < CONST; ++i) {    // INCORRECT
    // ...
}

```

在二元操作符两端添加单个空格，一元运算符不需要空格，可以在乘法运算符和除法运算符之间省略空格。

```

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

```

. 和 -> 操作符的周围不需要任何空格。

有时，在一行中添加水平间隔有助于提高代码的可读性。如下，可以添加空格来对齐函数参数：

```

gpio_matrix_in(PIN_CAM_D6,    I2S0I_DATA_IN14_IDX, false);
gpio_matrix_in(PIN_CAM_D7,    I2S0I_DATA_IN15_IDX, false);
gpio_matrix_in(PIN_CAM_HREF,  I2S0I_H_ENABLE_IDX,  false);
gpio_matrix_in(PIN_CAM_PCLK,  I2S0I_DATA_IN15_IDX, false);

```

- 但是请注意，如果有人添加了一个新行，第一个参数是一个更长的标识符（例如 PIN\_CAM\_VSYNC），它将不适合。因为必须重新对齐其他行，这添加了无意义的更改。因此，尽量少使用这种对齐，特别是如果您希望稍后将新行添加到这列中。
- 不要使用制表符进行水平对齐，不要在行尾添加尾随空格。

### 4. 括号 函数定义的大括号应该在单独的行上

```
// This is correct:
void function(int arg)
{

}

// NOT like this:
void function(int arg) {

}
```

在函数中，将左大括号与条件语句和循环语句放在同一行

```
if (condition) {
    do_one();
} else if (other_condition) {
    do_two();
}
```

**5. 注释** // 用于单行注释。对于多行注释，可以在每行上使用 // 或 / \* \* / 块注释。

虽然与格式没有直接关系，但下面是一些关于有效使用注释的注意事项。

- 不要使用一个注释来禁用某些功能

```
void init_something()
{
    setup_dma();
    // load_resources(); // WHY is this thing commented, asks the
↪ reader?
    start_timer();
}
```

- 如果不再需要某些代码，则将其完全删除。如果你需要，你可以随时在 `git` 历史中查找这个文件。如果您因为临时原因而禁用某些调用，并打算在将来恢复它，则在相邻行上添加解释

```
void init_something()
{
    setup_dma();
    // TODO: we should load resources here, but loader is not fully integrated yet.
    // load_resources();
    start_timer();
}
```

- `#if 0 ... #endif` 块也是如此。如果不使用，请完全删除代码块。否则，添加注释以解释为什么禁用该块。不要使用 `#if 0 ... #endif` 或注释来存储将来可能需要的代码段。
- 不要添加有关作者和更改日期的琐碎注释。您总是可以查找谁使用 `git` 修改了任何给定的行。例如，此注释在不添加任何有用信息的情况下，使代码混乱不堪：

```
void init_something()
{
    setup_dma();
    // XXX add 2016-09-01
    init_dma_list();
    fill_dma_item(0);
    // end XXX add
    start_timer();
}
```

**6. 代码行的结束** `commit` 中只能包含以 LF（Unix 风格）结尾的文件。

Windows 用户可以将 `git` 配置为在本地 `checkout` 是 CRLF (Windows 风格) 结尾, 通过设置 `core.autocrlf` 设置来 `commit` 时以 LF 结尾。Github 有一个关于设置此选项的文档。但是, 由于 MSYS2 使用 Unix 样式的行尾, 因此在编辑 ESP-IDF 源文件时, 通常更容易将文本编辑器配置为使用 LF (Unix 样式) 结尾。

如果您在分支中意外地 `commit` 了 LF 结尾, 则可以通过在 MSYS2 或 Unix 终端中运行此命令将它们转换为 Unix (将目录更改为 IDF 工作目录, 并预先检查当前是否已 `checkout` 正确的分支):

```
git rebase --exec 'git diff-tree --no-commit-id --name-only -r HEAD | xargs ↵
↵dos2unix && git commit -a --amend --no-edit --allow-empty' master
```

(请注意, 这行代码将在 `master` 上重新建立基, 并在最后更改分支名称以在另一个分支上建立基。)

要更新单个提交, 可以运行

```
dos2unix FILENAME
```

然后运行

```
git commit --amend
```

**7. 格式化代码** 您可以使用 `astyle` 程序根据上述建议对代码进行格式化。

如果您正在从头开始编写一个文件, 或者正在进行完全重写, 请随意重新格式化整个文件。如果您正在更改文件的一小部分, 不要重新格式化您没有更改的代码。这将帮助其他人检查您的更改。

要重新格式化文件, 请运行

```
tools/format.sh components/my_component/file.c
```

---

## CMake 代码风格

- 缩进是 4 个空格
- 最大行长为 120 个字符。分割行时, 请尝试尽可能集中于可读性 (例如, 通过在单独的行上配对关键字/参数对)。
- 不要在 `foreach()`、`endif()` 等后面的可选括号中放入任何内容。
- 对命令、函数和宏名使用小写 ( `with_underscores` )。
- 对于局部作用域的变量, 使用小写字母 ( `with_underscores` )。
- 对于全局作用域的变量, 使用大写 ( `WITH_UNDERSCORES` )。
- 其他, 请遵循 [cmake-lint](#) 项目的默认设置。



# 索引

## 符号

[anonymous] (C++ *enum*), 138  
[anonymous]::LED\_STATE\_25\_PERCENT  
(C++ *enumerator*), 138  
[anonymous]::LED\_STATE\_50\_PERCENT  
(C++ *enumerator*), 138  
[anonymous]::LED\_STATE\_75\_PERCENT  
(C++ *enumerator*), 138  
[anonymous]::LED\_STATE\_OFF (C++ *enumera-*  
*tor*), 138  
[anonymous]::LED\_STATE\_ON (C++ *enumera-*  
*tor*), 138

## A

ALARM\_COUNT\_US (C *macro*), 317  
ALIGNMENTDUTY (C *macro*), 318  
ALIGNMENTNMS (C *macro*), 318  
AVOID\_CONTINUE\_CURRENT\_TIME (C *macro*),  
319

## B

BAS\_CHR\_BATTERY\_INFO\_FEATURE\_BM\_NONE  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FEATURE\_BM\_RECHARGE  
(C *macro*), 51  
BAS\_CHR\_BATTERY\_INFO\_FEATURE\_BM\_REPLACE  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FEATURE\_BM\_RFU (C  
*macro*), 51  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_AGGREGATION\_GROUP  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_CHEMISTRY  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_CRITICAL\_ENERGY  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_DESIGNED\_CAPACITY  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_EXPIRATION\_DATE  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_LOW\_ENERGY  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_MANUFACTURE\_DATE  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_NOMINAL\_VOLTAGE  
(C *macro*), 50  
BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_NONE (C  
*macro*), 50

BAS\_CHR\_BATTERY\_INFO\_FLAGS\_BM\_RFU (C  
*macro*), 50  
BAS\_CHR\_CRITICAL\_STATUS\_FLAGS\_BM\_CRITICAL\_POWER\_S  
(C *macro*), 49  
BAS\_CHR\_CRITICAL\_STATUS\_FLAGS\_BM\_IMMEDIATE\_SERVIC  
(C *macro*), 49  
BAS\_CHR\_CRITICAL\_STATUS\_FLAGS\_BM\_NONE  
(C *macro*), 49  
BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_AVAILALBE\_BATTERY\_  
(C *macro*), 49  
BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_AVAILALBE\_ENERGY  
(C *macro*), 49  
BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_AVAILALBE\_ENERGY\_L  
(C *macro*), 49  
BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_CHARGE\_RATE  
(C *macro*), 49  
BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_EXTERNAL\_SOURCE\_PO  
(C *macro*), 49  
BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_NONE (C  
*macro*), 49  
BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_RFU (C  
*macro*), 49  
BAS\_CHR\_ENERGY\_STATUS\_FLAGS\_BM\_VOLTAGE  
(C *macro*), 49  
BAS\_CHR\_HEALTH\_INFO\_FLAGS\_BM\_CYCLE\_COUNT\_DESIGNED  
(C *macro*), 50  
BAS\_CHR\_HEALTH\_INFO\_FLAGS\_BM\_DESIGNED\_OPERATING\_T  
(C *macro*), 50  
BAS\_CHR\_HEALTH\_INFO\_FLAGS\_BM\_NONE (C  
*macro*), 50  
BAS\_CHR\_HEALTH\_INFO\_FLAGS\_BM\_RFU (C  
*macro*), 50  
BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_BATTERY\_HEALTH\_SUM  
(C *macro*), 50  
BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_CURRENT\_TEMPERATUR  
(C *macro*), 50  
BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_DEEP\_DISCHARGE\_COU  
(C *macro*), 50  
BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_NONE (C  
*macro*), 50  
BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_RCYCLE\_COUNT  
(C *macro*), 50  
BAS\_CHR\_HEALTH\_STATUS\_FLAGS\_BM\_RFU (C  
*macro*), 50  
BAS\_CHR\_LEVEL\_STATUS\_ASSITIONAL\_STATUS\_BATTERY\_FA  
(C *macro*), 49  
BAS\_CHR\_LEVEL\_STATUS\_ASSITIONAL\_STATUS\_BATTERY\_FA



- bldc\_control\_handle\_t (C++ type), 316  
 bldc\_control\_init (C++ function), 314  
 bldc\_control\_set\_dir (C++ function), 314  
 bldc\_control\_set\_duty (C++ function), 315  
 bldc\_control\_set\_speed\_rpm (C++ function), 315  
 bldc\_control\_start (C++ function), 314  
 bldc\_control\_stop (C++ function), 314  
 bldc\_debug\_config\_t (C++ struct), 315  
 bldc\_debug\_config\_t::debug\_operation (C++ member), 315  
 bldc\_debug\_config\_t::if\_debug (C++ member), 315  
 BLDC\_LEDC (C macro), 317  
 BLDC\_MCPWM (C macro), 317  
 BLE\_ANP\_CAT\_BM\_CALL (C macro), 64  
 BLE\_ANP\_CAT\_BM\_EMAIL (C macro), 64  
 BLE\_ANP\_CAT\_BM\_MISSED\_CALL (C macro), 64  
 BLE\_ANP\_CAT\_BM\_NEWS (C macro), 64  
 BLE\_ANP\_CAT\_BM\_NONE (C macro), 64  
 BLE\_ANP\_CAT\_BM\_SCHEDULE (C macro), 64  
 BLE\_ANP\_CAT\_BM\_SIMPLE\_ALERT (C macro), 64  
 BLE\_ANP\_CAT\_BM\_SMS (C macro), 64  
 BLE\_ANP\_CAT\_BM\_VOICE\_MAIL (C macro), 64  
 BLE\_ANP\_CAT\_ID\_CALL (C macro), 65  
 BLE\_ANP\_CAT\_ID\_EMAIL (C macro), 65  
 BLE\_ANP\_CAT\_ID\_MISSED\_CALL (C macro), 65  
 BLE\_ANP\_CAT\_ID\_NEWS (C macro), 65  
 BLE\_ANP\_CAT\_ID\_SCHEDULE (C macro), 65  
 BLE\_ANP\_CAT\_ID\_SIMPLE\_ALERT (C macro), 64  
 BLE\_ANP\_CAT\_ID\_SMS (C macro), 65  
 BLE\_ANP\_CAT\_ID\_VOICE\_MAIL (C macro), 65  
 BLE\_ANP\_CAT\_NUM (C macro), 65  
 BLE\_ANP\_CHR\_UUID16\_ALERT\_NOT\_CTRL\_PT (C macro), 64  
 BLE\_ANP\_CHR\_UUID16\_NEW\_ALERT (C macro), 64  
 BLE\_ANP\_CHR\_UUID16\_SUP\_NEW\_ALERT\_CAT (C macro), 64  
 BLE\_ANP\_CHR\_UUID16\_SUP\_UNR\_ALERT\_CAT (C macro), 64  
 BLE\_ANP\_CHR\_UUID16\_UNR\_ALERT\_STAT (C macro), 64  
 BLE\_ANP\_CMD\_DIS\_NEW\_ALERT\_CAT (C macro), 65  
 BLE\_ANP\_CMD\_DIS\_UNR\_ALERT\_CAT (C macro), 65  
 BLE\_ANP\_CMD\_EN\_NEW\_ALERT\_CAT (C macro), 65  
 BLE\_ANP\_CMD\_EN\_UNR\_ALERT\_CAT (C macro), 65  
 BLE\_ANP\_CMD\_NOT\_NEW\_ALERT\_IMMEDIATE (C macro), 65  
 BLE\_ANP\_CMD\_NOT\_UNR\_ALERT\_IMMEDIATE (C macro), 65  
 BLE\_ANP\_INFO\_STR\_MAX\_LEN (C macro), 65  
 BLE\_ANP\_NEW\_ALERT\_MAX\_LEN (C macro), 65  
 BLE\_ANP\_UUID16 (C macro), 64  
 BLE\_ANS\_CAT\_BM\_CALL (C macro), 36  
 BLE\_ANS\_CAT\_BM\_EMAIL (C macro), 36  
 BLE\_ANS\_CAT\_BM\_MISSED\_CALL (C macro), 36  
 BLE\_ANS\_CAT\_BM\_NEWS (C macro), 36  
 BLE\_ANS\_CAT\_BM\_NONE (C macro), 36  
 BLE\_ANS\_CAT\_BM\_SCHEDULE (C macro), 36  
 BLE\_ANS\_CAT\_BM\_SIMPLE\_ALERT (C macro), 36  
 BLE\_ANS\_CAT\_BM\_SMS (C macro), 36  
 BLE\_ANS\_CAT\_BM\_VOICE\_MAIL (C macro), 36  
 BLE\_ANS\_CAT\_ID\_CALL (C macro), 36  
 BLE\_ANS\_CAT\_ID\_EMAIL (C macro), 36  
 BLE\_ANS\_CAT\_ID\_MISSED\_CALL (C macro), 36  
 BLE\_ANS\_CAT\_ID\_NEWS (C macro), 36  
 BLE\_ANS\_CAT\_ID\_SCHEDULE (C macro), 37  
 BLE\_ANS\_CAT\_ID\_SIMPLE\_ALERT (C macro), 36  
 BLE\_ANS\_CAT\_ID\_SMS (C macro), 37  
 BLE\_ANS\_CAT\_ID\_VOICE\_MAIL (C macro), 37  
 BLE\_ANS\_CAT\_NUM (C macro), 37  
 BLE\_ANS\_CHR\_UUID16\_ALERT\_NOT\_CTRL\_PT (C macro), 36  
 BLE\_ANS\_CHR\_UUID16\_NEW\_ALERT (C macro), 36  
 BLE\_ANS\_CHR\_UUID16\_SUP\_NEW\_ALERT\_CAT (C macro), 36  
 BLE\_ANS\_CHR\_UUID16\_SUP\_UNR\_ALERT\_CAT (C macro), 36  
 BLE\_ANS\_CHR\_UUID16\_UNR\_ALERT\_STAT (C macro), 36  
 BLE\_ANS\_CMD\_DIS\_NEW\_ALERT\_CAT (C macro), 37  
 BLE\_ANS\_CMD\_DIS\_UNR\_ALERT\_CAT (C macro), 37  
 BLE\_ANS\_CMD\_EN\_NEW\_ALERT\_CAT (C macro), 37  
 BLE\_ANS\_CMD\_EN\_UNR\_ALERT\_CAT (C macro), 37  
 BLE\_ANS\_CMD\_NOT\_NEW\_ALERT\_IMMEDIATE (C macro), 37  
 BLE\_ANS\_CMD\_NOT\_UNR\_ALERT\_IMMEDIATE (C macro), 37  
 BLE\_ANS\_INFO\_STR\_MAX\_LEN (C macro), 37  
 BLE\_ANS\_NEW\_ALERT\_MAX\_LEN (C macro), 37  
 BLE\_ANS\_UUID16 (C macro), 36  
 BLE\_BAS\_CHR\_UUID16\_BATTERY\_INFO (C macro), 47  
 BLE\_BAS\_CHR\_UUID16\_CRITICAL\_STATUS (C macro), 47  
 BLE\_BAS\_CHR\_UUID16\_ENERGY\_STATUS (C macro), 47  
 BLE\_BAS\_CHR\_UUID16\_ESTIMATED\_SERVICE\_DATE (C macro), 47  
 BLE\_BAS\_CHR\_UUID16\_HEALTH\_INFO (C macro), 47  
 BLE\_BAS\_CHR\_UUID16\_HEALTH\_STATUS (C macro), 47  
 BLE\_BAS\_CHR\_UUID16\_LEVEL (C macro), 47  
 BLE\_BAS\_CHR\_UUID16\_LEVEL\_STATUS (C macro), 47



- BLE\_BAS\_CHR\_UUID16\_TIME\_STATUS (C macro), 47
- BLE\_BAS\_UUID16 (C macro), 47
- BLE\_CONN\_GATT\_CHR\_BROADCAST (C macro), 31
- BLE\_CONN\_GATT\_CHR\_INDICATE (C macro), 32
- BLE\_CONN\_GATT\_CHR\_NOTIFY (C macro), 31
- BLE\_CONN\_GATT\_CHR\_READ (C macro), 31
- BLE\_CONN\_GATT\_CHR\_WRITE (C macro), 31
- BLE\_CONN\_GATT\_CHR\_WRITE\_NO\_RSP (C macro), 31
- BLE\_DIS\_CHR\_UUID16\_FIRMWARE\_REVISION (C macro), 54
- BLE\_DIS\_CHR\_UUID16\_HARDWARE\_REVISION (C macro), 54
- BLE\_DIS\_CHR\_UUID16\_MANUFACTURER\_NAME (C macro), 54
- BLE\_DIS\_CHR\_UUID16\_MODEL\_NUMBER (C macro), 54
- BLE\_DIS\_CHR\_UUID16\_PNP\_ID (C macro), 55
- BLE\_DIS\_CHR\_UUID16\_REG\_CERT (C macro), 55
- BLE\_DIS\_CHR\_UUID16\_SERIAL\_NUMBER (C macro), 54
- BLE\_DIS\_CHR\_UUID16\_SOFTWARE\_REVISION (C macro), 54
- BLE\_DIS\_CHR\_UUID16\_SYSTEM\_ID (C macro), 54
- BLE\_DIS\_UUID16 (C macro), 54
- ble\_hci\_add\_to\_accept\_list (C++ function), 73
- BLE\_HCI\_ADDR\_LEN (C macro), 76
- ble\_hci\_addr\_t (C++ type), 76
- ble\_hci\_addr\_type\_t (C++ enum), 77
- ble\_hci\_addr\_type\_t::BLE\_ADDR\_TYPE\_PUBLIC (C++ enumerator), 77
- ble\_hci\_addr\_type\_t::BLE\_ADDR\_TYPE\_RANDOM (C++ enumerator), 77
- ble\_hci\_addr\_type\_t::BLE\_ADDR\_TYPE\_RPA\_PUBLIC (C++ enumerator), 77
- ble\_hci\_addr\_type\_t::BLE\_ADDR\_TYPE\_RPA\_RANDOM (C++ enumerator), 77
- ble\_hci\_adv\_channel\_t (C++ enum), 77
- ble\_hci\_adv\_channel\_t::ADV\_CHNL\_37 (C++ enumerator), 77
- ble\_hci\_adv\_channel\_t::ADV\_CHNL\_38 (C++ enumerator), 78
- ble\_hci\_adv\_channel\_t::ADV\_CHNL\_39 (C++ enumerator), 78
- ble\_hci\_adv\_channel\_t::ADV\_CHNL\_ALL (C++ enumerator), 78
- ble\_hci\_adv\_filter\_t (C++ enum), 78
- ble\_hci\_adv\_filter\_t::ADV\_FILTER\_ALLOW\_SCAN\_ANY\_CONN (C++ enumerator), 78
- ble\_hci\_adv\_filter\_t::ADV\_FILTER\_ALLOW\_SCAN\_ANY\_CONN\_WHEN\_IDLE (C++ enumerator), 78
- ble\_hci\_adv\_filter\_t::ADV\_FILTER\_ALLOW\_SCAN\_WHILE\_SCANNING (C++ enumerator), 78
- ble\_hci\_adv\_filter\_t::ADV\_FILTER\_ALLOW\_SCAN\_WHILE\_SCANNING\_WHEN\_IDLE (C++ enumerator), 78
- ble\_hci\_adv\_param\_t (C++ struct), 74
- ble\_hci\_adv\_param\_t::adv\_filter\_policy (C++ member), 75
- ble\_hci\_adv\_param\_t::adv\_int\_max (C++ member), 75
- ble\_hci\_adv\_param\_t::adv\_int\_min (C++ member), 75
- ble\_hci\_adv\_param\_t::adv\_type (C++ member), 75
- ble\_hci\_adv\_param\_t::channel\_map (C++ member), 75
- ble\_hci\_adv\_param\_t::own\_addr\_type (C++ member), 75
- ble\_hci\_adv\_param\_t::peer\_addr (C++ member), 75
- ble\_hci\_adv\_param\_t::peer\_addr\_type (C++ member), 75
- ble\_hci\_adv\_type\_t (C++ enum), 77
- ble\_hci\_adv\_type\_t::ADV\_TYPE\_DIRECT\_IND\_HIGH (C++ enumerator), 77
- ble\_hci\_adv\_type\_t::ADV\_TYPE\_DIRECT\_IND\_LOW (C++ enumerator), 77
- ble\_hci\_adv\_type\_t::ADV\_TYPE\_IND (C++ enumerator), 77
- ble\_hci\_adv\_type\_t::ADV\_TYPE\_NONCONN\_IND (C++ enumerator), 77
- ble\_hci\_adv\_type\_t::ADV\_TYPE\_SCAN\_IND (C++ enumerator), 77
- ble\_hci\_clear\_accept\_list (C++ function), 74
- ble\_hci\_deinit (C++ function), 72
- ble\_hci\_dev\_type\_t (C++ enum), 77
- ble\_hci\_dev\_type\_t::BLE\_HCI\_DEVICE\_TYPE\_BLE (C++ enumerator), 77
- ble\_hci\_dev\_type\_t::BLE\_HCI\_DEVICE\_TYPE\_BREDR (C++ enumerator), 77
- ble\_hci\_dev\_type\_t::BLE\_HCI\_DEVICE\_TYPE\_DUMO (C++ enumerator), 77
- ble\_hci\_enable\_meta\_event (C++ function), 72
- ble\_hci\_init (C++ function), 72
- ble\_hci\_reset (C++ function), 72
- ble\_hci\_scan\_cb\_t (C++ type), 76
- ble\_hci\_scan\_param\_t (C++ struct), 75
- ble\_hci\_scan\_param\_t::filter\_policy (C++ member), 75
- ble\_hci\_scan\_param\_t::own\_addr\_type (C++ member), 75
- ble\_hci\_scan\_param\_t::scan\_interval (C++ member), 75
- ble\_hci\_scan\_param\_t::scan\_type (C++ member), 75
- ble\_hci\_scan\_param\_t::scan\_window (C++ member), 75
- ble\_hci\_scan\_result\_t (C++ struct), 74
- ble\_hci\_scan\_result\_t::adv\_data\_len (C++ member), 74
- ble\_hci\_scan\_result\_t::bda (C++ member), 74

---

Espressif Systems **359** Release master

BLE_HTP_CHR_TEMPERATURE_TYPE_MAX (C macro), 71	BLE_HTS_CHR_TEMPERATURE_UNITS_CELSIUS (C macro), 60
BLE_HTP_CHR_TEMPERATURE_TYPE_MOUTH (C macro), 71	BLE_HTS_CHR_TEMPERATURE_UNITS_FAHRENHEIT (C macro), 60
BLE_HTP_CHR_TEMPERATURE_TYPE_RECTUM (C macro), 71	BLE_HTS_CHR_UUID16_INTERMEDIATE_TEMPERATURE (C macro), 60
BLE_HTP_CHR_TEMPERATURE_TYPE_RFU (C macro), 71	BLE_HTS_CHR_UUID16_MEASUREMENT_INTERVAL (C macro), 60
BLE_HTP_CHR_TEMPERATURE_TYPE_TOE (C macro), 71	BLE_HTS_CHR_UUID16_TEMPERATURE_MEASUREMENT (C macro), 60
BLE_HTP_CHR_TEMPERATURE_TYPE_TYMPANUM (C macro), 71	BLE_HTS_CHR_UUID16_TEMPERATURE_TYPE (C macro), 60
BLE_HTP_CHR_TEMPERATURE_UNITS_CELSIUS (C macro), 71	BLE_HTS_UUID16 (C macro), 60
BLE_HTP_CHR_TEMPERATURE_UNITS_FAHRENHEIT (C macro), 71	BLE_TPS_CHR_UUID16_TX_POWER_LEVEL (C macro), 61
BLE_HTP_CHR_UUID16_INTERMEDIATE_TEMPERATURE (C macro), 71	BLE_TPS_UUID16 (C macro), 61
BLE_HTP_CHR_UUID16_MEASUREMENT_INTERVAL (C macro), 71	BLE_UUID128_VAL_LEN (C macro), 31
BLE_HTP_CHR_UUID16_TEMPERATURE_MEASUREMENT (C macro), 70	BLE_UUID_CMP (C macro), 32
BLE_HTP_CHR_UUID16_TEMPERATURE_TYPE (C macro), 71	BLE_UUID_TYPE (C macro), 32
BLE_HTP_FLAGS_BM_NONE (C macro), 71	blink_step_t (C++ struct), 137
BLE_HTP_FLAGS_BM_RFU (C macro), 71	blink_step_t::hold_time_ms (C++ member), 137
BLE_HTP_FLAGS_BM_TEMPERATURE_TYPE (C macro), 71	blink_step_t::type (C++ member), 137
BLE_HTP_FLAGS_BM_TEMPERATURE_UNITS (C macro), 71	blink_step_t::value (C++ member), 137
BLE_HTP_FLAGS_BM_TIME_STAMP (C macro), 71	blink_step_type_t (C++ enum), 138
BLE_HTP_UUID16 (C macro), 70	blink_step_type_t::LED_BLINK_BREATHE (C++ enumerator), 139
BLE_HTS_CHR_TEMPERATURE_FLAGS_NOT (C macro), 60	blink_step_type_t::LED_BLINK_BRIGHTNESS (C++ enumerator), 139
BLE_HTS_CHR_TEMPERATURE_FLAGS_SET (C macro), 60	blink_step_type_t::LED_BLINK_HOLD (C++ enumerator), 138
BLE_HTS_CHR_TEMPERATURE_TYPE_ARMPIT (C macro), 60	blink_step_type_t::LED_BLINK_HSV (C++ enumerator), 139
BLE_HTS_CHR_TEMPERATURE_TYPE_BODY (C macro), 60	blink_step_type_t::LED_BLINK_HSV_RING (C++ enumerator), 139
BLE_HTS_CHR_TEMPERATURE_TYPE_EAR (C macro), 60	blink_step_type_t::LED_BLINK_LOOP (C++ enumerator), 139
BLE_HTS_CHR_TEMPERATURE_TYPE_FINGER (C macro), 60	blink_step_type_t::LED_BLINK_RGB (C++ enumerator), 139
BLE_HTS_CHR_TEMPERATURE_TYPE_GAST_TRAC (C macro), 60	blink_step_type_t::LED_BLINK_RGB_RING (C++ enumerator), 139
BLE_HTS_CHR_TEMPERATURE_TYPE_MAX (C macro), 61	blink_step_type_t::LED_BLINK_STOP (C++ enumerator), 138
BLE_HTS_CHR_TEMPERATURE_TYPE_MOUTH (C macro), 61	BROADCAST_PARAM_LEN (C macro), 31
BLE_HTS_CHR_TEMPERATURE_TYPE_RECTUM (C macro), 61	bus_handle_t (C++ type), 264
BLE_HTS_CHR_TEMPERATURE_TYPE_RFU (C macro), 60	button_cb_t (C++ type), 232
BLE_HTS_CHR_TEMPERATURE_TYPE_TOE (C macro), 61	button_config_t (C++ struct), 232
BLE_HTS_CHR_TEMPERATURE_TYPE_TYMPANUM (C macro), 61	button_config_t::custom_button_config (C++ member), 232
	button_config_t::gpio_button_config (C++ member), 232
	button_config_t::long_press_time (C++ member), 232
	button_config_t::matrix_button_config (C++ member), 232
	button_config_t::short_press_time (C++ member), 232
	button_config_t::type (C++ member), 232

- button\_config\_t::[anonymous] (C++ member), 232
  - button\_custom\_config\_t (C++ struct), 231
  - button\_custom\_config\_t::active\_level (C++ member), 231
  - button\_custom\_config\_t::button\_custom\_deinit (C++ member), 232
  - button\_custom\_config\_t::button\_custom\_get\_key (C++ member), 232
  - button\_custom\_config\_t::button\_custom\_init (C++ member), 231
  - button\_custom\_config\_t::priv (C++ member), 232
  - button\_event\_config\_t (C++ struct), 231
  - button\_event\_config\_t::event (C++ member), 231
  - button\_event\_config\_t::event\_data (C++ member), 231
  - button\_event\_data\_t (C++ union), 230
  - button\_event\_data\_t::long\_press (C++ member), 231
  - button\_event\_data\_t::long\_press\_t (C++ struct), 231
  - button\_event\_data\_t::long\_press\_t::press\_time (C++ member), 231
  - button\_event\_data\_t::multiple\_clicks (C++ member), 231
  - button\_event\_data\_t::multiple\_clicks\_t (C++ struct), 231
  - button\_event\_data\_t::multiple\_clicks\_t::clicks (C++ member), 231
  - button\_event\_t (C++ enum), 233
  - button\_event\_t::BUTTON\_DOUBLE\_CLICK (C++ enumerator), 233
  - button\_event\_t::BUTTON\_EVENT\_MAX (C++ enumerator), 233
  - button\_event\_t::BUTTON\_LONG\_PRESS\_HOLD (C++ enumerator), 233
  - button\_event\_t::BUTTON\_LONG\_PRESS\_START (C++ enumerator), 233
  - button\_event\_t::BUTTON\_LONG\_PRESS\_UP (C++ enumerator), 233
  - button\_event\_t::BUTTON\_MULTIPLE\_CLICK (C++ enumerator), 233
  - button\_event\_t::BUTTON\_NONE\_PRESS (C++ enumerator), 233
  - button\_event\_t::BUTTON\_PRESS\_DOWN (C++ enumerator), 233
  - button\_event\_t::BUTTON\_PRESS\_END (C++ enumerator), 233
  - button\_event\_t::BUTTON\_PRESS\_REPEAT (C++ enumerator), 233
  - button\_event\_t::BUTTON\_PRESS\_REPEAT\_DONE (C++ enumerator), 233
  - button\_event\_t::BUTTON\_PRESS\_UP (C++ enumerator), 233
  - button\_event\_t::BUTTON\_SINGLE\_CLICK (C++ enumerator), 233
  - button\_handle\_t (C++ type), 232
  - button\_param\_t (C++ enum), 233
  - button\_param\_t::BUTTON\_LONG\_PRESS\_TIME\_MS (C++ enumerator), 234
  - button\_param\_t::BUTTON\_PARAM\_MAX (C++ enumerator), 234
  - button\_param\_t::BUTTON\_SHORT\_PRESS\_TIME\_MS (C++ enumerator), 234
  - button\_param\_t::get\_key (C++ enumerator), 234
  - button\_type\_t (C++ enum), 233
  - button\_type\_t::BUTTON\_TYPE\_ADC (C++ enumerator), 233
  - button\_type\_t::BUTTON\_TYPE\_CUSTOM (C++ enumerator), 233
  - button\_type\_t::BUTTON\_TYPE\_GPIO (C++ enumerator), 233
  - button\_type\_t::BUTTON\_TYPE\_MATRIX (C++ enumerator), 233
- ## C
- CDC\_INTERFACE\_NUM\_MAX (C macro), 186
  - CHARGE\_TIME (C macro), 318
  - control\_mode\_t (C++ enum), 316
  - control\_mode\_t::BLDC\_FOC (C++ enumerator), 316
  - control\_mode\_t::BLDC\_SIX\_STEP (C++ enumerator), 316
- ## D
- dac\_audio\_config\_t (C++ struct), 199
  - dac\_audio\_config\_t::bits\_per\_sample (C++ member), 199
  - dac\_audio\_config\_t::dac\_mode (C++ member), 199
  - dac\_audio\_config\_t::dma\_buf\_count (C++ member), 199
  - dac\_audio\_config\_t::dma\_buf\_len (C++ member), 200
  - dac\_audio\_config\_t::i2s\_num (C++ member), 199
  - dac\_audio\_config\_t::max\_data\_size (C++ member), 200
  - dac\_audio\_config\_t::sample\_rate (C++ member), 199
  - dac\_audio\_deinit (C++ function), 198
  - dac\_audio\_init (C++ function), 198
  - dac\_audio\_set\_param (C++ function), 198
  - dac\_audio\_set\_volume (C++ function), 199
  - dac\_audio\_start (C++ function), 198
  - dac\_audio\_stop (C++ function), 198
  - dac\_audio\_write (C++ function), 199
  - DEFAULTS\_PROX\_CONFIGS (C macro), 294
  - DUTY\_MAX (C macro), 317
  - DUTY\_RES (C macro), 317
- ## E
- ENTER\_CLOSE\_TIME (C macro), 319
  - ESP\_BLE\_ADV\_DATA\_LEN\_MAX (C macro), 76
  - esp\_ble\_anp\_data\_t (C++ struct), 63

esp\_ble\_anp\_data\_t::cat\_id (C++ member), 63  
 esp\_ble\_anp\_data\_t::cat\_info (C++ member), 64  
 esp\_ble\_anp\_data\_t::count (C++ member), 64  
 esp\_ble\_anp\_data\_t::new\_alert\_val (C++ member), 64  
 esp\_ble\_anp\_data\_t::unr\_alert\_stat (C++ member), 64  
 esp\_ble\_anp\_data\_t::[anonymous] (C++ member), 64  
 esp\_ble\_anp\_deinit (C++ function), 63  
 esp\_ble\_anp\_get\_new\_alert (C++ function), 62  
 esp\_ble\_anp\_get\_unr\_alert (C++ function), 62  
 esp\_ble\_anp\_init (C++ function), 63  
 esp\_ble\_anp\_option\_t (C++ enum), 65  
 esp\_ble\_anp\_option\_t::BLE\_ANP\_OPT\_DISABLE (C++ enumerator), 65  
 esp\_ble\_anp\_option\_t::BLE\_ANP\_OPT\_ENABLE (C++ enumerator), 65  
 esp\_ble\_anp\_option\_t::BLE\_ANP\_OPT\_RECOVER (C++ enumerator), 65  
 esp\_ble\_anp\_set\_new\_alert (C++ function), 62  
 esp\_ble\_anp\_set\_unr\_alert (C++ function), 63  
 esp\_ble\_ans\_get\_new\_alert (C++ function), 35  
 esp\_ble\_ans\_get\_unread\_alert (C++ function), 35  
 esp\_ble\_ans\_init (C++ function), 35  
 esp\_ble\_ans\_set\_new\_alert (C++ function), 35  
 esp\_ble\_ans\_set\_unread\_alert (C++ function), 35  
 esp\_ble\_bas\_battery\_info\_t (C++ struct), 45  
 esp\_ble\_bas\_battery\_info\_t::aggregation\_group (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::chemistry (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::critical\_energy (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::designed\_capacity (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::en\_aggregation\_group (C++ member), 47  
 esp\_ble\_bas\_battery\_info\_t::en\_chemistry (C++ member), 47  
 esp\_ble\_bas\_battery\_info\_t::en\_critical\_energy (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::en\_designed\_capacity (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::en\_expiration\_date (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::en\_low\_energy (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::en\_manufacture\_date (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::en\_nominalvoltage (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::expiration\_date (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::features (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::flags (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::flags\_reserved (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::low\_energy (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::manufacture\_date (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::nominalvoltage (C++ member), 46  
 esp\_ble\_bas\_battery\_info\_t::recharge\_able (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::replace\_able (C++ member), 45  
 esp\_ble\_bas\_battery\_info\_t::reserved (C++ member), 45  
 esp\_ble\_bas\_critical\_status\_t (C++ struct), 41  
 esp\_ble\_bas\_critical\_status\_t::critical\_power\_status (C++ member), 41  
 esp\_ble\_bas\_critical\_status\_t::immediate\_service (C++ member), 41  
 esp\_ble\_bas\_critical\_status\_t::reserved (C++ member), 42  
 esp\_ble\_bas\_critical\_status\_t::status (C++ member), 42  
 esp\_ble\_bas\_data\_t (C++ struct), 46  
 esp\_ble\_bas\_data\_t::battery\_info (C++ member), 47  
 esp\_ble\_bas\_data\_t::battery\_level (C++ member), 46  
 esp\_ble\_bas\_data\_t::critical\_status (C++ member), 46  
 esp\_ble\_bas\_data\_t::energy\_status (C++ member), 46  
 esp\_ble\_bas\_data\_t::estimated\_service\_date (C++ member), 47  
 esp\_ble\_bas\_data\_t::health\_info (C++ member), 47  
 esp\_ble\_bas\_data\_t::health\_status (C++ member), 47  
 esp\_ble\_bas\_data\_t::level\_status (C++ member), 46  
 esp\_ble\_bas\_data\_t::time\_status (C++ member), 46  
 esp\_ble\_bas\_energy\_status\_t (C++ struct), 42  
 esp\_ble\_bas\_energy\_status\_t::available\_battery\_capacity (C++ member), 42



esp\_ble\_bas\_energy\_status\_t::available\_energy (C++ member), 44  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::available\_energy (C++ member), 43  
 (C++ member), 43  
 esp\_ble\_bas\_energy\_status\_t::charge\_rate (C++ member), 44  
 (C++ member), 43  
 esp\_ble\_bas\_energy\_status\_t::en\_available\_battery (C++ member), 44  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::en\_available\_energy (C++ member), 44  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::en\_available\_energy (C++ member), 44  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::en\_charge\_rate (C++ member), 43  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::en\_external\_source (C++ member), 43  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::en\_voltage (C++ member), 43  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::external\_source (C++ member), 44  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::flags (C++ member), 44  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::reserved (C++ member), 44  
 (C++ member), 42  
 esp\_ble\_bas\_energy\_status\_t::voltage (C++ member), 44  
 (C++ member), 42  
 esp\_ble\_bas\_get\_battery\_info (C++ function), 39  
 esp\_ble\_bas\_get\_battery\_level (C++ function), 37  
 esp\_ble\_bas\_get\_critical\_status (C++ function), 38  
 esp\_ble\_bas\_get\_energy\_status (C++ function), 38  
 esp\_ble\_bas\_get\_estimated\_date (C++ function), 38  
 esp\_ble\_bas\_get\_health\_info (C++ function), 39  
 esp\_ble\_bas\_get\_health\_status (C++ function), 39  
 esp\_ble\_bas\_get\_level\_status (C++ function), 38  
 esp\_ble\_bas\_get\_time\_status (C++ function), 39  
 esp\_ble\_bas\_health\_info\_t (C++ struct), 44  
 esp\_ble\_bas\_health\_info\_t::cycle\_count (C++ member), 44  
 esp\_ble\_bas\_health\_info\_t::en\_cycle\_count (C++ member), 44  
 esp\_ble\_bas\_health\_info\_t::flags (C++ member), 44  
 esp\_ble\_bas\_health\_info\_t::max\_designed\_operating\_temperature (C++ member), 44  
 esp\_ble\_bas\_health\_info\_t::min\_designed\_operating\_temperature (C++ member), 44  
 esp\_ble\_bas\_health\_info\_t::min\_max\_designed\_operating\_temperature (C++ member), 44  
 esp\_ble\_bas\_health\_info\_t::reserved (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t (C++ struct), 44  
 esp\_ble\_bas\_health\_status\_t::battery\_health\_summary (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::battery\_health\_summary (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::current\_temperature (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::cycle\_count (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::deep\_discharge\_count (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::en\_battery\_health\_summary (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::en\_current\_temperature (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::en\_cycle\_count (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::en\_deep\_discharge\_count (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::flags (C++ member), 44  
 esp\_ble\_bas\_health\_status\_t::reserved (C++ member), 44  
 esp\_ble\_bas\_init (C++ function), 40  
 esp\_ble\_bas\_level\_status\_t (C++ struct), 40  
 esp\_ble\_bas\_level\_status\_t::additional\_status (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::battery (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::battery\_charge\_level (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::battery\_charge\_state (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::battery\_charge\_type (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::battery\_fault (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::battery\_level (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::charging\_fault\_reason (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::en\_additional\_status (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::en\_battery\_level (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::en\_battery\_level (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::en\_identifier (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::en\_level\_status (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::flags (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::flags\_reserved (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::identifier (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::power\_state (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::power\_state\_reserved (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::reserved (C++ member), 41

(C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::service\_required (C++ member), 29  
 (C++ member), 41  
 esp\_ble\_bas\_level\_status\_t::wired\_external\_power (C++ member), 29  
 (C++ member), 40  
 esp\_ble\_bas\_level\_status\_t::wireless\_external (C++ member), 29  
 (C++ member), 40  
 esp\_ble\_bas\_set\_battery\_info (C++ function), 39  
 esp\_ble\_bas\_set\_battery\_level (C++ function), 37  
 esp\_ble\_bas\_set\_critical\_status (C++ function), 38  
 esp\_ble\_bas\_set\_energy\_status (C++ function), 38  
 esp\_ble\_bas\_set\_estimated\_date (C++ function), 38  
 esp\_ble\_bas\_set\_health\_info (C++ function), 39  
 esp\_ble\_bas\_set\_health\_status (C++ function), 39  
 esp\_ble\_bas\_set\_level\_status (C++ function), 38  
 esp\_ble\_bas\_set\_time\_status (C++ function), 39  
 esp\_ble\_bas\_time\_status\_t (C++ struct), 43  
 esp\_ble\_bas\_time\_status\_t::discharged (C++ member), 43  
 esp\_ble\_bas\_time\_status\_t::discharged\_standby (C++ member), 43  
 esp\_ble\_bas\_time\_status\_t::en\_discharged\_standby (C++ member), 43  
 esp\_ble\_bas\_time\_status\_t::en\_recharged (C++ member), 43  
 esp\_ble\_bas\_time\_status\_t::flags (C++ member), 43  
 esp\_ble\_bas\_time\_status\_t::recharged (C++ member), 43  
 esp\_ble\_bas\_time\_status\_t::reserved (C++ member), 43  
 esp\_ble\_conn\_add\_svc (C++ function), 27  
 esp\_ble\_conn\_cb\_t (C++ type), 32  
 esp\_ble\_conn\_character\_t (C++ struct), 28  
 esp\_ble\_conn\_character\_t::flag (C++ member), 28  
 esp\_ble\_conn\_character\_t::name (C++ member), 28  
 esp\_ble\_conn\_character\_t::type (C++ member), 28  
 esp\_ble\_conn\_character\_t::uuid (C++ member), 28  
 esp\_ble\_conn\_character\_t::uuid\_fn (C++ member), 28  
 esp\_ble\_conn\_config\_t (C++ struct), 29  
 esp\_ble\_conn\_config\_t::broadcast\_data (C++ member), 29  
 esp\_ble\_conn\_config\_t::device\_name (C++ member), 29  
 esp\_ble\_conn\_config\_t::extended\_adv\_data (C++ member), 29  
 esp\_ble\_conn\_config\_t::extended\_adv\_len (C++ member), 29  
 esp\_ble\_conn\_config\_t::extended\_adv\_rsp\_data (C++ member), 29  
 esp\_ble\_conn\_config\_t::extended\_adv\_rsp\_len (C++ member), 29  
 esp\_ble\_conn\_config\_t::include\_service\_uuid (C++ member), 29  
 esp\_ble\_conn\_config\_t::periodic\_adv\_data (C++ member), 29  
 esp\_ble\_conn\_config\_t::periodic\_adv\_len (C++ member), 29  
 esp\_ble\_conn\_connect (C++ function), 26  
 esp\_ble\_conn\_data\_t (C++ struct), 29  
 esp\_ble\_conn\_data\_t::data (C++ member), 30  
 esp\_ble\_conn\_data\_t::data\_len (C++ member), 30  
 esp\_ble\_conn\_data\_t::type (C++ member), 29  
 esp\_ble\_conn\_data\_t::uuid (C++ member), 29  
 esp\_ble\_conn\_data\_t::write\_conn\_id (C++ member), 29  
 esp\_ble\_conn\_deinit (C++ function), 26  
 esp\_ble\_conn\_desc\_t (C++ enum), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_AGG\_FORMAT (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_CIENT\_CONF (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_COMPLETE\_B (C++ enumerator), 34  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_DIGITAL (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_ENV\_SENS\_C (C++ enumerator), 34  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_ENV\_SENS\_M (C++ enumerator), 34  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_ENV\_TRIGGER (C++ enumerator), 34  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_EXTENDED (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_EXTREPORT (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_PRE\_FORMAT (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_REPORT (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_SERVER\_CON (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_TIME\_TRIGGER (C++ enumerator), 34  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_UNKNOWN (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_USER (C++ enumerator), 33

esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_ADV\_IDLE\_RANGE (C++ enumerator), 33  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_ADV\_IDLE\_RANGE periodic\_sync\_t::sid (C++ member), 30  
 esp\_ble\_conn\_desc\_t::ESP\_BLE\_CONN\_DESC\_ADV\_IDLE\_RANGE periodic\_sync\_t::status (C++ member), 30  
 esp\_ble\_conn\_disconnect (C++ function), 26  
 esp\_ble\_conn\_event\_t (C++ enum), 32  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED read (C++ function), 27  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED (C++ enumerator), 32  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED esp\_ble\_conn\_remove\_svc (C++ function), 27  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED esp\_ble\_conn\_set\_mtu (C++ function), 26  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED (C++ enumerator), 32  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED esp\_ble\_conn\_start (C++ function), 26  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED esp\_ble\_conn\_stop (C++ function), 26  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED (C++ enumerator), 33  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_CONNECTED esp\_ble\_conn\_subscribe (C++ function), 27  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED svc\_t (C++ struct), 28  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 32  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED esp\_ble\_conn\_svc\_t::nu\_lookup (C++ member), 28  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED esp\_ble\_conn\_svc\_t::nu\_lookup\_count (C++ member), 28  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 33  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED esp\_ble\_conn\_svc\_t::type (C++ member), 28  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 33  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED esp\_ble\_conn\_sync\_host (C++ member), 28  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 33  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED esp\_ble\_conn\_uuid\_t (C++ union), 27  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 32  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED esp\_ble\_conn\_uuid\_t::uuid128 (C++ member), 28  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 32  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED esp\_ble\_conn\_uuid\_t::uuid16 (C++ member), 27  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 32  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED esp\_ble\_conn\_uuid\_t::uuid32 (C++ member), 28  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 32  
 esp\_ble\_conn\_event\_t::ESP\_BLE\_CONN\_EVENT\_DISCONNECTED (C++ enumerator), 32  
 esp\_ble\_conn\_init (C++ function), 25  
 esp\_ble\_conn\_notify (C++ function), 26  
 esp\_ble\_conn\_periodic\_report\_t (C++ struct), 30  
 esp\_ble\_conn\_periodic\_report\_t::data (C++ member), 31  
 esp\_ble\_conn\_periodic\_report\_t::data\_length (C++ member), 31  
 esp\_ble\_conn\_periodic\_report\_t::data\_status (C++ member), 31  
 esp\_ble\_conn\_periodic\_report\_t::data\_status esp\_ble\_dis\_data (C++ struct), 54  
 esp\_ble\_conn\_periodic\_report\_t::rssi (C++ member), 30  
 esp\_ble\_conn\_periodic\_report\_t::rssi (C++ member), 54  
 esp\_ble\_conn\_periodic\_report\_t::sync\_handle (C++ member), 30  
 esp\_ble\_conn\_periodic\_report\_t::sync\_handle (C++ member), 54  
 esp\_ble\_conn\_periodic\_report\_t::tx\_power (C++ member), 30  
 esp\_ble\_conn\_periodic\_report\_t::tx\_power (C++ member), 54  
 esp\_ble\_conn\_periodic\_report\_t::sync\_lost\_t (C++ struct), 31  
 esp\_ble\_conn\_periodic\_report\_t::sync\_lost\_t (C++ member), 54  
 esp\_ble\_conn\_periodic\_report\_t::sync\_lost\_t::reason (C++ member), 31  
 esp\_ble\_conn\_periodic\_report\_t::sync\_lost\_t::reason (C++ member), 54  
 esp\_ble\_conn\_periodic\_report\_t::sync\_lost\_t::sync\_handle (C++ member), 31  
 esp\_ble\_conn\_periodic\_report\_t::sync\_lost\_t::sync\_handle (C++ member), 54  
 esp\_ble\_conn\_periodic\_sync\_t (C++ struct), 30  
 esp\_ble\_conn\_periodic\_sync\_t::adv\_addr (C++ member), 30  
 esp\_ble\_conn\_periodic\_sync\_t::adv\_addr esp\_ble\_dis\_data\_t (C++ type), 55  
 esp\_ble\_conn\_periodic\_sync\_t::adv\_clk\_accuracy (C++ member), 30  
 esp\_ble\_conn\_periodic\_sync\_t::adv\_clk\_accuracy (C++ member), 51  
 esp\_ble\_conn\_periodic\_sync\_t::adv\_phy (C++ member), 30  
 esp\_ble\_conn\_periodic\_sync\_t::adv\_phy (C++ member), 52  
 esp\_ble\_conn\_periodic\_sync\_t::per\_adv\_ival (C++ member), 30  
 esp\_ble\_conn\_periodic\_sync\_t::per\_adv\_ival (C++ member), 52  
 esp\_ble\_conn\_periodic\_sync\_t::per\_adv\_ival (C++ member), 30  
 esp\_ble\_conn\_periodic\_sync\_t::per\_adv\_ival (C++ member), 52



- tion*), 51
- `esp_ble_dis_get_pnp_id` (C++ *function*), 53
- `esp_ble_dis_get_serial_number` (C++ *function*), 51
- `esp_ble_dis_get_software_revision` (C++ *function*), 52
- `esp_ble_dis_get_system_id` (C++ *function*), 52
- `esp_ble_dis_init` (C++ *function*), 53
- `esp_ble_dis_pnp` (C++ *struct*), 53
- `esp_ble_dis_pnp::pid` (C++ *member*), 53
- `esp_ble_dis_pnp::src` (C++ *member*), 53
- `esp_ble_dis_pnp::ver` (C++ *member*), 53
- `esp_ble_dis_pnp::vid` (C++ *member*), 53
- `esp_ble_dis_pnp_t` (C++ *type*), 55
- `esp_ble_dis_set_firmware_revision` (C++ *function*), 52
- `esp_ble_dis_set_hardware_revision` (C++ *function*), 52
- `esp_ble_dis_set_manufacturer_name` (C++ *function*), 52
- `esp_ble_dis_set_model_number` (C++ *function*), 51
- `esp_ble_dis_set_pnp_id` (C++ *function*), 53
- `esp_ble_dis_set_serial_number` (C++ *function*), 51
- `esp_ble_dis_set_software_revision` (C++ *function*), 52
- `esp_ble_dis_set_system_id` (C++ *function*), 53
- `esp_ble_hrp_data_t` (C++ *struct*), 67
- `esp_ble_hrp_data_t::detected` (C++ *member*), 67
- `esp_ble_hrp_data_t::energy` (C++ *member*), 67
- `esp_ble_hrp_data_t::energy_val` (C++ *member*), 67
- `esp_ble_hrp_data_t::flags` (C++ *member*), 67
- `esp_ble_hrp_data_t::format` (C++ *member*), 67
- `esp_ble_hrp_data_t::heartrate` (C++ *member*), 67
- `esp_ble_hrp_data_t::interval` (C++ *member*), 67
- `esp_ble_hrp_data_t::interval_buf` (C++ *member*), 67
- `esp_ble_hrp_data_t::reserved` (C++ *member*), 67
- `esp_ble_hrp_data_t::supported` (C++ *member*), 67
- `esp_ble_hrp_data_t::u16` (C++ *member*), 67
- `esp_ble_hrp_data_t::u8` (C++ *member*), 67
- `esp_ble_hrp_deinit` (C++ *function*), 66
- `esp_ble_hrp_get_ctrl` (C++ *function*), 66
- `esp_ble_hrp_get_location` (C++ *function*), 66
- `esp_ble_hrp_init` (C++ *function*), 66
- `esp_ble_hrp_set_ctrl` (C++ *function*), 66
- `esp_ble_hrs_get_hrm` (C++ *function*), 55
- `esp_ble_hrs_get_location` (C++ *function*), 55
- `esp_ble_hrs_hrm_t` (C++ *struct*), 56
- `esp_ble_hrs_hrm_t::detected` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::energy` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::energy_val` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::flags` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::format` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::heartrate` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::interval` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::interval_buf` (C++ *member*), 57
- `esp_ble_hrs_hrm_t::reserved` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::supported` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::u16` (C++ *member*), 56
- `esp_ble_hrs_hrm_t::u8` (C++ *member*), 56
- `esp_ble_hrs_init` (C++ *function*), 56
- `esp_ble_hrs_set_hrm` (C++ *function*), 55
- `esp_ble_hrs_set_location` (C++ *function*), 55
- `esp_ble_http_data_t` (C++ *struct*), 69
- `esp_ble_http_data_t::celsius` (C++ *member*), 70
- `esp_ble_http_data_t::day` (C++ *member*), 70
- `esp_ble_http_data_t::fahrenheit` (C++ *member*), 70
- `esp_ble_http_data_t::flags` (C++ *member*), 70
- `esp_ble_http_data_t::hours` (C++ *member*), 70
- `esp_ble_http_data_t::location` (C++ *member*), 70
- `esp_ble_http_data_t::minutes` (C++ *member*), 70
- `esp_ble_http_data_t::month` (C++ *member*), 70
- `esp_ble_http_data_t::reserved` (C++ *member*), 70
- `esp_ble_http_data_t::seconds` (C++ *member*), 70
- `esp_ble_http_data_t::temperature` (C++ *member*), 70
- `esp_ble_http_data_t::temperature_type` (C++ *member*), 70
- `esp_ble_http_data_t::temperature_unit` (C++ *member*), 69
- `esp_ble_http_data_t::time_stamp` (C++ *member*), 69
- `esp_ble_http_data_t::timestamp` (C++ *member*), 70
- `esp_ble_http_data_t::year` (C++ *member*), 70

- esp\_ble\_htp\_deinit (C++ function), 69
- esp\_ble\_htp\_get\_measurement\_interval (C++ function), 69
- esp\_ble\_htp\_get\_temp\_type (C++ function), 69
- esp\_ble\_htp\_init (C++ function), 69
- esp\_ble\_htp\_set\_measurement\_interval (C++ function), 69
- esp\_ble\_hst\_get\_intermediate\_temp (C++ function), 58
- esp\_ble\_hst\_get\_measurement\_interval (C++ function), 58
- esp\_ble\_hst\_get\_measurement\_temp (C++ function), 58
- esp\_ble\_hst\_get\_temp\_type (C++ function), 58
- esp\_ble\_hst\_init (C++ function), 59
- esp\_ble\_hst\_set\_intermediate\_temp (C++ function), 58
- esp\_ble\_hst\_set\_measurement\_interval (C++ function), 58
- esp\_ble\_hst\_set\_measurement\_temp (C++ function), 58
- esp\_ble\_hst\_set\_temp\_type (C++ function), 58
- esp\_ble\_hst\_temp\_t (C++ struct), 59
- esp\_ble\_hst\_temp\_t::celsius (C++ member), 59
- esp\_ble\_hst\_temp\_t::day (C++ member), 59
- esp\_ble\_hst\_temp\_t::fahrenheit (C++ member), 59
- esp\_ble\_hst\_temp\_t::flags (C++ member), 59
- esp\_ble\_hst\_temp\_t::hours (C++ member), 60
- esp\_ble\_hst\_temp\_t::location (C++ member), 60
- esp\_ble\_hst\_temp\_t::minutes (C++ member), 60
- esp\_ble\_hst\_temp\_t::month (C++ member), 59
- esp\_ble\_hst\_temp\_t::reserved (C++ member), 59
- esp\_ble\_hst\_temp\_t::seconds (C++ member), 60
- esp\_ble\_hst\_temp\_t::temperature (C++ member), 59
- esp\_ble\_hst\_temp\_t::temperature\_type (C++ member), 59
- esp\_ble\_hst\_temp\_t::temperature\_unit (C++ member), 59
- esp\_ble\_hst\_temp\_t::time\_stamp (C++ member), 59
- esp\_ble\_hst\_temp\_t::timestamp (C++ member), 60
- esp\_ble\_hst\_temp\_t::year (C++ member), 59
- ESP\_BLE\_SCAN\_RSP\_DATA\_LEN\_MAX (C macro), 76
- esp\_ble\_tps\_get\_tx\_power\_level (C++ function), 61
- esp\_ble\_tps\_init (C++ function), 61
- esp\_ble\_tps\_set\_tx\_power\_level (C++ function), 61
- ESP\_EVENT\_DECLARE\_BASE (C++ function), 314
- esp\_lv\_split\_png\_deinit (C++ function), 140
- esp\_lv\_split\_png\_init (C++ function), 140
- esp\_lv\_spng\_decoder\_handle\_t (C++ type), 141
- esp\_msc\_ota (C++ function), 180
- esp\_msc\_ota\_abort (C++ function), 179
- esp\_msc\_ota\_begin (C++ function), 179
- esp\_msc\_ota\_config\_t (C++ struct), 180
- esp\_msc\_ota\_config\_t::buffer\_size (C++ member), 181
- esp\_msc\_ota\_config\_t::bulk\_flash\_erase (C++ member), 181
- esp\_msc\_ota\_config\_t::ota\_bin\_path (C++ member), 180
- esp\_msc\_ota\_config\_t::wait\_msc\_connect (C++ member), 181
- esp\_msc\_ota\_end (C++ function), 179
- esp\_msc\_ota\_event\_t (C++ enum), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_ABORT (C++ enumerator), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_FAILED (C++ enumerator), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_FINISH (C++ enumerator), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_GET\_IMG\_DESC (C++ enumerator), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_READY\_UPDATE (C++ enumerator), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_START (C++ enumerator), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_UPDATE\_BOOT\_PART (C++ enumerator), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_VERIFY\_CHIP\_ID (C++ enumerator), 181
- esp\_msc\_ota\_event\_t::ESP\_MSC\_OTA\_WRITE\_FLASH (C++ enumerator), 181
- esp\_msc\_ota\_get\_img\_desc (C++ function), 180
- esp\_msc\_ota\_get\_status (C++ function), 180
- esp\_msc\_ota\_handle\_t (C++ type), 181
- esp\_msc\_ota\_is\_complete\_data\_received (C++ function), 180
- esp\_msc\_ota\_perform (C++ function), 179
- esp\_msc\_ota\_set\_msc\_connect\_state (C++ function), 180
- esp\_msc\_ota\_status\_t (C++ enum), 182
- esp\_msc\_ota\_status\_t::ESP\_MSC\_OTA\_BEGIN (C++ enumerator), 182
- esp\_msc\_ota\_status\_t::ESP\_MSC\_OTA\_IN\_PROGRESS (C++ enumerator), 182
- esp\_msc\_ota\_status\_t::ESP\_MSC\_OTA\_INIT (C++ enumerator), 182

- esp\_msc\_ota\_status\_t::ESP\_MSC\_OTA\_SUCCESS  
(C++ enumerator), 182
- ## F
- FLAG\_UAC\_MIC\_SUSPEND\_AFTER\_START (C  
macro), 176
- FLAG\_UAC\_SPK\_SUSPEND\_AFTER\_START (C  
macro), 176
- FLAG\_UVC\_SUSPEND\_AFTER\_START (C macro),  
176
- FPS2INTERVAL (C macro), 175
- FRAME\_INTERVAL\_FPS\_10 (C macro), 176
- FRAME\_INTERVAL\_FPS\_15 (C macro), 176
- FRAME\_INTERVAL\_FPS\_20 (C macro), 176
- FRAME\_INTERVAL\_FPS\_30 (C macro), 176
- FRAME\_INTERVAL\_FPS\_5 (C macro), 175
- FRAME\_RESOLUTION\_ANY (C macro), 175
- FREQ\_HZ (C macro), 317
- ## G
- gpio\_pad\_select\_gpio (C macro), 14
- ## H
- humiture\_acquire (C++ function), 273
- humiture\_acquire\_humidity (C++ function),  
272
- humiture\_acquire\_temperature (C++ func-  
tion), 272
- humiture\_control (C++ function), 273
- humiture\_create (C++ function), 272
- humiture\_delete (C++ function), 272
- humiture\_id\_t (C++ enum), 273
- humiture\_id\_t::HTS221\_ID (C++ enumerator),  
273
- humiture\_id\_t::HUMITURE\_MAX\_ID (C++  
enumerator), 273
- humiture\_id\_t::SHT3X\_ID (C++ enumerator),  
273
- humiture\_sleep (C++ function), 272
- humiture\_test (C++ function), 272
- humiture\_wakeup (C++ function), 272
- ## I
- i2c\_bus\_cmd\_begin (C++ function), 13
- i2c\_bus\_create (C++ function), 10
- i2c\_bus\_delete (C++ function), 10
- i2c\_bus\_device\_create (C++ function), 11
- i2c\_bus\_device\_delete (C++ function), 11
- i2c\_bus\_device\_get\_address (C++ function),  
11
- i2c\_bus\_device\_handle\_t (C++ type), 15
- i2c\_bus\_get\_created\_device\_num (C++  
function), 11
- i2c\_bus\_get\_current\_clk\_speed (C++ func-  
tion), 10
- i2c\_bus\_handle\_t (C++ type), 15
- i2c\_bus\_read\_bit (C++ function), 12
- i2c\_bus\_read\_bits (C++ function), 12
- i2c\_bus\_read\_byte (C++ function), 11
- i2c\_bus\_read\_bytes (C++ function), 11
- i2c\_bus\_read\_reg16 (C++ function), 14
- i2c\_bus\_scan (C++ function), 10
- i2c\_bus\_write\_bit (C++ function), 13
- i2c\_bus\_write\_bits (C++ function), 13
- i2c\_bus\_write\_byte (C++ function), 12
- i2c\_bus\_write\_bytes (C++ function), 13
- i2c\_bus\_write\_reg16 (C++ function), 14
- i2s\_lcd\_acquire (C++ function), 19
- i2s\_lcd\_config\_t (C++ struct), 19
- i2s\_lcd\_config\_t::buffer\_size (C++  
member), 19
- i2s\_lcd\_config\_t::clk\_freq (C++ member),  
19
- i2s\_lcd\_config\_t::data\_width (C++ mem-  
ber), 19
- i2s\_lcd\_config\_t::i2s\_port (C++ member),  
19
- i2s\_lcd\_config\_t::pin\_data\_num (C++  
member), 19
- i2s\_lcd\_config\_t::pin\_num\_cs (C++ mem-  
ber), 19
- i2s\_lcd\_config\_t::pin\_num\_rs (C++ mem-  
ber), 19
- i2s\_lcd\_config\_t::pin\_num\_wr (C++ mem-  
ber), 19
- i2s\_lcd\_config\_t::swap\_data (C++ mem-  
ber), 19
- i2s\_lcd\_driver\_deinit (C++ function), 18
- i2s\_lcd\_driver\_init (C++ function), 18
- i2s\_lcd\_handle\_t (C++ type), 20
- i2s\_lcd\_release (C++ function), 19
- i2s\_lcd\_write (C++ function), 18
- i2s\_lcd\_write\_cmd (C++ function), 18
- i2s\_lcd\_write\_command (C++ function), 18
- i2s\_lcd\_write\_data (C++ function), 18
- imu\_acquire (C++ function), 275
- imu\_acquire\_acce (C++ function), 274
- imu\_acquire\_gyro (C++ function), 274
- imu\_control (C++ function), 275
- imu\_create (C++ function), 274
- imu\_delete (C++ function), 274
- imu\_id\_t (C++ enum), 275
- imu\_id\_t::IMU\_MAX\_ID (C++ enumerator), 276
- imu\_id\_t::LIS2DH12\_ID (C++ enumerator), 276
- imu\_id\_t::MPU6050\_ID (C++ enumerator), 275
- imu\_sleep (C++ function), 275
- imu\_test (C++ function), 274
- imu\_wakeup (C++ function), 275
- ina236\_clear\_mask (C++ function), 283
- ina236\_config\_t (C++ struct), 283
- ina236\_config\_t::alert\_cb (C++ member),  
284
- ina236\_config\_t::alert\_en (C++ member),  
283
- ina236\_config\_t::alert\_pin (C++ member),  
283

- ina236\_config\_t::bus (C++ member), 283  
 ina236\_config\_t::dev\_addr (C++ member), 283  
 ina236\_create (C++ function), 282  
 ina236\_delete (C++ function), 283  
 ina236\_get\_current (C++ function), 283  
 ina236\_get\_voltage (C++ function), 283  
 ina236\_handle\_t (C++ type), 284  
 INA236\_I2C\_ADDRESS\_DEFAULT (C macro), 284  
 INJECT\_DUTY (C macro), 318  
 INJECT\_ENABLE (C macro), 318  
 int236\_alert\_cb\_t (C++ type), 284  
 iot\_button\_count\_cb (C++ function), 229  
 iot\_button\_count\_event (C++ function), 229  
 iot\_button\_create (C++ function), 228  
 iot\_button\_delete (C++ function), 228  
 iot\_button\_get\_event (C++ function), 229  
 iot\_button\_get\_key\_level (C++ function), 230  
 iot\_button\_get\_long\_press\_hold\_cnt (C++ function), 230  
 iot\_button\_get\_repeat (C++ function), 230  
 iot\_button\_get\_ticks\_time (C++ function), 230  
 iot\_button\_register\_cb (C++ function), 228  
 iot\_button\_register\_event\_cb (C++ function), 228  
 iot\_button\_resume (C++ function), 230  
 iot\_button\_set\_param (C++ function), 230  
 iot\_button\_stop (C++ function), 230  
 iot\_button\_unregister\_cb (C++ function), 229  
 iot\_button\_unregister\_event (C++ function), 229  
 iot\_knob\_clear\_count\_value (C++ function), 245  
 iot\_knob\_create (C++ function), 244  
 iot\_knob\_delete (C++ function), 244  
 iot\_knob\_get\_count\_value (C++ function), 245  
 iot\_knob\_get\_event (C++ function), 245  
 iot\_knob\_register\_cb (C++ function), 244  
 iot\_knob\_resume (C++ function), 245  
 iot\_knob\_stop (C++ function), 245  
 iot\_knob\_unregister\_cb (C++ function), 245  
 iot\_sensor\_create (C++ function), 268  
 iot\_sensor\_delete (C++ function), 268  
 iot\_sensor\_handler\_register (C++ function), 269  
 iot\_sensor\_handler\_register\_with\_type (C++ function), 269  
 iot\_sensor\_handler\_unregister (C++ function), 269  
 iot\_sensor\_handler\_unregister\_with\_type (C++ function), 269  
 iot\_sensor\_scan (C++ function), 268  
 iot\_sensor\_start (C++ function), 268  
 iot\_sensor\_stop (C++ function), 268  
 iot\_servo\_deinit (C++ function), 322  
 iot\_servo\_init (C++ function), 321  
 iot\_servo\_read\_angle (C++ function), 322  
 iot\_servo\_write\_angle (C++ function), 322  
 ir\_learn\_add\_list\_node (C++ function), 258  
 ir\_learn\_add\_sub\_list\_node (C++ function), 258  
 ir\_learn\_cfg\_t (C++ struct), 259  
 ir\_learn\_cfg\_t::callback (C++ member), 259  
 ir\_learn\_cfg\_t::clk\_src (C++ member), 259  
 ir\_learn\_cfg\_t::learn\_count (C++ member), 259  
 ir\_learn\_cfg\_t::learn\_gpio (C++ member), 259  
 ir\_learn\_cfg\_t::resolution (C++ member), 259  
 ir\_learn\_cfg\_t::task\_affinity (C++ member), 260  
 ir\_learn\_cfg\_t::task\_priority (C++ member), 259  
 ir\_learn\_cfg\_t::task\_stack (C++ member), 260  
 ir\_learn\_check\_valid (C++ function), 258  
 ir\_learn\_clean\_data (C++ function), 258  
 ir\_learn\_clean\_sub\_data (C++ function), 258  
 ir\_learn\_handle\_t (C++ type), 260  
 ir\_learn\_list\_t (C++ struct), 259  
 ir\_learn\_list\_t::cmd\_sub\_node (C++ member), 259  
 ir\_learn\_new (C++ function), 257  
 ir\_learn\_print\_raw (C++ function), 258  
 ir\_learn\_restart (C++ function), 257  
 ir\_learn\_result\_cb (C++ type), 260  
 ir\_learn\_state\_t (C++ enum), 260  
 ir\_learn\_state\_t::IR\_LEARN\_STATE\_END (C++ enumerator), 260  
 ir\_learn\_state\_t::IR\_LEARN\_STATE\_EXIT (C++ enumerator), 260  
 ir\_learn\_state\_t::IR\_LEARN\_STATE\_FAIL (C++ enumerator), 260  
 ir\_learn\_state\_t::IR\_LEARN\_STATE\_READY (C++ enumerator), 260  
 ir\_learn\_state\_t::IR\_LEARN\_STATE\_STEP (C++ enumerator), 260  
 ir\_learn\_stop (C++ function), 257  
 ir\_learn\_sub\_list\_t (C++ struct), 258  
 ir\_learn\_sub\_list\_t::symbols (C++ member), 259  
 ir\_learn\_sub\_list\_t::timediff (C++ member), 259
- ## K
- keyboard\_btn\_callback\_t (C++ type), 240  
 keyboard\_btn\_cb\_config\_t (C++ struct), 239  
 keyboard\_btn\_cb\_config\_t::callback (C++ member), 239

- keyboard\_btn\_cb\_config\_t::event (C++ member), 239  
 keyboard\_btn\_cb\_config\_t::event\_data (C++ member), 239  
 keyboard\_btn\_cb\_config\_t::user\_data (C++ member), 239  
 keyboard\_btn\_cb\_handle\_t (C++ type), 240  
 keyboard\_btn\_config\_t (C++ struct), 239  
 keyboard\_btn\_config\_t::active\_level (C++ member), 239  
 keyboard\_btn\_config\_t::core\_id (C++ member), 240  
 keyboard\_btn\_config\_t::debounce\_ticks (C++ member), 239  
 keyboard\_btn\_config\_t::enable\_power\_save (C++ member), 239  
 keyboard\_btn\_config\_t::input\_gpio\_num (C++ member), 239  
 keyboard\_btn\_config\_t::input\_gpios (C++ member), 239  
 keyboard\_btn\_config\_t::output\_gpio\_num (C++ member), 239  
 keyboard\_btn\_config\_t::output\_gpios (C++ member), 239  
 keyboard\_btn\_config\_t::priority (C++ member), 240  
 keyboard\_btn\_config\_t::ticks\_interval (C++ member), 239  
 keyboard\_btn\_data\_t (C++ struct), 238  
 keyboard\_btn\_data\_t::input\_index (C++ member), 238  
 keyboard\_btn\_data\_t::output\_index (C++ member), 238  
 keyboard\_btn\_event\_data\_t (C++ union), 237  
 keyboard\_btn\_event\_data\_t::combination (C++ member), 237  
 keyboard\_btn\_event\_data\_t::combination\_key\_data (C++ struct), 238  
 keyboard\_btn\_event\_data\_t::combination\_key\_data (C++ member), 238  
 keyboard\_btn\_event\_data\_t::combination\_key\_data (C++ member), 238  
 keyboard\_btn\_event\_t (C++ enum), 240  
 keyboard\_btn\_event\_t::KBD\_EVENT\_COMBINATION (C++ enumerator), 240  
 keyboard\_btn\_event\_t::KBD\_EVENT\_MAX (C++ enumerator), 240  
 keyboard\_btn\_event\_t::KBD\_EVENT\_PRESSED (C++ enumerator), 240  
 keyboard\_btn\_handle\_t (C++ type), 240  
 keyboard\_btn\_report\_t (C++ struct), 238  
 keyboard\_btn\_report\_t::key\_change\_num (C++ member), 238  
 keyboard\_btn\_report\_t::key\_data (C++ member), 238  
 keyboard\_btn\_report\_t::key\_pressed\_num (C++ member), 238  
 keyboard\_btn\_report\_t::key\_release\_data (C++ member), 238  
 keyboard\_btn\_report\_t::key\_release\_num (C++ member), 238  
 keyboard\_button\_create (C++ function), 236  
 keyboard\_button\_delete (C++ function), 236  
 keyboard\_button\_get\_gpio\_by\_index (C++ function), 237  
 keyboard\_button\_get\_index\_by\_gpio (C++ function), 237  
 keyboard\_button\_register\_cb (C++ function), 236  
 keyboard\_button\_unregister\_cb (C++ function), 237  
 knob\_cb\_t (C++ type), 246  
 knob\_config\_t (C++ struct), 245  
 knob\_config\_t::default\_direction (C++ member), 245  
 knob\_config\_t::enable\_power\_save (C++ member), 246  
 knob\_config\_t::gpio\_encoder\_a (C++ member), 246  
 knob\_config\_t::gpio\_encoder\_b (C++ member), 246  
 knob\_event\_t (C++ enum), 246  
 knob\_event\_t::KNOB\_EVENT\_MAX (C++ enumerator), 246  
 knob\_event\_t::KNOB\_H\_LIM (C++ enumerator), 246  
 knob\_event\_t::KNOB\_L\_LIM (C++ enumerator), 246  
 knob\_event\_t::KNOB\_LEFT (C++ enumerator), 246  
 knob\_event\_t::KNOB\_NONE (C++ enumerator), 246  
 knob\_event\_t::KNOB\_RIGHT (C++ enumerator), 246  
 knob\_event\_t::KNOB\_ZERO (C++ enumerator), 246  
 knob\_event\_t::KNOB\_ZERO (C++ enumerator), 246  
 knob\_event\_t (C++ type), 246  
 L::key\_num  
 LCD\_CMD\_LEV (C macro), 20  
 LCD\_DATA\_LEV (C macro), 20  
 led\_indicator\_config\_t (C++ struct), 137  
 led\_indicator\_config\_t::blink\_list\_num (C++ member), 138  
 led\_indicator\_config\_t::blink\_lists (C++ member), 138  
 led\_indicator\_config\_t::led\_indicator\_custom\_conf (C++ member), 138  
 led\_indicator\_config\_t::led\_indicator\_gpio\_config (C++ member), 137  
 led\_indicator\_config\_t::led\_indicator\_ledc\_config (C++ member), 137  
 led\_indicator\_config\_t::led\_indicator\_rgb\_config (C++ member), 137  
 led\_indicator\_config\_t::led\_indicator\_strips\_conf (C++ member), 137



led\_indicator\_config\_t::mode (C++ member), 137

led\_indicator\_config\_t::[anonymous] (C++ member), 138

led\_indicator\_create (C++ function), 134

led\_indicator\_delete (C++ function), 134

led\_indicator\_get\_brightness (C++ function), 135

led\_indicator\_get\_hsv (C++ function), 136

led\_indicator\_get\_rgb (C++ function), 136

led\_indicator\_handle\_t (C++ type), 138

led\_indicator\_mode\_t (C++ enum), 139

led\_indicator\_mode\_t::LED\_CUSTOM\_MODE (C++ enumerator), 139

led\_indicator\_mode\_t::LED\_GPIO\_MODE (C++ enumerator), 139

led\_indicator\_mode\_t::LED\_LEDC\_MODE (C++ enumerator), 139

led\_indicator\_mode\_t::LED\_RGB\_MODE (C++ enumerator), 139

led\_indicator\_mode\_t::LED\_STRIPS\_MODE (C++ enumerator), 139

led\_indicator\_preempt\_start (C++ function), 135

led\_indicator\_preempt\_stop (C++ function), 135

led\_indicator\_set\_brightness (C++ function), 135

led\_indicator\_set\_color\_temperature (C++ function), 136

led\_indicator\_set\_hsv (C++ function), 136

led\_indicator\_set\_on\_off (C++ function), 135

led\_indicator\_set\_rgb (C++ function), 136

led\_indicator\_start (C++ function), 134

led\_indicator\_stop (C++ function), 134

light\_sensor\_acquire (C++ function), 277

light\_sensor\_acquire\_light (C++ function), 277

light\_sensor\_acquire\_rgbw (C++ function), 277

light\_sensor\_acquire\_uv (C++ function), 277

light\_sensor\_control (C++ function), 278

light\_sensor\_create (C++ function), 276

light\_sensor\_delete (C++ function), 276

light\_sensor\_id\_t (C++ enum), 278

light\_sensor\_id\_t::BH1750\_ID (C++ enumerator), 278

light\_sensor\_id\_t::LIGHT\_MAX\_ID (C++ enumerator), 278

light\_sensor\_id\_t::VEML6040\_ID (C++ enumerator), 278

light\_sensor\_id\_t::VEML6075\_ID (C++ enumerator), 278

light\_sensor\_sleep (C++ function), 277

light\_sensor\_test (C++ function), 276

light\_sensor\_wakeup (C++ function), 277

## M

MAX\_BLE\_DEVNAME\_LEN (C macro), 31

MAX\_SPEED\_MEASUREMENT\_FACTOR (C macro), 320

MCPWM\_CLK\_SRC (C macro), 317

MCPWM\_PERIOD (C macro), 317

mic\_callback\_t (C++ type), 176

mic\_frame\_t (C++ struct), 172

mic\_frame\_t::bit\_resolution (C++ member), 173

mic\_frame\_t::data (C++ member), 173

mic\_frame\_t::data\_bytes (C++ member), 173

mic\_frame\_t::samples\_frequence (C++ member), 173

MIN (C macro), 32

mmap\_assets\_config\_t (C++ struct), 143

mmap\_assets\_config\_t::checksum (C++ member), 143

mmap\_assets\_config\_t::max\_files (C++ member), 143

mmap\_assets\_config\_t::partition\_label (C++ member), 143

mmap\_assets\_del (C++ function), 142

mmap\_assets\_get\_height (C++ function), 143

mmap\_assets\_get\_mem (C++ function), 142

mmap\_assets\_get\_name (C++ function), 142

mmap\_assets\_get\_size (C++ function), 143

mmap\_assets\_get\_width (C++ function), 143

mmap\_assets\_handle\_t (C++ type), 143

mmap\_assets\_new (C++ function), 142

## N

ntc\_circuit\_mode\_t (C++ enum), 282

ntc\_circuit\_mode\_t::CIRCUIT\_MODE\_NTC\_GND (C++ enumerator), 282

ntc\_circuit\_mode\_t::CIRCUIT\_MODE\_NTC\_VCC (C++ enumerator), 282

ntc\_config\_t (C++ struct), 281

ntc\_config\_t::atten (C++ member), 281

ntc\_config\_t::b\_value (C++ member), 281

ntc\_config\_t::channel (C++ member), 281

ntc\_config\_t::circuit\_mode (C++ member), 281

ntc\_config\_t::fixed\_ohm (C++ member), 281

ntc\_config\_t::r25\_ohm (C++ member), 281

ntc\_config\_t::unit (C++ member), 281

ntc\_config\_t::vdd\_mv (C++ member), 281

ntc\_dev\_create (C++ function), 280

ntc\_dev\_delete (C++ function), 281

ntc\_dev\_get\_adc\_handle (C++ function), 280

ntc\_dev\_get\_temperature (C++ function), 281

ntc\_device\_handle\_t (C++ type), 282

NULL\_I2C\_DEV\_ADDR (C macro), 14

NULL\_I2C\_MEM\_ADDR (C macro), 14

NULL\_SPI\_CS\_PIN (C macro), 17

## O

OpenAI (C++ struct), 216

Espressif Systems	372	Release master
-------------------	-----	----------------

- OpenAI\_Completion::setTopP (C++ member), 209
- OpenAI\_Completion::setUser (C++ member), 210
- OpenAI\_Completion\_t (C++ type), 217
- OpenAI\_Edit (C++ struct), 211
- OpenAI\_Edit::process (C++ member), 212
- OpenAI\_Edit::setModel (C++ member), 212
- OpenAI\_Edit::setN (C++ member), 212
- OpenAI\_Edit::setTemperature (C++ member), 212
- OpenAI\_Edit::setTopP (C++ member), 212
- OpenAI\_Edit\_t (C++ type), 217
- OpenAI\_EmbeddingData\_t (C++ struct), 206
- OpenAI\_EmbeddingData\_t::data (C++ member), 206
- OpenAI\_EmbeddingData\_t::len (C++ member), 206
- OpenAI\_EmbeddingResponse (C++ struct), 206
- OpenAI\_EmbeddingResponse::deleteResponse (C++ member), 206
- OpenAI\_EmbeddingResponse::getData (C++ member), 206
- OpenAI\_EmbeddingResponse::getError (C++ member), 206
- OpenAI\_EmbeddingResponse::getLen (C++ member), 206
- OpenAI\_EmbeddingResponse::getUsage (C++ member), 206
- OpenAI\_EmbeddingResponse\_t (C++ type), 217
- OpenAI\_Image\_Response\_Format (C++ enum), 218
- OpenAI\_Image\_Response\_Format::OPENAI\_IMAGE\_RESPONSE\_FORMAT\_JSON (C++ enumerator), 218
- OpenAI\_Image\_Response\_Format::OPENAI\_IMAGE\_RESPONSE\_FORMAT\_TEXT (C++ enumerator), 218
- OpenAI\_Image\_Size (C++ enum), 218
- OpenAI\_Image\_Size::OPENAI\_IMAGE\_SIZE\_1024x1024 (C++ enumerator), 218
- OpenAI\_Image\_Size::OPENAI\_IMAGE\_SIZE\_256x256 (C++ enumerator), 218
- OpenAI\_Image\_Size::OPENAI\_IMAGE\_SIZE\_512x512 (C++ enumerator), 218
- OpenAI\_ImageEdit (C++ struct), 214
- OpenAI\_ImageEdit::image (C++ member), 214
- OpenAI\_ImageEdit::setN (C++ member), 214
- OpenAI\_ImageEdit::setPrompt (C++ member), 214
- OpenAI\_ImageEdit::setResponseFormat (C++ member), 214
- OpenAI\_ImageEdit::setSize (C++ member), 214
- OpenAI\_ImageEdit::setUser (C++ member), 214
- OpenAI\_ImageEdit\_t (C++ type), 217
- OpenAI\_ImageGeneration (C++ struct), 212
- OpenAI\_ImageGeneration::prompt (C++ member), 213
- OpenAI\_ImageGeneration::setN (C++ member), 212
- OpenAI\_ImageGeneration::setResponseFormat (C++ member), 212
- OpenAI\_ImageGeneration::setSize (C++ member), 212
- OpenAI\_ImageGeneration::setUser (C++ member), 213
- OpenAI\_ImageGeneration\_t (C++ type), 217
- OpenAI\_ImageResponse (C++ struct), 207
- OpenAI\_ImageResponse::deleteResponse (C++ member), 208
- OpenAI\_ImageResponse::getData (C++ member), 207
- OpenAI\_ImageResponse::getError (C++ member), 207
- OpenAI\_ImageResponse::getLen (C++ member), 207
- OpenAI\_ImageResponse\_t (C++ type), 217
- OpenAI\_ImageVariation (C++ struct), 213
- OpenAI\_ImageVariation::image (C++ member), 213
- OpenAI\_ImageVariation::setN (C++ member), 213
- OpenAI\_ImageVariation::setResponseFormat (C++ member), 213
- OpenAI\_ImageVariation::setSize (C++ member), 213
- OpenAI\_ImageVariation::setUser (C++ member), 213
- OpenAI\_ImageVariation\_t (C++ type), 217
- OpenAI\_ModerationResponse (C++ struct), 207
- OpenAI\_ModerationResponse::deleteResponse (C++ member), 207
- OpenAI\_ModerationResponse::getData (C++ member), 207
- OpenAI\_ModerationResponse::getError (C++ member), 207
- OpenAI\_ModerationResponse::getLen (C++ member), 207
- OpenAI\_ModerationResponse\_t (C++ type), 217
- OpenAI\_SpeechResponse (C++ struct), 208
- OpenAI\_SpeechResponse::deleteResponse (C++ member), 209
- OpenAI\_SpeechResponse::getData (C++ member), 209
- OpenAI\_SpeechResponse::getLen (C++ member), 208
- OpenAI\_SpeechResponse\_t (C++ type), 217
- OpenAI\_StringResponse (C++ struct), 208
- OpenAI\_StringResponse::deleteResponse (C++ member), 208
- OpenAI\_StringResponse::getData (C++ member), 208
- OpenAI\_StringResponse::getError (C++ member), 208
- OpenAI\_StringResponse::getLen (C++ member), 208



member), 208  
 OpenAI\_StringResponse::getUsage (C++ member), 208  
 OpenAI\_StringResponse\_t (C++ type), 217  
 OpenAI\_t (C++ type), 217  
 OpenAIChangeBaseURL (C++ function), 206  
 OpenAICreate (C++ function), 205  
 OpenAIDelete (C++ function), 205

## P

POLE\_PAIR (C macro), 319  
 portTICK\_RATE\_MS (C macro), 14  
 proxy\_cb\_t (C++ type), 295  
 proxy\_config\_t (C++ struct), 293  
 proxy\_config\_t::baseline\_coef (C++ member), 294  
 proxy\_config\_t::channel\_list (C++ member), 293  
 proxy\_config\_t::channel\_num (C++ member), 293  
 proxy\_config\_t::debounce\_n (C++ member), 294  
 proxy\_config\_t::debounce\_p (C++ member), 294  
 proxy\_config\_t::gold\_value (C++ member), 294  
 proxy\_config\_t::hysteresis\_p (C++ member), 294  
 proxy\_config\_t::max\_p (C++ member), 294  
 proxy\_config\_t::meas\_count (C++ member), 293  
 proxy\_config\_t::min\_n (C++ member), 294  
 proxy\_config\_t::noise\_n (C++ member), 294  
 proxy\_config\_t::noise\_p (C++ member), 294  
 proxy\_config\_t::reset\_n (C++ member), 294  
 proxy\_config\_t::reset\_p (C++ member), 294  
 proxy\_config\_t::smooth\_coef (C++ member), 294  
 proxy\_config\_t::threshold\_n (C++ member), 294  
 proxy\_config\_t::threshold\_p (C++ member), 294  
 proxy\_evt\_t (C++ enum), 295  
 proxy\_evt\_t::PROXI\_EVT\_ACTIVE (C++ enumerator), 295  
 proxy\_evt\_t::PROXI\_EVT\_INACTIVE (C++ enumerator), 295  
 pwm\_audio\_channel\_t (C++ enum), 197  
 pwm\_audio\_channel\_t::PWM\_AUDIO\_CH\_MAX (C++ enumerator), 197  
 pwm\_audio\_channel\_t::PWM\_AUDIO\_CH\_MONO (C++ enumerator), 197  
 pwm\_audio\_channel\_t::PWM\_AUDIO\_CH\_STEREO (C++ enumerator), 197  
 pwm\_audio\_config\_t (C++ struct), 196  
 pwm\_audio\_config\_t::duty\_resolution (C++ member), 197

pwm\_audio\_config\_t::gpio\_num\_left (C++ member), 196  
 pwm\_audio\_config\_t::gpio\_num\_right (C++ member), 197  
 pwm\_audio\_config\_t::ledc\_channel\_left (C++ member), 197  
 pwm\_audio\_config\_t::ledc\_channel\_right (C++ member), 197  
 pwm\_audio\_config\_t::ledc\_timer\_sel (C++ member), 197  
 pwm\_audio\_config\_t::ringbuf\_len (C++ member), 197  
 pwm\_audio\_deinit (C++ function), 195  
 pwm\_audio\_get\_param (C++ function), 196  
 pwm\_audio\_get\_status (C++ function), 196  
 pwm\_audio\_get\_volume (C++ function), 196  
 pwm\_audio\_init (C++ function), 194  
 pwm\_audio\_set\_param (C++ function), 195  
 pwm\_audio\_set\_sample\_rate (C++ function), 196  
 pwm\_audio\_set\_volume (C++ function), 196  
 pwm\_audio\_start (C++ function), 195  
 pwm\_audio\_status\_t (C++ enum), 197  
 pwm\_audio\_status\_t::PWM\_AUDIO\_STATUS\_BUSY (C++ enumerator), 197  
 pwm\_audio\_status\_t::PWM\_AUDIO\_STATUS\_IDLE (C++ enumerator), 197  
 pwm\_audio\_status\_t::PWM\_AUDIO\_STATUS\_UN\_INIT (C++ enumerator), 197  
 pwm\_audio\_stop (C++ function), 195  
 pwm\_audio\_write (C++ function), 195  
 PWM\_DUTYCYCLE\_05 (C macro), 317  
 PWM\_DUTYCYCLE\_10 (C macro), 317  
 PWM\_DUTYCYCLE\_100 (C macro), 318  
 PWM\_DUTYCYCLE\_15 (C macro), 317  
 PWM\_DUTYCYCLE\_20 (C macro), 317  
 PWM\_DUTYCYCLE\_25 (C macro), 317  
 PWM\_DUTYCYCLE\_30 (C macro), 317  
 PWM\_DUTYCYCLE\_40 (C macro), 318  
 PWM\_DUTYCYCLE\_50 (C macro), 318  
 PWM\_DUTYCYCLE\_60 (C macro), 318  
 PWM\_DUTYCYCLE\_80 (C macro), 318  
 PWM\_DUTYCYCLE\_90 (C macro), 318  
 PWM\_MODE (C macro), 317

## R

RAMP\_DUTY\_END (C macro), 319  
 RAMP\_DUTY\_INC (C macro), 319  
 RAMP\_DUTY\_STA (C macro), 318  
 RAMP\_TIM\_END (C macro), 318  
 RAMP\_TIM\_STA (C macro), 318  
 RAMP\_TIM\_STEP (C macro), 318

## S

sensor\_command\_t (C++ enum), 265  
 sensor\_command\_t::COMMAND\_MAX (C++ enumerator), 265

sensor\_command\_t::COMMAND\_SELF\_TEST (C++ enumerator), 265  
 sensor\_command\_t::COMMAND\_SET\_MODE (C++ enumerator), 265  
 sensor\_command\_t::COMMAND\_SET\_ODR (C++ enumerator), 265  
 sensor\_command\_t::COMMAND\_SET\_POWER (C++ enumerator), 265  
 sensor\_command\_t::COMMAND\_SET\_RANGE (C++ enumerator), 265  
 sensor\_config\_t (C++ struct), 270  
 sensor\_config\_t::bus (C++ member), 270  
 sensor\_config\_t::intr\_pin (C++ member), 270  
 sensor\_config\_t::intr\_type (C++ member), 271  
 sensor\_config\_t::min\_delay (C++ member), 270  
 sensor\_config\_t::mode (C++ member), 270  
 sensor\_config\_t::range (C++ member), 270  
 sensor\_data\_event\_id\_t (C++ enum), 267  
 sensor\_data\_event\_id\_t::SENSOR\_ACCE\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_BARO\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_CURRENT\_DATA\_READY (C++ enumerator), 268  
 sensor\_data\_event\_id\_t::SENSOR\_EVENT\_ID\_END (C++ enumerator), 268  
 sensor\_data\_event\_id\_t::SENSOR\_FORCE\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_GYRO\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_HR\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_HUMI\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_LIGHT\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_MAG\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_NOISE\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_PROXI\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_RGBW\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_STEP\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_TEMP\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_TVOC\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_UV\_DATA\_READY (C++ enumerator), 267  
 sensor\_data\_event\_id\_t::SENSOR\_VOLTAGE\_DATA\_READY (C++ enumerator), 268  
 sensor\_data\_group\_t (C++ struct), 264  
 sensor\_data\_group\_t::number (C++ member), 264  
 sensor\_data\_group\_t::sensor\_data (C++ member), 264  
 sensor\_data\_t (C++ struct), 262  
 sensor\_data\_t::acce (C++ member), 263  
 sensor\_data\_t::baro (C++ member), 263  
 sensor\_data\_t::current (C++ member), 264  
 sensor\_data\_t::data (C++ member), 264  
 sensor\_data\_t::event\_id (C++ member), 263  
 sensor\_data\_t::force (C++ member), 264  
 sensor\_data\_t::gyro (C++ member), 263  
 sensor\_data\_t::hr (C++ member), 263  
 sensor\_data\_t::humidity (C++ member), 263  
 sensor\_data\_t::light (C++ member), 263  
 sensor\_data\_t::mag (C++ member), 263  
 sensor\_data\_t::min\_delay (C++ member), 263  
 sensor\_data\_t::noise (C++ member), 263  
 sensor\_data\_t::proximity (C++ member), 263  
 sensor\_data\_t::rgbw (C++ member), 263  
 sensor\_data\_t::sensor\_id (C++ member), 263  
 sensor\_data\_t::step (C++ member), 264  
 sensor\_data\_t::temperature (C++ member), 263  
 sensor\_data\_t::timestamp (C++ member), 262  
 sensor\_data\_t::tvoc (C++ member), 263  
 sensor\_data\_t::uv (C++ member), 263  
 sensor\_data\_t::voltage (C++ member), 264  
 sensor\_driver\_handle\_t (C++ type), 264  
 SENSOR\_EVENT\_ANY\_ID (C macro), 264  
 sensor\_event\_handler\_instance\_t (C++ type), 271  
 sensor\_event\_handler\_t (C++ type), 271  
 sensor\_event\_id\_t (C++ enum), 266  
 sensor\_event\_id\_t::SENSOR\_EVENT\_COMMON\_END (C++ enumerator), 266  
 sensor\_event\_id\_t::SENSOR\_STARTED (C++ enumerator), 266  
 sensor\_event\_id\_t::SENSOR\_STOPPED (C++ enumerator), 266  
 sensor\_handle\_t (C++ type), 271  
 sensor\_humidity\_handle\_t (C++ type), 273  
 sensor\_id\_t (C++ enum), 271  
 sensor\_imu\_handle\_t (C++ type), 275  
 sensor\_info\_t (C++ struct), 270  
 sensor\_info\_t::addr (C++ member), 270  
 sensor\_info\_t::desc (C++ member), 270  
 sensor\_info\_t::name (C++ member), 270  
 sensor\_info\_t::sensor\_id (C++ member), 270  
 sensor\_light\_handle\_t (C++ type), 278  
 sensor\_mode\_t (C++ enum), 266  
 sensor\_mode\_t::MODE\_DEFAULT (C++ enumerator), 266

- sensor\_mode\_t::MODE\_INTERRUPT (C++ enumerator), 266  
 sensor\_mode\_t::MODE\_MAX (C++ enumerator), 266  
 sensor\_mode\_t::MODE\_POLLING (C++ enumerator), 266  
 sensor\_power\_mode\_t (C++ enum), 265  
 sensor\_power\_mode\_t::POWER\_MAX (C++ enumerator), 266  
 sensor\_power\_mode\_t::POWER\_MODE\_SLEEP (C++ enumerator), 265  
 sensor\_power\_mode\_t::POWER\_MODE\_WAKEUP (C++ enumerator), 265  
 sensor\_range\_t (C++ enum), 266  
 sensor\_range\_t::RANGE\_DEFAULT (C++ enumerator), 266  
 sensor\_range\_t::RANGE\_MAX (C++ enumerator), 266  
 sensor\_range\_t::RANGE\_MEDIUM (C++ enumerator), 266  
 sensor\_range\_t::RANGE\_MIN (C++ enumerator), 266  
 sensor\_type\_t (C++ enum), 264  
 sensor\_type\_t::HUMITURE\_ID (C++ enumerator), 265  
 sensor\_type\_t::IMU\_ID (C++ enumerator), 265  
 sensor\_type\_t::LIGHT\_SENSOR\_ID (C++ enumerator), 265  
 sensor\_type\_t::NULL\_ID (C++ enumerator), 265  
 sensor\_type\_t::SENSOR\_TYPE\_MAX (C++ enumerator), 265  
 servo\_channel\_t (C++ struct), 322  
 servo\_channel\_t::ch (C++ member), 322  
 servo\_channel\_t::servo\_pin (C++ member), 322  
 servo\_config\_t (C++ struct), 323  
 servo\_config\_t::channel\_number (C++ member), 323  
 servo\_config\_t::channels (C++ member), 323  
 servo\_config\_t::freq (C++ member), 323  
 servo\_config\_t::max\_angle (C++ member), 323  
 servo\_config\_t::max\_width\_us (C++ member), 323  
 servo\_config\_t::min\_width\_us (C++ member), 323  
 servo\_config\_t::timer\_number (C++ member), 323  
 SPEED\_CAL\_TYPE (C macro), 320  
 SPEED\_KD (C macro), 319  
 SPEED\_KI (C macro), 319  
 SPEED\_KP (C macro), 319  
 SPEED\_MAX\_INTEGRAL (C macro), 320  
 SPEED\_MAX\_OUTPUT (C macro), 320  
 SPEED\_MAX\_RPM (C macro), 320  
 SPEED\_MIN\_INTEGRAL (C macro), 320  
 SPEED\_MIN\_OUTPUT (C macro), 320  
 SPEED\_MIN\_RPM (C macro), 320  
 speed\_mode\_t (C++ enum), 316  
 speed\_mode\_t::SPEED\_CLOSED\_LOOP (C++ enumerator), 316  
 speed\_mode\_t::SPEED\_OPEN\_LOOP (C++ enumerator), 316  
 spi\_bus\_create (C++ function), 15  
 spi\_bus\_delete (C++ function), 15  
 spi\_bus\_device\_create (C++ function), 15  
 spi\_bus\_device\_delete (C++ function), 15  
 spi\_bus\_device\_handle\_t (C++ type), 17  
 spi\_bus\_handle\_t (C++ type), 17  
 spi\_bus\_transfer\_byte (C++ function), 15  
 spi\_bus\_transfer\_bytes (C++ function), 16  
 spi\_bus\_transfer\_reg16 (C++ function), 16  
 spi\_bus\_transfer\_reg32 (C++ function), 16  
 spi\_bus\_transmit\_begin (C++ function), 16  
 spi\_config\_t (C++ struct), 17  
 spi\_config\_t::max\_transfer\_sz (C++ member), 17  
 spi\_config\_t::miso\_io\_num (C++ member), 17  
 spi\_config\_t::mosi\_io\_num (C++ member), 17  
 spi\_config\_t::sclk\_io\_num (C++ member), 17  
 spi\_device\_config\_t (C++ struct), 17  
 spi\_device\_config\_t::clock\_speed\_hz (C++ member), 17  
 spi\_device\_config\_t::cs\_io\_num (C++ member), 17  
 spi\_device\_config\_t::mode (C++ member), 17  
 state\_callback\_t (C++ type), 176  
 stream\_ctrl\_t (C++ enum), 177  
 stream\_ctrl\_t::CTRL\_MAX (C++ enumerator), 177  
 stream\_ctrl\_t::CTRL\_NONE (C++ enumerator), 177  
 stream\_ctrl\_t::CTRL\_RESUME (C++ enumerator), 177  
 stream\_ctrl\_t::CTRL\_SUSPEND (C++ enumerator), 177  
 stream\_ctrl\_t::CTRL\_UAC\_MUTE (C++ enumerator), 177  
 stream\_ctrl\_t::CTRL\_UAC\_VOLUME (C++ enumerator), 177
- ## T
- TOUCH\_MAX\_POINT\_NUMBER (C macro), 251  
 touch\_panel\_config\_t (C++ struct), 249  
 touch\_panel\_config\_t::clk\_freq (C++ member), 249  
 touch\_panel\_config\_t::direction (C++ member), 250  
 touch\_panel\_config\_t::height (C++ member), 250

- touch\_panel\_config\_t::i2c\_addr (C++ member), 249
  - touch\_panel\_config\_t::i2c\_bus (C++ member), 249
  - touch\_panel\_config\_t::interface\_i2c (C++ member), 249
  - touch\_panel\_config\_t::interface\_spi (C++ member), 250
  - touch\_panel\_config\_t::interface\_type (C++ member), 250
  - touch\_panel\_config\_t::pin\_num\_cs (C++ member), 250
  - touch\_panel\_config\_t::pin\_num\_int (C++ member), 250
  - touch\_panel\_config\_t::spi\_bus (C++ member), 250
  - touch\_panel\_config\_t::width (C++ member), 250
  - touch\_panel\_config\_t::[anonymous] (C++ member), 250
  - touch\_panel\_controller\_t (C++ enum), 252
  - touch\_panel\_controller\_t::TOUCH\_PANEL\_CONTROLLER\_FT5X06 (C++ enumerator), 252
  - touch\_panel\_controller\_t::TOUCH\_PANEL\_CONTROLLER\_MS2016 (C++ enumerator), 253
  - touch\_panel\_controller\_t::TOUCH\_PANEL\_CONTROLLER\_MT2016 (C++ enumerator), 253
  - touch\_panel\_dir\_t (C++ enum), 251
  - touch\_panel\_dir\_t::TOUCH\_DIR\_BTLLR (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_DIR\_BTRL (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_DIR\_LRBT (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_DIR\_LRTB (C++ enumerator), 251
  - touch\_panel\_dir\_t::TOUCH\_DIR\_MAX (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_DIR\_RLBT (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_DIR\_RLTB (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_DIR\_TBLR (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_DIR\_TBRL (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_MIRROR\_X (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_MIRROR\_Y (C++ enumerator), 252
  - touch\_panel\_dir\_t::TOUCH\_SWAP\_XY (C++ enumerator), 252
  - touch\_panel\_driver\_t (C++ struct), 250
  - touch\_panel\_driver\_t::calibration\_run (C++ member), 251
  - touch\_panel\_driver\_t::deinit (C++ member), 250
  - touch\_panel\_driver\_t::init (C++ member), 250
  - touch\_panel\_driver\_t::read\_point\_data (C++ member), 251
  - touch\_panel\_driver\_t::set\_direction (C++ member), 251
  - touch\_panel\_event\_t (C++ enum), 251
  - touch\_panel\_event\_t::TOUCH\_EVT\_PRESS (C++ enumerator), 251
  - touch\_panel\_event\_t::TOUCH\_EVT\_RELEASE (C++ enumerator), 251
  - touch\_panel\_find\_driver (C++ function), 249
  - touch\_panel\_interface\_type\_t (C++ enum), 252
  - touch\_panel\_interface\_type\_t::TOUCH\_PANEL\_IFACE\_I (C++ enumerator), 252
  - touch\_panel\_interface\_type\_t::TOUCH\_PANEL\_IFACE\_S (C++ enumerator), 252
  - touch\_panel\_points\_t (C++ struct), 249
  - touch\_panel\_points\_t::curx (C++ member), 249
  - touch\_panel\_points\_t::cury (C++ member), 249
  - touch\_panel\_points\_t::event (C++ member), 249
  - touch\_panel\_points\_t::point\_num (C++ member), 249
  - touch\_proximity\_handle\_t (C++ type), 295
  - TOUCH\_PROXIMITY\_NUM\_MAX (C macro), 294
  - touch\_proximity\_sensor\_create (C++ function), 290
  - touch\_proximity\_sensor\_delete (C++ function), 293
  - touch\_proximity\_sensor\_start (C++ function), 293
  - touch\_proximity\_sensor\_stop (C++ function), 293
- ## U
- UAC\_BITS\_ANY (C macro), 175
  - UAC\_CH\_ANY (C macro), 175
  - uac\_config\_t (C++ struct), 174
  - uac\_config\_t::ac\_interface (C++ member), 175
  - uac\_config\_t::flags (C++ member), 175
  - uac\_config\_t::mic\_bit\_resolution (C++ member), 174
  - uac\_config\_t::mic\_buf\_size (C++ member), 174
  - uac\_config\_t::mic\_cb (C++ member), 174
  - uac\_config\_t::mic\_cb\_arg (C++ member), 174
  - uac\_config\_t::mic\_ch\_num (C++ member), 174
  - uac\_config\_t::mic\_ep\_addr (C++ member), 175
  - uac\_config\_t::mic\_ep\_mps (C++ member), 175
  - uac\_config\_t::mic\_fu\_id (C++ member), 175

- `uac_config_t::mic_interface` (C++ member), 175
- `uac_config_t::mic_samples_frequence` (C++ member), 174
- `uac_config_t::spk_bit_resolution` (C++ member), 174
- `uac_config_t::spk_buf_size` (C++ member), 174
- `uac_config_t::spk_ch_num` (C++ member), 174
- `uac_config_t::spk_ep_addr` (C++ member), 175
- `uac_config_t::spk_ep_mps` (C++ member), 175
- `uac_config_t::spk_fu_id` (C++ member), 175
- `uac_config_t::spk_interface` (C++ member), 175
- `uac_config_t::spk_samples_frequence` (C++ member), 174
- `uac_device_config_t` (C++ struct), 191
- `uac_device_config_t::cb_ctx` (C++ member), 191
- `uac_device_config_t::input_cb` (C++ member), 191
- `uac_device_config_t::output_cb` (C++ member), 191
- `uac_device_config_t::set_mute_cb` (C++ member), 191
- `uac_device_config_t::set_volume_cb` (C++ member), 191
- `uac_device_config_t::skip_tinyusb_init` (C++ member), 191
- `uac_device_init` (C++ function), 190
- `uac_frame_size_list_get` (C++ function), 170
- `uac_frame_size_reset` (C++ function), 171
- `uac_frame_size_t` (C++ struct), 173
- `uac_frame_size_t::bit_resolution` (C++ member), 174
- `uac_frame_size_t::ch_num` (C++ member), 173
- `uac_frame_size_t::samples_frequence` (C++ member), 174
- `uac_frame_size_t::samples_frequence_max` (C++ member), 174
- `uac_frame_size_t::samples_frequence_min` (C++ member), 174
- `UAC_FREQUENCY_ANY` (C macro), 175
- `uac_input_cb_t` (C++ type), 191
- `uac_mic_streaming_read` (C++ function), 170
- `uac_output_cb_t` (C++ type), 191
- `uac_set_mute_cb_t` (C++ type), 191
- `uac_set_volume_cb_t` (C++ type), 191
- `uac_spk_streaming_write` (C++ function), 170
- `uac_streaming_config` (C++ function), 169
- `uint24_t` (C++ struct), 40
- `uint24_t::u24` (C++ member), 40
- `usb_stream_state_t` (C++ enum), 177
- `usb_stream_state_t::STREAM_CONNECTED` (C++ enumerator), 177
- `usb_stream_state_t::STREAM_DISCONNECTED` (C++ enumerator), 177
- `usb_stream_t` (C++ enum), 177
- `usb_stream_t::STREAM_MAX` (C++ enumerator), 177
- `usb_stream_t::STREAM_UAC_MIC` (C++ enumerator), 177
- `usb_stream_t::STREAM_UAC_SPK` (C++ enumerator), 177
- `usb_stream_t::STREAM_UVC` (C++ enumerator), 177
- `usb_streaming_connect_wait` (C++ function), 169
- `usb_streaming_control` (C++ function), 170
- `usb_streaming_start` (C++ function), 169
- `usb_streaming_state_register` (C++ function), 170
- `usb_streaming_stop` (C++ function), 169
- `usbh_cdc_cb_t` (C++ type), 186
- `usbh_cdc_config` (C++ struct), 184
- `usbh_cdc_config::bulk_in_ep` (C++ member), 185
- `usbh_cdc_config::bulk_in_ep_addr` (C++ member), 184
- `usbh_cdc_config::bulk_in_ep_addrs` (C++ member), 185
- `usbh_cdc_config::bulk_in_eps` (C++ member), 185
- `usbh_cdc_config::bulk_out_ep` (C++ member), 185
- `usbh_cdc_config::bulk_out_ep_addr` (C++ member), 185
- `usbh_cdc_config::bulk_out_ep_addrs` (C++ member), 185
- `usbh_cdc_config::bulk_out_eps` (C++ member), 185
- `usbh_cdc_config::conn_callback` (C++ member), 185
- `usbh_cdc_config::conn_callback_arg` (C++ member), 185
- `usbh_cdc_config::disconn_callback` (C++ member), 185
- `usbh_cdc_config::disconn_callback_arg` (C++ member), 186
- `usbh_cdc_config::itf_num` (C++ member), 184
- `usbh_cdc_config::rx_buffer_size` (C++ member), 185
- `usbh_cdc_config::rx_buffer_sizes` (C++ member), 185
- `usbh_cdc_config::rx_callback` (C++ member), 185
- `usbh_cdc_config::rx_callback_arg` (C++ member), 186
- `usbh_cdc_config::rx_callback_args` (C++ member), 186
- `usbh_cdc_config::rx_callbacks` (C++ member), 186



- member*), 185
  - usbh\_cdc\_config::tx\_buffer\_size (C++ *member*), 185
  - usbh\_cdc\_config::tx\_buffer\_sizes (C++ *member*), 185
  - usbh\_cdc\_config\_t (C++ *type*), 186
  - usbh\_cdc\_driver\_delete (C++ *function*), 183
  - usbh\_cdc\_driver\_install (C++ *function*), 183
  - usbh\_cdc\_flush\_rx\_buffer (C++ *function*), 184
  - usbh\_cdc\_flush\_tx\_buffer (C++ *function*), 184
  - usbh\_cdc\_get\_buffered\_data\_len (C++ *function*), 183
  - usbh\_cdc\_get\_itf\_state (C++ *function*), 184
  - usbh\_cdc\_itf\_get\_buffered\_data\_len (C++ *function*), 183
  - usbh\_cdc\_itf\_read\_bytes (C++ *function*), 184
  - usbh\_cdc\_itf\_write\_bytes (C++ *function*), 183
  - usbh\_cdc\_read\_bytes (C++ *function*), 184
  - usbh\_cdc\_wait\_connect (C++ *function*), 183
  - usbh\_cdc\_write\_bytes (C++ *function*), 183
  - uvc\_config (C++ *struct*), 171
  - uvc\_config::ep\_addr (C++ *member*), 172
  - uvc\_config::ep\_mps (C++ *member*), 172
  - uvc\_config::flags (C++ *member*), 172
  - uvc\_config::format (C++ *member*), 172
  - uvc\_config::format\_index (C++ *member*), 172
  - uvc\_config::frame\_buffer (C++ *member*), 172
  - uvc\_config::frame\_buffer\_size (C++ *member*), 172
  - uvc\_config::frame\_cb (C++ *member*), 172
  - uvc\_config::frame\_cb\_arg (C++ *member*), 172
  - uvc\_config::frame\_height (C++ *member*), 171
  - uvc\_config::frame\_index (C++ *member*), 172
  - uvc\_config::frame\_interval (C++ *member*), 171
  - uvc\_config::frame\_width (C++ *member*), 171
  - uvc\_config::interface (C++ *member*), 172
  - uvc\_config::interface\_alt (C++ *member*), 172
  - uvc\_config::xfer\_buffer\_a (C++ *member*), 172
  - uvc\_config::xfer\_buffer\_b (C++ *member*), 172
  - uvc\_config::xfer\_buffer\_size (C++ *member*), 171
  - uvc\_config::xfer\_type (C++ *member*), 172
  - uvc\_config\_t (C++ *type*), 176
  - uvc\_device\_config (C++ *function*), 188
  - uvc\_device\_config\_t (C++ *struct*), 188
  - uvc\_device\_config\_t::cb\_ctx (C++ *member*), 189
  - uvc\_device\_config\_t::fb\_get\_cb (C++ *member*), 189
  - uvc\_device\_config\_t::fb\_return\_cb (C++ *member*), 189
  - uvc\_device\_config\_t::start\_cb (C++ *member*), 188
  - uvc\_device\_config\_t::stop\_cb (C++ *member*), 189
  - uvc\_device\_config\_t::uvc\_buffer (C++ *member*), 188
  - uvc\_device\_config\_t::uvc\_buffer\_size (C++ *member*), 188
  - uvc\_device\_init (C++ *function*), 188
  - uvc\_fb\_t (C++ *struct*), 188
  - uvc\_fb\_t::buf (C++ *member*), 188
  - uvc\_fb\_t::format (C++ *member*), 188
  - uvc\_fb\_t::height (C++ *member*), 188
  - uvc\_fb\_t::len (C++ *member*), 188
  - uvc\_fb\_t::timestamp (C++ *member*), 188
  - uvc\_fb\_t::width (C++ *member*), 188
  - uvc\_format\_t (C++ *enum*), 189
  - uvc\_format\_t::UVC\_FORMAT\_H264 (C++ *enumerator*), 189
  - uvc\_format\_t::UVC\_FORMAT\_JPEG (C++ *enumerator*), 189
  - uvc\_frame\_size\_list\_get (C++ *function*), 171
  - uvc\_frame\_size\_reset (C++ *function*), 171
  - uvc\_frame\_size\_t (C++ *struct*), 173
  - uvc\_frame\_size\_t::height (C++ *member*), 173
  - uvc\_frame\_size\_t::interval (C++ *member*), 173
  - uvc\_frame\_size\_t::interval\_max (C++ *member*), 173
  - uvc\_frame\_size\_t::interval\_min (C++ *member*), 173
  - uvc\_frame\_size\_t::interval\_step (C++ *member*), 173
  - uvc\_frame\_size\_t::width (C++ *member*), 173
  - uvc\_input\_fb\_get\_cb\_t (C++ *type*), 189
  - uvc\_input\_fb\_return\_cb\_t (C++ *type*), 189
  - uvc\_input\_start\_cb\_t (C++ *type*), 189
  - uvc\_input\_stop\_cb\_t (C++ *type*), 189
  - uvc\_streaming\_config (C++ *function*), 169
  - uvc\_xfer\_t (C++ *enum*), 176
  - uvc\_xfer\_t::UVC\_XFER\_BULK (C++ *enumerator*), 176
  - uvc\_xfer\_t::UVC\_XFER\_ISOC (C++ *enumerator*), 176
  - uvc\_xfer\_t::UVC\_XFER\_UNKNOWN (C++ *enumerator*), 176
- ## Z
- ZERO\_CROSS\_ADVANCE (C *macro*), 319
  - zero\_cross\_cb\_t (C++ *type*), 345
  - ZERO\_CROSS\_DETECTION\_ACCURACY (C *macro*), 319
  - zero\_detect\_cb\_param\_t (C++ *union*), 342

